FACULTAD DE ESTUDIOS ESTADÍSTICOS

MÁSTER EN MINERÍA DE DATOS E INTELIGENCIA DE NEGOCIOS

Curso 2020/2021

Trabajo de Fin de Máster

TITULO: Desarrollo de un modelo de Lead Scoring

***Alumno*: Adrián Ureña**

***Tutor*: Ramón Alberto Carrasco González**

Septiembre de 2021

UNIVERSIDAD COMPLUTENSE
MADRID

Índice de contenido

1. Introducción 5
2. Objetivos 7
3. Metodología y arquitectura del sistema 8
4. Descripción del conjunto de datos. 12
5. Depuración de datos 14
   5.1. Depuración en Miner 14
   5.2. Pre-procesado en R 16
   5.3. Selección de variables 19
6. Modelización 25
   6.1. Árbol de decisión 26
   6.2. Regresión Logística 28
   6.3. Redes Neuronales 34
   6.4. Random Forest 44
   6.5. Modelos de Gradient Boosting 50
      6.5.1. Gradient Boosting (GBM) 50
      6.5.2. Extreme Gradient Boosting (XGBoost) 55
   6.6. Support Vector Machines 57
7. Evaluación de modelos y construcción de un modelo de Stacking 61
   7.1. Selección del mejor modelo 61
   7.2. Construcción de un modelo de Stacking 63
8. Diferencias con la solución propuesta en Kaggle 72
9. Conclusiones 74
10. Trabajo Futuro 76
11. Bibliografía 78

Índice de figuras

Figura 1: Esquema del proceso de lead scoring. 5
Figura 2: Esquema de la metodología propuesta. 11
Figura 3: Diagrama de la depuración realizada en Miner. 14
Figura 4: resultado de la función summary sobre el conjunto de datos. 16
Figura 5: summary con variables renombradas. 17
Figura 6: Resultado del árbol de decisión 27
Figura 7: Representación de un problema clásico de clasificación con separación lineal. 28
Figura 8: Diagrama de cajas de comparación entre modelos de regresión. 33
Figura 9: Esquema de la estructura de una red neuronal clásica. 35
Figura 10: Representación del concepto general de early stopping (izquierda: convergencia sin sobreajuste, derecha: convergencia con sobreajuste tras el punto de early stopping). 37
Figura 11: Representación del proceso de convergencia con early stopping en base a la medición del AUC. 37
Figura 12: Diagrama de cajas para estudio del número de nodos en redes neuronales 41
Figura 13. Diagrama de cajas del estudio de la tasa de aprendizaje en redes neuronales 42
Figura 14: Diagrama de cajas de las seis mejores combinaciones de redes neuronales 43
Figura 15: Esquema de funcionamiento de random forest. Puntos negors: representan los valores reales de la variable objetivo. Líneas horizontales: son las predicciones obtenidas con cada árbol de decisión. Curva roja: predicción media. 45
Figura 16. Esquema del sistema de sorteo de observaciones y variables en random forest. 45
Figura 17: Variación del AUC en entrenamiento y prueba en función del número de árboles en random forest 47
Figura 18: Gráfico de variación del AUC en función del número de observaciones incluidas en cada árbol en random forest. 48
Figura 19: Diagrama de cajas de comparación del AUC frente al número de variables incluidas en cada árbol en random forest 49
Figura 20: Diagrama de cajas de comparación del AUC frente al valor del shrinkage en gradient boosting. 51
Figura 21: Diagrama de cajas de comparación del AUC frente al valor del número de árboles en gradient boosting. 52
Figura 22: Diagrama de cajas de comparación del AUC frente al número de observaciones mínimas por nodo en GBM. 53
Figura 23: Diagrama de cajas de comparación del AUC frente a las mejores combinaciones de parámetros posibles en modelos de GBM.. 54
Figura 24: Comparación del AUC entre diferentes combinaciones del modelo XGBoost. 56

Figura 25: Esquema del "maximal margin" usado en SVM. 57
Figura 26: Diagrama de cajas de comparación del AUC frente a la constante C en SVM. 58
Figura 27: Diagrama de cajas de comparación del AUC frente a los combinaciones con diferentes tipos de kernel en SVM. 59
Figura 28: Diagrama de cajas del AUC frente a los efectos del valor de sigma en SVM radial. 60
Figura 29: Diagrama de cajas de comparación final entre los mejores modelos construidos con cada tipo de algoritmo. 62
Figura 30: Esquema de funcionamiento de Ensembled Voting 63
Figura 31: Esquema de funcionamiento de los modelos de stacking. 64
Figura 32: Resumen de la estructura de los modelos de stacking desarrollados. 65
Figura 33: Comparación del AUC entre los mejores modelos desarrollados y los tres modelos de stacking construidos en base a estos. 66
Figura 34: Agrupación de contactos por probabilidad media de cada grupo. 70
Figura 35: Agrupación de contactos por probabilidad media de cada grupo (tres grupos). 71

1. Introducción

La evolución de la ciencia de datos ha permitido desarrollar gran cantidad de nuevas estrategias enfocadas al marketing. Y es que la gran cantidad de datos accesibles para las empresas hoy en día hace posible conocer quiénes son nuestros clientes y cómo se comportan, explotando al máximo las oportunidades de negocio y mejorando la experiencia en todos los sentidos.

Entre las estrategias de automatización del marketing más comunes se encuentra el *Inbound Marketing* (*Lead nurturing y lead scoring en el inbound marketing - InboundCycle.pdf*, s. f.). Este tipo de estrategias están enfocadas en reconocer y acompañar al cliente desde el punto inicial de contacto hasta el final del proceso, siendo este el punto en el que se produce la compra (punto de conversión).

Dentro de este tipo de enfoque se encuentra el *Lead Scoring* (Barnhard, 2020). Esta técnica se usa principalmente en los medios de *e-commerce*, y consiste en evaluar numéricamente a nuestros posibles compradores, intentando reconocer aquellos patrones (variables) más influyentes en su tendencia al proceso de conversión. En otras palabras, las técnicas de lead scoring tienen como objetivo puntuar a los compradores potenciales según la probabilidad de que, finalmente, realicen la compra (Porras Blanco, 2018).

De esta forma, se identificará a cada contacto con una temperatura, que no será más que la probabilidad de conversión, diferenciando entre *hot leads* y *cold leads.*

Para obtener dichas temperaturas se hace uso de las técnicas supervisadas de aprendizaje automático (o *Machine Learning*), que son capaces de interiorizar los patrones ocultos tras los datos de comportamiento de cada individuo. Esto es lo que se conoce como proceso de entrenamiento. El proceso se esquematiza en la Figura 1.

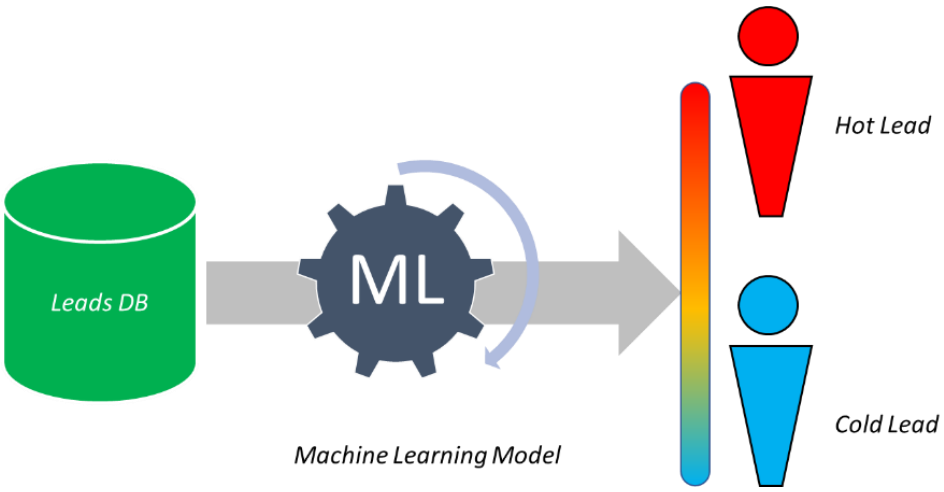


Figura 1: Esquema del proceso de lead scoring.

Desde un punto de vista más matemático, las técnicas de machine learning son una serie de algoritmos que, partiendo de una serie de datos almacenados en forma de

variables x_i, son capaces de definir una función que predice el valor de una variable y. Dicha predicción, $\hat{y}$, tendrá implícito un error $|y - \hat{y}|$. Los mejores modelos de predicción, por tanto, conllevarán a un error de predicción más bajo, tanto en términos medios como en los de su varianza.
El funcionamiento de cada algoritmo varía mucho de uno a otro, pudiendo estar basado en modelos más sencillos como los árboles de decisión (caso de *Random Forest* o *Gradient Boosting*), puede tratarse de una función compleja como en el caso de las redes neuronales etc.

En el presente trabajo se expondrá un ejemplo del proceso de lead scoring sobre un conjunto de datos particular. Este modelo se ha obtenido de la página web *Kaggle* (*Lead Scoring (Logistic Regression)*, s. f.). Para ello, se ahondará en diversos algoritmos de machine learning tratados durante el curso y se llegará a unos resultados finales sobre su utilidad y utilización.

Además, a lo largo del trabajo se intentará tener en cuenta la metodología CRISP-DM, enfocando cada punto del proceso desde el punto de vista del negocio. Al tratarse de un ejemplo puramente académico, se harán suposiciones sobre lo que se debería hacer en un caso real, pero no se podrán replicar las estructuras o procesos propios de una empresa al no disponer de los recursos necesarios.

2. Objetivos

Los objetivos que se intentarán atacar en el desarrollo de este trabajo son:

- Explorar, conocer y familiarizarse con el conjunto de datos, intentando comprender el origen y significado de cada variable, así como su influencia en los modelos de predicción.

- Familiarizarse con conceptos puros de marketing, como puede ser contacto, lead, estrategias de captación e intentar enfocar los resultados como si se tratase de un caso real aplicado al departamento de *Data Science* dentro del área de Marketing de una empresa.

- Conocer y emplear las técnicas de minería de datos más habituales a la hora depurar un conjunto de datos (tratamiento de datos ausentes, atípicos, creación de *dummies* etc).

- Profundizar en los principales algoritmos de Machine Learning, intentando encontrar la mejor opción posible dentro de cada algoritmo. Intentar entender por qué cada algoritmo se comporta de determinada manera frente al *dataset* propuesto.

- Discernir qué tipo de algoritmo se ajusta mejor al resultado buscado, desde un punto de vista estadístico, pero sobre todo intentando seguir una perspectiva práctica teniendo en cuenta su implementación en un caso real, su interpretabilidad, posibilidad de desarrollo etc.

- Comentar las diferencias entre el desarrollo realizado y la solución obtenida con respecto al modelo de lead scoring de *Kaggle*.

- Finalmente, intentar dilucidar si los resultados obtenidos cumplen con el propósito del trabajo, si se han logrado el resto de objetivos y determinar los puntos de mejora y posibilidades para seguir trabajando en este problema en proyectos futuros.

3. Metodología y arquitectura del sistema

La metodología que se va a seguir es CRISP-DM. Esta metodología aboga por implementar la ciencia de datos teniendo siempre en cuenta el punto de vista del negocio. Para ello se basa en cinco pilares fundamentales: entender el negocio, entender el modelo de datos, preparar dichos datos, realizar el correspondiente análisis o modelización, evaluar su implantación y poner en producción el resultado. Se trata de un proceso cíclico, que se debe ir refinando y perfeccionado, y que requiere de un completo conocimiento tanto de la parte más puramente técnica como de la parte operativa de la empresa.

Además, aplicar dicha metodología requiere que la empresa posea un ecosistema adaptado a la ciencia de datos. Es decir, tener completa disponibilidad de todos los sistemas, tablas, bases de datos y procesos de la organización por parte del equipo de científicos de datos; de tal forma que puedan acceder a los mismos ágil, intuitivamente y de forma independiente al equipo de IT. Esto implica establecer una arquitectura compatible con los sistemas de datos de los que dispone el negocio.

Al final, el trabajo del científico de datos no es sólo aplicar modelos de Machine Learning ni encontrar relaciones entre variables. Una parte fundamental consiste en manejar los sistemas de la organización y comprender todas las relaciones entre las bases de datos y la parte puramente operacional. Y es que la situación típica a la que debe enfrenarse el científico de datos suele ser desfavorable, teniendo que aplicar todos los conocimientos disponibles sobre sistemas informáticos, servidores, computación en la nube, conexión con las fuentes, así como empaparse de toda la parte de negocio.

En este trabajo no se va a tratar directamente la parte operacional de la empresa, puesto que cada una tiene su propio ámbito de actuación y sería un ejercicio poco representativo, pero sí se va a dar un pequeño esbozo de la arquitectura que debería tener un buen sistema de datos que permita realizar las labores del científico de datos.

Una buena arquitectura debe contener los siguientes elementos básicos:

- Una base de datos dedicada a la analítica, que replique de la base de datos operacional o transaccional. Es lo que se llamará *Data Warehouse* (DWH). Esta gran base de datos será gestionada con un gestor de bases de datos como por ejemplo *SQL Server Management Studio*. Además, se evitará en toda medida de lo posible realizar modificaciones en las tablas operacionales, puesto que es información sensible que recae fuera del ámbito del área de ciencia de datos.

- Un gestor o planificador de tareas que permita el lanzamiento de procesos automatizados (*pipelines* y *triggers*). El objetivo de este planificador es lanzar los procesos *ETL* (extracción, transformación y carga) desde el operacional al DWH. También se realizará el trasvase en la dirección opuesta (siempre con la supervisión del equipo de IT), subiendo los resultados de las predicciones o automatizaciones realizadas desde el DWH hasta el operacional. Un ejemplo de

este tipo de planificadores es *Azure Data Factory*, que permite lanzar pipelines, ejecutar procedimientos almacenados en SQL Server y ejecutar scripts de Python y R a través de la rama de *Logic Apps*.

- Una estación de computación (máquina virtual o física) para realizar las tareas de cálculo y procesamiento de datos. Normalmente los volúmenes de datos que se van a tratar son excesivamente pesados como para ser tratados en la máquina del científico de datos. Estas máquinas destinadas a la computación suelen estar configuradas con el sistema operativo Linux, puesto que reduce sensiblemente la cantidad de recursos dedicados a la interfaz gráfica, destinando la mayoría de la memoria a tareas de cálculo. Es aquí donde se ejecutan los scripts asociados a las tareas de entrenamiento de modelos en R o Python. Además, para facilitar la gestión y visualización de resultados de este tipo de máquinas desde sistemas Windows se hace uso de software de control como *Putty* o *MobaXterm*.

- Software para las tareas propias de desarrollo del científico de datos. Aquí están incluidas tanto las suites para manejar los lenguajes de programación (*Anaconda*, *R Studio* etc) como las herramientas de control de versiones (normalmente basadas en Git, como *Sourcetree* y *Bitbucket*), las de gestión de tareas (*Jira*) etc.

Una vez montada toda la arquitectura, es importante establecer un flujo de trabajo que permita encadenar todas las etapas del procesamiento de los datos. Este flujo debe poder realizarse automáticamente (por lotes o *batches*, para los proyectos en producción) o manualmente (cuando se trate de nuevos desarrollos o ejecuciones en estado de prueba).

Con la arquitectura anterior, podría seguirse el siguiente planteamiento para actualizar las predicciones provenientes de los modelos en producción:

1. El primer día del mes, a primera hora de la mañana (por ejemplo, a las 7:00), se lanza el proceso para actualizar los datos de cara al entrenamiento de los modelos. Esto está programado en forma de pipeline en el gestor de tareas (Azure Data Factory).

2. Durante la ejecución del pipeline, se ejecutan los procesos ETL. Estos procesos son en realidad procedimientos almacenados que contienen consultas consecutivas en código SQL. Aquí se transforman, alteran y agregan los diferentes datos provenientes de las tablas del operacional. El resultado de este proceso es el dataset final de análisis, con todas las variables calculadas y configuradas para el posterior entrenamiento de los modelos. La creación del dataset es, quizá, la tarea más importante del científico de datos y donde aporta la mayor diferenciación, puesto que un modelo es incapaz de predecir bien si la información de partida no contiene ninguna información de utilidad. Y es que durante este proceso se deben calcular, modificar y generar todas las variables que pasarán a la fase de modelización.

3. Una vez finalizado el pipeline dedicado al ETL, se lanza el trigger (también a través del gestor de tareas) para la ejecución de los scripts dedicados a la analítica. En procesos ya en producción, estos scripts están diseñados de forma estandarizada y se activan mediante batches. Esto quiere decir, que se trata de scripts muy testeados, que se ejecutan automáticamente uno tras otro mediante archivos *.bash*. En ellos se realiza toda la labor de depuración de datos, así como el entrenamiento de los modelos y su posterior evaluación y predicción. Evidentemente, todas estas tareas son realizadas en la estación de trabajo, puesto que pueden contener millones de registros y centenares de campos. Este trabajo se va a centrar en esta parte (depuración de datos y entrenamiento de modelos), pero desde un punto de vista más cercano a la investigación y no a un proceso en producción.

4. El resultado de la parte analítica son unas predicciones, que dependiendo del negocio y el tipo de algoritmo utilizado pueden ser puntuaciones, probabilidades de pago, agrupamiento de individuos mediante algoritmos no supervisados etc. En cualquier caso, suele tratarse de un listado de ID's con su correspondiente predicción. El siguiente paso, por tanto, consiste en lanzar otro proceso desde el gestor de tareas para lanzar la subida de las predicciones al operacional. De esta manera, el resto de departamentos pueden utilizar dichas predicciones, ya sea reportando informes a la directiva (departamento de BI) o integrándolas en las herramientas (departamento de IT) para que tengan un efecto directo en las operaciones de la empresa.

Cabe señalar que una parte muy importante del trabajo del científico de datos también consiste en evaluar los resultados de las soluciones aportadas y tomar acciones al respecto. De nada sirve implantar un modelo de predicción si el modelo no tiene aplicación en la parte operacional y la productividad no mejora. Por tanto, otra etapa posterior al desarrollo de las soluciones consiste en la evaluación de las mismas, mediante la definición y medición de unos *KPI's* (*Key Perfomance Indicators*) que permitan conocer el desempeño de los modelos implantados en la operación de la empresa.

Esto requiere un conocimiento elevado de las tareas desarrolladas por la parte de operaciones, así como saber traducir las oportunidades de mejora en nuevos desarrollos.

Dado que el presente documento se trata de un trabajo de investigación, el desarrollo del mismo estará centrado en la etapa analítica. Es decir, el proceso de tratamiento de datos y modelización, excluyendo la labor de puesta a punto de la arquitectura, la cual sería muy costosa de reproducir en un entorno meramente académico.

Por tanto, se llevarán a cabo fundamentalmente las tareas propias de creación y selección del mejor algoritmo de machine learning. Para ello se utilizarán diferentes tipos de modelos, intentando optimizarlos hasta alcanzar el mejor modelo posible

tanto en términos de sesgo como de varianza, y posteriormente se realizará una comparación para encontrar la solución que mejor se ajuste al problema en cuestión.

Al final, el modelo de lead scoring se trata de un problema clásico de clasificación, donde el objetivo final será predecir la probabilidad de que un contacto se convierta en cliente. El desarrollo de estos algoritmos está descrito en la figura y seguirá las siguientes fases:

1. Exploración del conjunto de datos desde un punto de vista estadístico. Aquí se medirá la volumetría de cada variable, su relación con la variable objetivo, el grado de correlación entre variables etc.

2. La depuración del conjunto de datos. Esta tarea se desarrollará principalmente en *SAS Enterprise Miner*, con una pequeña parte realizada en *R Studio.*

3. División del conjunto de datos en dos muestras: entrenamiento y prueba. El primero se utilizará para entrenar los modelos y tunear los parámetros, mientras que el segundo será usado en la etapa de validación y selección del mejor modelo.

4. Construcción de los modelos y parametrización de los mismos. Aquí se intentará hallar la combinación de parámetros que optimice la calidad de las predicciones. De cada tipo de algoritmo se obtendrá un modelo finalista que pasará a la fase de comparación. Para poder seleccionar la mejor combinación de parámetros se hará uso de validación cruzada repetida, siempre bajo las mismas condiciones.

5. Evaluación final de los modelos y selección del modelo ganador. Aquí se decidirá qué modelo resulta ser el más equilibrado y se evaluará con los datos de prueba.

6. Por último, se desarrollará un modelo de *Stacking Ensemble* utilizando como base los mejores modelos de los apartados anteriores.

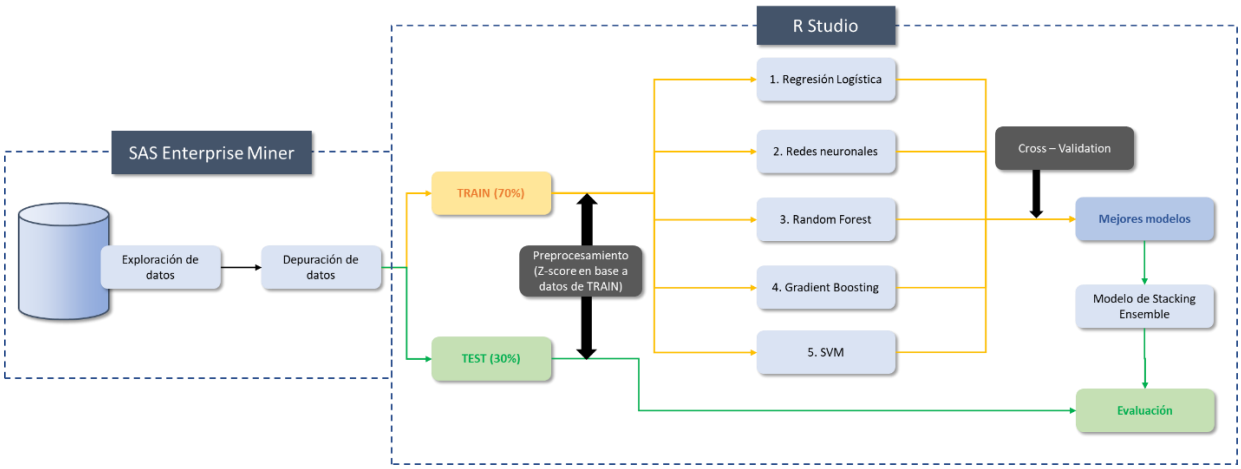


Figura 2: Esquema de la metodología propuesta.

4. Descripción del conjunto de datos.

El conjunto de datos a estudiar corresponde con una base de datos de visitantes a una web de formación online llamada *X Education*. Esta web se encarga de impartir cursos a profesionales de diferentes ámbitos dentro del sector industrial (cadena de suministro, mecánica etc).

Se trata de un conjunto de datos extraído de *Kaggle* (2018), empleado para entrenar y probar modelos de *Lead Scoring* con el objetivo de establecer estrategias de captación de clientes. Por tanto, cada individuo de la tabla se corresponde con un visitante o contacto del que se tiene constancia que ha visitado la web al menos una vez. En total la tabla dispone de 9240 observaciones.

La variable objetivo es de tipo binaria y se llama *Converted*. Esta variable toma un valor de 1 si el contacto en cuestión resulta ser cliente (es decir, se ha registrado y ha pagado por algún curso o subscripción) y toma un valor de 0 si no ha realizado dicha acción. La tasa de conversión es del 39%, es decir, 3561 contactos de los 9240 resultan ser clientes,

En cuanto al número de variables, en total hay 33. Las variables presentes en el conjunto de datos se muestran en la Tabla 1. Como se puede comprobar, hay dos variables identificativas (que serán eliminadas del conjunto puesto que carecen de utilidad en predicción), tres variables continuas, 15 variables binarias y 13 variables categóricas con más de dos categorías.

Durante el proceso de depuración de datos se eliminarán algunas de las variables puesto que no resultan interesantes por su baja variabilidad, su elevada cardinalidad o por su escasa importancia.

Variables	Rol	Tipo	Descripción
Prospect ID	ID	ID	Variable alfanumérica que identifica a cada visitante
Lead Number	ID	ID	Variable numérica que identifica a cada visitante, usada para medir la cantidad de clientes potenciales
Lead Origin	Input	Clase	Se trata de una variable categórica que sirve para identificar el origen de cada cliente potencial. Incluye API, Landing Page Submission, etc.
Lead Source	Input	Clase	La fuente desde la que el lead llegó a la web. Puede ser Google, a través de un chat etc.
Do Not Email	Input	Binaria	Una variable que indica si el cliente que visita la página no quiere recibir emails de publicidad.
Do Not Call	Input	Binaria	Una variable que indica si el cliente que visita la página no quiere recibir llamadas respecto a cursos ofertados.
Converted	Target	Binaria	La variable objetivo. Indica si el lead se ha convertido en cliente o no.
TotalVisits	Input	Continua	Número de visitas de cada visitante en la web.
Total Time Spent on Website	Input	Continua	Tiempo total que el lead ha pasado en la página
Page Views Per Visit	Input	Continua	Número medio de páginas visitadas por el lead en cada visita.

Last Activity	Input	Clase	Última actividad registrada por el consumidor. Incluye abrir el mail, tener conversación por el chat, etc.
Country	Input	Clase	País de origen del consumidor.
Specialization	Input	Clase	Rama de la industria donde se especializa el cliente potencial
How did you hear about X Education	Input	Clase	La fuente a través de la cual el cliente se enteró de la existencia de la web
What is your current occupation	Input	Clase	Indica si el cliente es estudiante, trabajador o desempleado
What matters most to you in choosing this course	Input	Clase	Principal razón para matricularse en alguno d ellos cursos
Search	Input	Binaria	Indica si el cliente ha visto la publicidad de la web en alguna de estas fuentes
Magazine	Input	Binaria	
Newspaper Article	Input	Binaria	
X Education Forums	Input	Binaria	
Newspaper	Input	Binaria	
Digital Advertisement	Input	Binaria	
Through Recommendations	Input	Binaria	Indica si el cliente llegó por recomendación de alguien
Receive More Updates About Our Courses	Input	Binaria	Indica si el cliente quiere recibir notificaciones sobre los cursos
Tags	Input	Clase	Son etiquetas usadas para valorar el estado actual del lead en cuestión
Lead Quality	Input	Clase	Indica la “calidad del lead” basada en los datos disponibles y la intuición del personal encargado de clasificar a los clientes
Update me on Supply Chain Content	Input	Binaria	Indica si el lead quiere saber de alguna actualización acerca de los cursos de Supply Chain
Get updates on DM Content	Input	Binaria	Indica si el lead quiere saber de alguna actualización acerca de los cursos de DM
Lead Profile	Input	Clase	Se trata de otra medida subjetiva de clasificación del lead
City	Input	Clase	Ciudad de origen del lead
I agree to pay the amount through cheque	Input	Binaria	Indica si el lead está dispuesto a pagar con un cheque
a free copy of Mastering The Interview	Input	Binaria	Indica si el cliente quiere una copia de la guía “Mastering the Interview”
Last Notable Activity	Input	Clase	Última actividad a destacar realizada por el cliente

Tabla 1: Definición de las variables del dataset de estudio.

5. Depuración de datos

5.1. Depuración en Miner

El proceso de depuración llevado a cabo consta de varias fases. Además, la mayoría del proceso se ha realizado a través del software SAS Enterprise Miner, mientras que el pre-procesado de los datos (estandarización de variables continuas y creación de dummies) se ha realizado en R.

El proceso de depuración llevado a cabo a través de SAS Enterprise Miner se resume brevemente en la Figura 1.

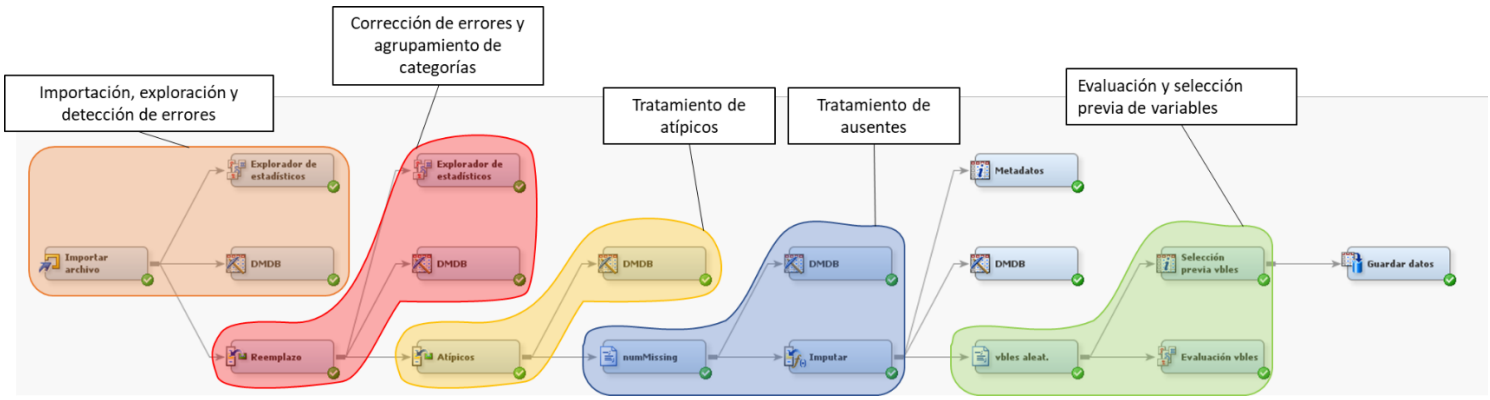


Figura 3: Diagrama de la depuración realizada en Miner.

De forma más detallada, el proceso seguido es el siguiente:

- En primer lugar, se han importado los datos al Miner. En el proceso de importación se han ajustado los roles y tipologías de las variables según se ha visto en el primer apartado del trabajo.

- En la exploración se han localizado diversas variables con un alto porcentaje de valores faltantes que deberán ser tratadas posteriormente. Además de eso, se han encontrado múltiples categorías de las variables de clase infrarrepresentadas (en este caso, se considera infrarrepresentadas aquellas categorías con menos de entre un 2% del total de observaciones). Por ejemplo, para la variable *country*, algunos países como Italia o Qatar tienen menos de 10 ocurrencias.
- Además, algunas variables poseen demasiadas categorías (más de 10). Estas variables son *country* (25 categorías), *Last_Activity* (17 categorías), *Last_Notable_Activity* (16 categorías), *Lead_Source* (20 categorías), *Specialization (*19 categorías) y *Tags* (25 categorías). Más allá de eso, no se han encontrado errores en las variables de intervalo a tener en cuenta.

- Por tanto, en el paso de corrección de errores lo que se ha intentado ha sido unificar todas las categorías infrarrepresentadas de forma que tuviera cierto

sentido. Este agrupamiento se ha llevado a cabo cuidadosamente para cada clase teniendo en cuenta dos factores: que el significado "físico" de las categorías a unificar tuviese sentido (por ejemplo, se han agrupado todos los países europeos en una sola nueva categoría llamada "Europa" para la variable *country*) y que la proporción de ceros y unos de la variable objetivo fuese similar en las categorías a unificar (de esta forma intentamos que la información sobre la varianza se pierda).

- Además, se han sustituido algunas categorías poco informativas por valores ausentes, si estaban en baja proporción. Todas las correcciones se han llevado a cabo con un nodo Reemplazo, que modifica el nombre de las variables añadiendo el prefijo *REP_*. Tras realizar el agrupamiento de categorías, se ha conseguido que todas las variables de clase tengan, como máximo, 10 categorías.

- Posteriormente, en la parte de atípicos se ha realizado un estudio de las variables de intervalo, para detectar posibles outliers que pudiesen generar problemas. Esto se ha realizado con otro nodo Reemplazo, que en este caso detecta los valores atípicos de las tres variables continuas a través del método de la desviación absoluta media. Los atípicos detectados se han convertido en ausentes (si su proporción era baja) para ser tratados posteriormente. En este caso, se han detectado únicamente 32 valores atípicos para la variable *PagesViewsPerVisit* y 41 para *TotalVisits.*

- Después, se ha utilizado un nodo de Código SAS para calcular el número de variables con valores ausentes para cada observación, generando la variable *NumMissing.* Gracias a esta variable, podemos detectar observaciones con gran cantidad de información vacía para proceder a eliminarlas (si son pocas observaciones) o imputarlas. Tras observar los datos, no se han detectado observaciones con más de 11 variables vacías, así que no ha habido que eliminar ninguna observación.

- A través de un nodo Imputar se ha realizado el tratamiento de los datos ausentes. Se han imputado con el método de la distribución aquellas variables con menos de 5% de valores ausentes. Las que tenían entre un 5 y un 50% se han recategorizado con la categoría "No_Consta" porque suponen una parte importante de los datos. Las variables con más de un 50% de ausentes directamente se han eliminado. Además, se ha configurado el nodo para generar las *M_Variables.* Se trata de un tipo de variables binarias que pueden ser de utilidad, pues determinan aquellas observaciones sobre las que se ha realizado imputación de alguna variable de clase con 1. Si no ha habido imputación recibe un valor de 0.

- Después, mediante otro nodo de código SAS se han generado dos variables aleatorias para valorar en qué grado son explicativas las variables input que restan. Estas variables se llaman *aleat1* y *aleat2*. Aquellas que obtengan resultados inferiores que las dos aleatorias en el test Chi-cuadrado y/o en el de Pearson, serán eliminadas del conjunto de datos. Como resultado de este proceso se han rechazado un total de ocho variables por su escaso valor: *REP_REP_Search, REP_REP_Through_Reccommendations, REP_REP_Country, REP_REP_City, IMP_REP_PageViewsPerVisit, REP_REP_Do_Not_Email, IMP_REP_Total_Visits.*

- Por tanto, en total han quedado 25 variables que pasarán a la fase de pre-procesado. Se ha decidido dejar de momento las variables aleatorias *aleat1* y *aleat2* puesto que todavía servir de utilidad en la parte de selección de variables. Los datos se han guardado bajo el nombre de *lead_scoring_train.sas7bdat* para su uso a partir de este punto mediante el software R Studio.

5.2. Pre-procesado en R

En este apartado se va a detallar el proceso de pre-procesamiento de los datos realizado en R Studio. Este compendio de tareas servirá para definir el conjunto de datos final, preparado para ser ingerido en la fase de modelización.

Para empezar, se ha realizado una exploración general de los datos cargados en R a través de la función summary, obteniendo los resultados mostrados en la Figura 4.

```
> summary(data_train) # Algunas variables no tienen la tipología correcta. Las variables categóricas hay que convertirlas en factores
 Prospect_ID        Lead_Number      REP_Total_Time_Spent_on_Website REP_REP_Converted REP_REP_Digital_Advertisement REP_REP_Do_Not_Call REP_REP_Get_updates_on_DM_Conten REP_REP_I_agree_to_pay_the_amoun
 Length:9240        Min.   :579533   Min.   :   0.0                  Min.   :0.0000    Length:9240                 Length:9240          Length:9240                  Length:9240
 Class :character   1st Qu.:596485   1st Qu.:  12.0                  1st Qu.:0.0000    Class :character            Class :character     Class :character             Class :character
 Mode  :character   Median :615479   Median : 248.0                  Median :0.0000    Mode  :character            Mode  :character     Mode  :character             Mode  :character
                    Mean   :617188   Mean   : 487.7                  Mean   :0.3854
                    3rd Qu.:637387   3rd Qu.: 936.0                  3rd Qu.:1.0000
                    Max.   :660737   Max.   :2272.0                  Max.   :1.0000
 REP_REP_Lead_Origin REP_REP_Newspaper  REP_REP_Newspaper_Article REP_REP_Receive_More_Updates_Abo REP_REP_Update_me_on_Supply_Chai REP_REP_X_Education_Forums   numMissing     IMP_REP_REP_Last_Activity
 Length:9240         Length:9240        Length:9240              Length:9240                     Length:9240                     Length:9240               Min.   : 0.000   Length:9240
 Class :character    Class :character   Class :character         Class :character                Class :character                Class :character          1st Qu.: 2.000   Class :character
 Mode  :character    Mode  :character   Mode  :character         Mode  :character                Mode  :character                Mode  :character          Median : 4.000   Mode  :character
                                                                                                                                                      Mean   : 4.124
                                                                                                                                                      3rd Qu.: 6.000
                                                                                                                                                      Max.   :11.000
 IMP_REP_REP_Last_Notable_Activit IMP_REP_REP_Lead_Source IMP_REP_REP_Specialization IMP_REP_REP_Tags   IMP_REP_REP_What_is_your_current IMP_REP_REP_What_matters_most_to   M_Variable
 Length:9240                      Length:9240             Length:9240                Length:9240        Length:9240                    Length:9240                      Min.   :0.000
 Class :character                 Class :character        Class :character           Class :character   Class :character               Class :character                 1st Qu.:0.000
 Mode  :character                 Mode  :character        Mode  :character           Mode  :character   Mode  :character               Mode  :character                 Median :2.000
                                                                                                                                                                      Mean   :2.081
                                                                                                                                                                      3rd Qu.:3.000
                                                                                                                                                                      Max.   :8.000
     aleat1             aleat2
 Min.   :0.0000044   Min.   :0.0002702
 1st Qu.:0.2476052   1st Qu.:0.2466428
 Median :0.4968094   Median :0.4962047
 Mean   :0.4984640   Mean   :0.4972298
 3rd Qu.:0.7500822   3rd Qu.:0.7458500
 Max.   :0.9999268   Max.   :0.9998150
```

Figura 4: resultado de la función summary sobre el conjunto de datos.

Se han detectado una serie de flecos o puntos de mejora a ser tratados:

- Es necesario realizar una pequeña limpieza, sustituyendo la tipología de las variables “character” por “factor”. De esta forma se podrán generar los dummies. Esta tarea se ha llevado a cabo con el siguiente código:

```
data_train[,as.vector(which(sapply(data_train, class)=="character"))] <-
lapply(data_train[,as.vector(which(sapply(data_train,
class)=="character"))] , as.factor)
```

- Además, los nombres de las variables son muy largos, y contienen caracteres problemáticos como el espacio en blanco. Esto puede ocasionar errores en el futuro, así que se han sustituido los nombres por otros más cortos y sencillos:

```
names(data_train)[names(data_train)=="REP_Total_Time_Spent_on_Website"]
<- "Time_on_Website"
names(data_train)[names(data_train)=="REP_REP_Converted"] <- "Y"
names(data_train)[names(data_train)=="REP_REP_Get_updates_on_DM_Conten"]
<- "DM_Content"
names(data_train)[names(data_train)=="REP_REP_Digital_Advertisement"] <-
"Dig_Advertisement"
names(data_train)[names(data_train)=="REP_REP_I_agree_to_pay_the_amoun"]
<- "pay_with_cheque"
names(data_train)[names(data_train)=="REP_REP_Last_Notable_Activity"] <-
"Last_Notable_Act"
names(data_train)[names(data_train)=="REP_REP_Newspaper_Article"] <-
"News_Article"
names(data_train)[names(data_train)=="REP_REP_Receive_More_Updates_Abo"]
<- "Receive_Updates"
names(data_train)[names(data_train)=="REP_REP_Update_me_on_Supply_Chai"]
<- "Supply_Chain"
names(data_train)[names(data_train)=="IMP_REP_REP_What_is_your_current"]
<- "Current_Occupation"
names(data_train)[names(data_train)=="IMP_REP_REP_What_matters_most_to"]
<- "Choosing_course"
names(data_train)[names(data_train)=="REP_REP_Newspaper"] <- "Newspaper"
names(data_train)[names(data_train)=="REP_REP_X_Education_Forums"] <-
"Forums"
names(data_train)[names(data_train)=="IMP_REP_REP_Last_Activity"] <-
"Last_Activity"
names(data_train)[names(data_train)=="IMP_REP_REP_Lead_Source"] <-
"Lead_Source"
names(data_train)[names(data_train)=="IMP_REP_REP_Specialization"] <-
"Specialization"
names(data_train)[names(data_train)=="IMP_REP_REP_Tags"] <- "Tags"
names(data_train)[names(data_train)=="REP_REP_Lead_Origin"] <-
"Lead_Origin"
names(data_train)[names(data_train)=="REP_REP_Do_Not_Call"] <-
"Do_Not_Call"
```

- Volviendo a hacer un summary, se puede ver que hay ciertas variables con muy poca variabilidad (variables de clase donde más del 95% de observaciones pertenecen a una sola categoría y, por tanto, son cuasi-constantes):

```
> summary(data_train)
                                Prospect_ID    Lead_Number     Time_on_Website  Y       Dig_Advertisement Do_Not_Call DM_Content pay_with_cheque                Lead_Origin   Newspaper  News_Article
 000104b9-23e4-4ddc-8caa-8629fe8ad7f4:   1   Min.   :579533   Min.   :   0.0   0:5679   No :9236          No :9238    No:9240    No:9240        API                 :3580   No :9239   No :9238
 0006d10a-eb01-4ba9-92e2-ad78588b2a40:   1   1st Qu.:596485   1st Qu.:  12.0   1:3561   Yes:   4          Yes:   2                          Landing Page Submission:4886   Yes:   1   Yes:   2
 0011be30-fa97-465b-8e44-0ae83dff7eed:   1   Median :615479   Median : 248.0                                                                Lead Import          :  55
 0011f23e-9fd9-4256-b316-efc2e2639b0d:   1   Mean   :617188   Mean   : 487.7                                                                Lead_Add_Form        : 719
 001b0ad3-9096-4af8-8205-912f5c6dafd8:   1   3rd Qu.:637387   3rd Qu.: 936.0
 001e6e14-2183-47ab-a405-108e44bc2e66:   1   Max.   :660737   Max.   :2272.0
 (Other)                             :9234
 Receive_Updates Supply_Chain Forums      numMissing                     Last_Activity  Last_Notable_Act           Lead_Source                  Specialization
 No:9240         No:9240      No :9239   Min.   : 0.000   Converted to Lead       : 428   Email   :3060    Chat_and_blogs   :1755   No_Consta             :3380
                              Yes:   1   1st Qu.: 2.000   Email                   :4071   Modified:3507    Direct Traffic   :2543   Management            :1373
                                         Median : 4.000   Email Link Clicked      : 267   SMS Sent:2172    Email            :   2   Finance               :1314
                                         Mean   : 4.124   Form Submitted on Website: 116   Website : 501    Organic Search   :1154   Media_and_Marketing   :1041
                                         3rd Qu.: 6.000   Olark Chat Conversation  : 973                    Reference        : 659   Operations            : 952
                                         Max.   :11.000   Page Visited on Website  : 640                    Socialmedia_chats: 204   Business_administration: 581
                                                          SMS Sent                :2745                    web_browsers     :2923   (Other)               : 599
                                    Tags              Current_Occupation            Choosing_course   M_Variable       aleat1            aleat2
 No_Consta                             :3509   No_Consta       :2690     Better Career Prospects:6528   Min.   :0.000   Min.   :0.0000044   Min.   :0.0002702
 Will revert after reading the email:2072   Student          : 210     No_Consta              :2712   1st Qu.:0.000   1st Qu.:0.2476052   1st Qu.:0.2466428
 wrong_number_Busy                   :1712   Unemployed       :5600                                    Median :2.000   Median :0.4968094   Median :0.4962047
 Interested_other_courses            : 630   Working_and_others: 740                                   Mean   :2.081   Mean   :0.4984640   Mean   :0.4972298
 Already a student                   : 465                                                             3rd Qu.:3.000   3rd Qu.:0.7500822   3rd Qu.:0.7458500
 Closed by Horizzon                  : 358                                                             Max.   :8.000   Max.   :0.9999268   Max.   :0.9998150
 (Other)                             : 494
```

Figura 5: summary con variables renombradas.

Se ha decidido eliminar dichas variables. También se han eliminado las variables identificativas puesto que, como se explicó previamente, no sirven de ninguna utilidad en predicción. Todas estas variables se han eliminado por medio del código:

```
data_train <- data_train[ , -which(names(data_train) %in%
c("Prospect_ID", "Lead_Number", "Dig_Advertisement", "Do_Not_Call",
"DM_Content", "pay_with_cheque", "Newspaper", "News_Article",
"Receive_Updates", "Supply_Chain", "Forums"))]
```

- Como último punto previo al pre-procesado, se debe realizar la partición de datos en entrenamiento (train) y prueba (test). La proporción elegida es de 70/30, de tal forma que la distribución de la variable objetivo sea lo más parecida posible entre los dos conjuntos. El código empleado para ello es:

```
set.seed(150990)
trainIndex <- createDataPartition(data_train$Y, p = .7,
                                  list = FALSE,
                                  times = 1)
train <- data_train[ trainIndex,]
test  <- data_train[-trainIndex,]
```

En este punto consideramos que los datos ya están preparados para poder realizar el pre-procesado propiamente dicho. Esto consta de dos técnicas fundamentales:

- **Creación de dummies**: consiste en realizar una transformación sobre las variables de clase, de tal forma que pasen de ser N variables con un total de M categorías a ser M variables binarias, cada una representando la pertenencia a una categoría de la variable de clase asociada. De esta forma se logra convertir estas variables en valores numéricos, preparados para ser utilizados en cualquier algoritmo. Este procedimiento trae como inconveniente que aumenta considerablemente la dimensión del conjunto de datos si la presencia de variables de clase es elevada, por lo que hay que valorar su implementación en términos de coste computacional.

- **Normalización**: aquí el objetivo es reescalar los valores de las variables de intervalo a límites más manejables. Esto permite que todas las variables se encuentren dentro de rangos valores equiparables, lo cual es muy útil para trabajar con distancias. También facilita el proceso de convergencia en aquellos algoritmos que utilizan métodos numéricos en el proceso de ajuste. Los tipos de normalización más utilizados son el Z-Score (estandarización) y el min-max.

$$z = \frac{x - \mu}{\sigma} \qquad X_{norm} = \frac{x - x_{\min}}{x_{max} - x_{\min}}$$

Lo habitual cuando se trabaja en un entorno real de producción es que la normalización de los datos se realice siempre partiendo de los mismos valores de media y desviación (es decir, que se utilicen los datos originales de entrenamiento como patrón para normalizar los datos que sirvan de ingesta en el futuro). De esta forma, nos aseguramos que los resultados sean siempre comparables.

Después de este paréntesis, se va a explicar cómo se ha aplicado el pre-procesado sobre los datos.

En primer lugar, se ha llevado a cabo la normalización de las variables continuas a través de la estandarización clásica (z-score). Se ha decidido realizar este paso previamente a la generación de los dummies puesto que así se evita que las variables dummies también queden afectadas por la normalización. De esta forma queda asegurado que tengan siempre valor de 0 o 1.

Es importante señalar que el estandarizado debe realizarse siempre en base a los mismos valores (media y desviación típica) para evitar que se produzcan variaciones al cambiar la semilla o al realizar un refresco de los datos. Por este motivo, se ha utilizado la función preProcess del paquete caret para realizar el estandarizado sobre los datos de entrenamiento, guardarlo y aplicarlo de nuevo sobre los datos de prueba.

Después, se han creado los dummies a través de la función *dummy.data.frame* de la librería *dummies*. Dada la alta cantidad de variables categóricas de las que dispone el conjunto de datos, el nuevo conjunto de datos incluyendo los dummies ha resultado contener nada menos que 50 variables. El código aplicado para este pre-procesado es:

```
pre <- preProcess(train)
train_norm <- predict(pre, train)
test_norm <- predict(pre, test)
train_dummy <- dummy.data.frame(train_norm, categoricas, sep = ".")
test_dummy <- dummy.data.frame(test_norm, categoricas, sep = ".")
```

El conjunto de datos resultante está compuesto por un total de 49 variables explicativas, todas ellas preparadas para poder llevar a cabo la fase de modelización. Pero antes de poder pasar a dicha fase, es muy importante realizar una selección para obtener un conjunto de datos más reducido, sencillo y aplicable. En la Tabla 2 se muestran las 49 variables obtenidas.

Variables explicativas			
Time_on_Website	*Lead_Origin.Lead_Add_Form*	*Lead_Origin.API*	*Lead_Origin.Landing Page Submission*
Lead_Origin.Lead Import	*Last_Activity.Email Link Clicked*	*numMissing*	*Last_Activity.Converted to Lead*
Last_Activity.Email	*Last_Activity.SMS Sent*	*Last_Activity.Form Submitted on Website*	*Last_Activity.Olark Chat Conversation*
Last_Activity.Page Visited on Website	*Last_Notable_Act.Website*	*Last_Notable_Act.Email*	*Last_Notable_Act.Modified*
Last_Notable_Act.SMS Sent	*Lead_Source.Reference*	*Lead_Source.Chat_and_blogs*	*Lead_Source.Direct Traffic*
Lead_Source.Organic Search	*Specialization.Finance*	*Lead_Source.Socialmedia_chats*	*Lead_Source.web_browsers*
Specialization.Business_administration	*Specialization.Online_business*	*Specialization.Management*	*Specialization.Media_and_Marketing*
Specialization.No_Consta	*Tags.Closed by Horizzon*	*Specialization.Operations*	*Specialization.Tourism_Services*
Tags.Already a student	*Tags.Not doing further education*	*Tags.Interested_other_courses*	*Tags.Lost to EINS*
Tags.No_Consta	*Current_Occupation.No_Consta*	*Tags.Not_graduated_yet*	*Tags.Will revert after reading the email*
Tags.wrong_number_Busy	*Choosing_course.Better Career Prospects*	*Current_Occupation.Student*	*Current_Occupation.Unemployed*
Current_Occupation.Working_and_others	*aleat2*	*Choosing_course.No_Consta*	*M_Variable*
aleat1			

Tabla 2: Conjunto de variables tras el proceso de depuración.

5.3. Selección de variables

Una parte esencial para optimizar el rendimiento y el desempeño de los algoritmos de machine learning es la selección de variables. Existen múltiples métodos de selección de variables, algunos de ellos ya se han visto o aplicado en este mismo trabajo (test Chi-Cuadrado, diferencia de medias, observar la variabilidad, descartar variables con alta cardinalidad etc). Otros se basan en técnicas más avanzadas.

En este apartado se van a aplicar dos métodos complementarios de selección de variables.

El primer método utilizado está basado en la construcción de varias regresiones logísticas con validación cruzada repetida, utilizando el método de selección stepwise y como criterio de selección el BIC.

La idea es la siguiente: se van a construir un total de seis regresiones logísticas, cada una con una semilla distinta, sobre las que se va a aplicar validación cruzada con una proporción 80/20. Se obtendrán por tanto seis modelos distintos, de los que se espera que cada uno seleccione una combinación de variables distinta de entre las 49 disponibles. Después se recogerá en una tabla el número de apariciones que presenta cada variable, siendo el máximo seis y el mínimo cero.

Las variables que aparezcan con mayor frecuencia serán porque tienen mayor poder predictivo, y por tanto serán seleccionadas. El resto serán descartadas.

Con este método lo que se intenta es encontrar las variables que muestran un comportamiento lineal con la variable objetivo, puesto que la regresión logística por sí sola no es capaz de hallar relaciones que no sean lineales.

Variables	Apariciones
Tags.Will_revert_after_reading_the_email	5
Tags.Closed_by_Horizzon	5
Tags.Lost_to_EINS	5
Tags.No_Consta	5
Time_on_Website	5
Lead_Source.Chat_and_blogs	5
Lead_Source.Socialmedia_chats	5
Lead_Origin.Lead_Import	4
Last_Notable_Act.SMS_Sent	4
Choosing_course.Better_Career_Prospects	4
Last_Activity.SMS_Sent	4
Last_Notable_Act.Email	3
Last_Activity.Olark_Chat_Conversation	2
M_Variable	2
numMissing	2
Lead_Origin.Landing_Page_Submission	2
Last_Notable_Act.Modified	1
Last_Activity.Converted_to_Lead	1
Choosing_course.No_Consta	1
Lead_Origin.Lead_Add_Form	1

Tabla 3: Tabla resumen de apariciones de las variables más influyentes por método lineal.

Los resultados se muestran en la Tabla 3. Como se puede comprobar, hay un total de 18 variables que aparecen en al menos un modelo. Por otro lado, las otras 31 variables no aparecen ni una sola vez, por lo que se han eliminado de la tabla y son candidatas a ser descartadas.

No hay que olvidar que este método de selección por si solo puede no ser lo suficientemente efectivo. Hay relaciones entre las variables explicativas y la variable objetivo que las regresiones no son capaces de hallar (o al menos si no se incorporan interacciones o transformaciones de variables). Por ello, es importante contrastar estos resultados con algún método capaz de encontrar patrones y reglas que queden fuera del ámbito explorado por la regresión.

Los algoritmos basados en árboles pueden ser buena opción para estos casos.
En concreto, se ha utilizado el algoritmo de *Random Forest* (que se explicará más detenidamente más adelante en este mismo trabajo) como segundo método de selección de variables con validación cruzada.

A grandes rasgos, este algoritmo promedia los resultados de la predicción de numerosos árboles de decisión construidos en base a los datos de entrenamiento, en los que en cada iteración se sortean tanto las observaciones incluidas en el árbol como las propias variables elegidas en cada subconjunto de datos.

De esta forma, este método permite obtener las variables más importantes, basándonos en sus apariciones en las diferentes reglas de división a la hora de conformar cada árbol. Lo que se hace es realizar un test de diferencia de medias entre la precisión de las predicciones obtenidas al utilizar las variables muestreadas y las que se obtendrían con las variables "*Out of the Bag*" (OOB, que son las que no han caído en las muestras).

Este método por si solo es bastante efectivo a la hora de seleccionar variables, pero mejora todavía más si se añade validación cruzada repetida. En este caso se ha ensayado en R con un random forest sencillo, sorteando un total de 7 variables en cada iteración y todas las observaciones, al que se le ha aplicado validación cruzada repetida con 10 grupos y 3 repeticiones mediante el siguiente código:

```
control <- trainControl(method = 'repeatedcv',
                        number = 10,
                        repeats = 3)
mtry <- round(sqrt(ncol(data_train_std)) + 1)
tunegrid <- expand.grid(.mtry = mtry)
rf_default <- train(Y~.,
                    data = data_train_std,
                    method = 'rf',
                    metric = 'Accuracy',
                    tuneGrid = tunegrid,
                    trControl = control)
fm <- rf_default$finalModel
varImp(fm)
```

Los resultados de ambos métodos de selección aplicados se han recogido en la Tabla 4.

Cabe aclarar que la tercera columna de la tabla muestra los mismos resultados de la segunda, pero transformados en forma de porcentaje sobre el total.

Variable	Importancia	Importancia %
Tags.Will revert after reading the email	659.849607	23%
Time_on_Website	356.54043	13%
Tags.wrong_number_Busy	192.402115	7%
Tags.Closed by Horizzon	135.613634	5%
Last_Notable_Act.SMS Sent	121.798075	4%
Lead_Origin.Lead_Add_Form	117.431853	4%
aleat2	105.115337	4%
aleat1	97.976137	3%
Last_Activity.SMS Sent	92.104153	3%
numMissing	86.518979	3%
Tags.Lost to EINS	83.091678	3%
Choosing_course.No_Consta	71.798864	3%
Tags.Interested_other_courses	62.384596	2%
M_Variable	59.40671	2%
Choosing_course.Better Career Prospects	58.394642	2%
Tags.No_Consta	56.712897	2%
Current_Occupation.No_Consta	50.955954	2%
Current_Occupation.Working_and_others	48.251991	2%
Tags.Already a student	44.122903	2%
Last_Notable_Act.Modified	39.085096	1%
Lead_Source.Reference	24.164766	1%
Current_Occupation.Unemployed	19.081256	1%
Last_Activity.Email	16.186973	1%
Lead_Origin.API	15.959379	1%
Lead_Origin.Landing Page Submission	15.750205	1%
Last_Notable_Act.Email	15.321682	1%
Tags.Not_graduated_yet	14.231405	1%
Lead_Source.web_browsers	13.489988	0%
Lead_Source.Direct Traffic	13.328072	0%
Specialization.Management	12.026349	0%
Specialization.No_Consta	11.778616	0%
Lead_Source.Organic Search	11.265784	0%
Specialization.Operations	11.177927	0%
Last_Activity.Olark Chat Conversation	11.095869	0%
Specialization.Media_and_Marketing	11.005608	0%
Lead_Source.Socialmedia_chats	10.989268	0%
Lead_Source.Chat_and_blogs	10.74813	0%
Tags.Not doing further education	10.186578	0%
Specialization.Finance	9.243524	0%
Specialization.Business_administration	8.351436	0%
Last_Activity.Page Visited on Website	6.218641	0%
Last_Notable_Act.Website	6.076856	0%
Specialization.Tourism_Services	5.68319	0%
Last_Activity.Converted to Lead	5.614632	0%
Last_Activity.Email Link Clicked	4.653442	0%

Specialization.Online_business	2.896591	0%
Last_Activity.Form Submitted on Website	1.990761	0%
Current_Occupation.Student	1.942963	0%
Lead_Origin.Lead Import	1.465684	0%

Tabla 4: Tabla resumen de las variables más influyentes por Random Forest.

A la vista de los resultados parece que no hay unanimidad total entre los dos métodos a la hora de seleccionar las variables. Si bien es cierto que la mayoría de variables que en regresión parecían importantes también han resultado serlo en random forest, hay algunas variables que en random forest parecen haber tenido bastante peso y, sin embargo, en regresión no aparecieron ni una sola vez (por ejemplo, *Tags.wrong_number_Busy*).

En base a estos resultados, se van a seleccionar las mejores variables en concordancia con ambos métodos. En total, el set de variables escogido contiene 13 variables, las cuales corresponden con las que aparecieron al menos 4 veces en el método de selección lineal y las que tuvieron más de un 3% de importancia relativa en el método de árboles.

En la Tabla 5 se muestra el conjunto de variables seleccionado.

Variables seleccionadas
Tags.Will_revert_after_reading_the_email
Tags.Closed_by_Horizzon
Tags.Lost_to_EINS
Tags.No_Consta
Time_on_Website
Lead_Source.Chat_and_blogs
Lead_Source.Socialmedia_chats
Tags.wrong_number_Busy
Last_Notable_Act.SMS_Sent
Lead_Origin.Lead_Add_Form
Lead_Origin.Lead_Import
Choosing_course.Better_Career_Prospects
Last_Activity.SMS_Sent

Tabla 5: Variables escogidas para el dataset final

Por último, decir que los nombres de las variables se han simplificado para que sea más sencillo trabajar con ellos. Los nombres finales se muestran en la Tabla 6.

Variables seleccionadas
Tags_Revert
Tags_Horizzon
Tags_Lost
Tags_NC
Time_website
Source_Chat
Source_socialmedia

Tags_wrong_N
Last_Act_SMS
Origin_Add_Form
Origin_Import
Better_career
Last_N_Act_SMS

Tabla 6: Variables finales renombradas

6. Modelización

Una vez los datos han sido depurados, se va a comenzar con la fase de modelización.

En esta fase se tuneará cada uno de los cinco principales algoritmos de predicción (regresión logística, redes neuronales, random forest, gradient boosting y SVM) sobre los datos de entrenamiento.

Se utilizará como medida para evaluar la bondad de ajuste el valor del área bajo la curva ROC (en adelante *AUC*), y como técnica de validación se usará validación cruzada repetida. De esta forma, se irá evaluando la mejor combinación de hiperparámetros para cada algoritmo, representando en forma de diagrama de cajas los resultados del AUC para tener una vista de qué combinación es la mejor tanto a nivel de sesgo como de varianza.

Además, se intentará que las soluciones elegidas no correspondan a valores aislados, sino a intervalos o entornos dentro de los cuales el algoritmo muestre una estabilidad en sus resultados. Con esto lo que se quiere decir es que no se busca un valor que maximice el AUC si ese valor muestra un comportamiento extraño o se encuentra de un rango muy variable, donde aumentar o disminuir levemente su valor haga que los resultados empeoren drásticamente.

Hay que tener en cuenta que estos modelos siempre subestiman el error en comparación con un caso real. Por tanto, no conviene elegir soluciones que sean inestables u obtengan una varianza elevada.

Tras construir la mejor opción dentro de cada algoritmo, se compararán entre ellos, seleccionando el modelo o modelos. Después se atacará el problema de evaluación sobre el conjunto de datos de prueba, calculando la tasa de acierto, así como el resto de medidas referentes a la matriz de confusión.

6.1. Árbol de decisión

El primer modelo que se va a usar es el árbol de decisión / clasificación. Este modelo, a diferencia del resto que se tratarán en los siguientes apartados, solamente se va a tratar con propósito introductorio y como base para entender algunos de los modelos más avanzados que se verán a continuación.

El árbol de decisión es uno de los modelos clásicos utilizados en problemas de clasificación. A pesar de tratarse de un modelo sencillo y que requiere de poco cálculo computacional, sorprende por su versatilidad, pudiendo ser usado independientemente de la tipología de las variables, la presencia de valores ausentes y atípicos, la posibilidad de usar variables de clase sin tratar etc. También obtiene resultados relativamente buenos dada su simpleza.

La idea detrás de los árboles de decisión consiste en encontrar el punto óptimo para dividir el conjunto de datos en sucesivas ramas. De esta forma, con cada división, se genera una partición de los datos donde se maximiza la diferencia entre los grupos generados en términos de la variable objetivo.

Por tanto, cada árbol viene formado por un conjunto de hojas o nodos, que van surgiendo de la aplicación de las sucesivas reglas de división. Se dice que la pureza de los nodos es mayor cuanto mayor es la diferencia entre las poblaciones de ambas clases de la variable objetivo. El árbol se sigue ramificando hasta que se alcanzan nodos del tamaño mínimo fijado o hasta que no se encuentran diferencias entre las poblaciones de los nodos resultantes.

Un árbol viene definido por una serie de parámetros como pueden ser el número de nodos, el número mínimo de observaciones en cada nodo o la profundidad máxima.

Muchas veces el árbol de decisión no es usado por su poder predictivo, sino por su capacidad para establecer una jerarquía en la importancia de las variables (como ya se pudo ver previamente en el proceso de selección de variables por random forest) o por su gran interpretabilidad. Y es que el otro punto fuerte de los árboles es que resultan muy útiles para encontrar una interpretación de lo que sucede en nuestros datos.

Con dicho objetivo, se va a construir un árbol de decisión en R. Esta tarea es muy sencilla si se usa la librería *rpart* y *rpart.plot:*

```
tree <- rpart(train$Y ~ ., data = train[,1:14], method = 'class')
rpart.plot(tree, extra = 106)
```

Los resultados quedan graficados según se muestra en la Figura 6. La interpretación es la siguiente:

- El nodo inicial es el punto de partida, y contiene una tasa de conversión del 39%. En este nodo se encuentran el 100% de las observaciones.

- La variable más influyente es la que establece la primera división, que parece ser *Tags_Revert*. Se recuerda que esta variable es un dummy que proviene de la variable original Tags, la cual corresponde con etiquetas descriptivas con respecto al contacto. En concreto, esta etiqueta se refiere a si el contacto ha contestado tras recibir un mail. Parece que, si esta variable es 1, tasa de conversión aumenta al 97%.

- La siguiente variable más influyente es *Tags_Horizzon*, que también es un dummy de la variable Tags. En este caso, si vale 1, la tasa es del 100%. Después iría *Tags_Lost,* con un comportamiento similar.

- Después va *Last_N_Act_SMS,* que es un dummy y se refiere a si la última actividad notable realizada por el contacto fue usar SMS. Si vale 0, la tasa de conversión es del 9%.

- La última regla de división hace referencia de nuevo a la variable Tags. Esta vez se refiere a si se ha contactado con un número erróneo.

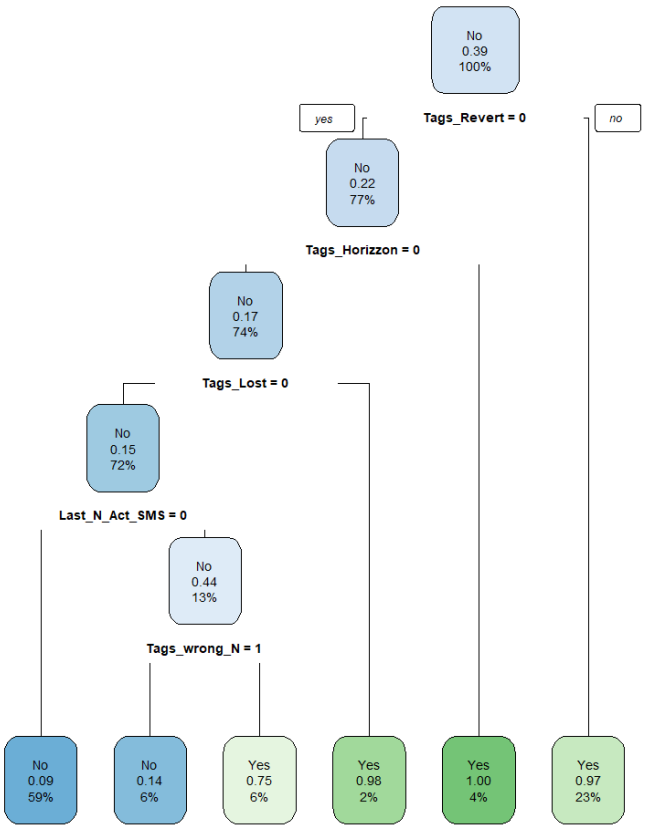


Figura 6: Resultado del árbol de decisión

Una vez construido el árbol ya se entiende más fácilmente qué reglas son las que establecen el patrón de comportamiento d ellos datos. Se destaca que la totalidad de ellas son variables dummies.

6.2. Regresión Logística

El modelo de regresión logística es uno de los modelos clásicos utilizados en los problemas de clasificación. Su uso en R está comprendido en la librería caret (*RPubs - Introducción al paquete Caret*, s. f.), dentro de los modelos lineales o *glm* (*glm function | R Documentation*, s. f.).

Se trata de un algoritmo que funciona de forma análoga a la regresión lineal, pero en este caso la variable objetivo es de tipología categórica o binaria (*Lead Scoring (Logistic Regression)*, s. f.).

La regresión logística, como cualquier algoritmo de clasificación, trata de calcular la probabilidad de que se produzca un evento (normalmente el evento se asocia al valor de 1 en la variable objetivo), que en este caso se trata de la conversión del contacto o lead en cliente final de la empresa. Dicha probabilidad se obtiene a través de la estimación de una serie de parámetros (β) que actúan como "pesos" de cada una de las variables input.

$$p1 = P(Y = 1\,|x_1, x_2\,, \ldots, x_m\,) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x_1 + \cdots + \beta_m x_m)}}$$

La estimación de dichos pesos suele realizarse a través de un método numérico basado en maximizar el valor de la función de máxima verosimilitud, que persigue que las probabilidades de evento ($y = 1$) sean próximas a 1.0 si el evento es verdadero y cercanas a 0.0 si no se produce el evento ($y = 0$).

Al final, la regresión logística es un método sencillo, que requiere bajo coste computacional y que es capaz de reproducir muy bien los problemas de clasificación en los que hay una separación entre clases de tipo lineal, como se muestra en la Figura 7.

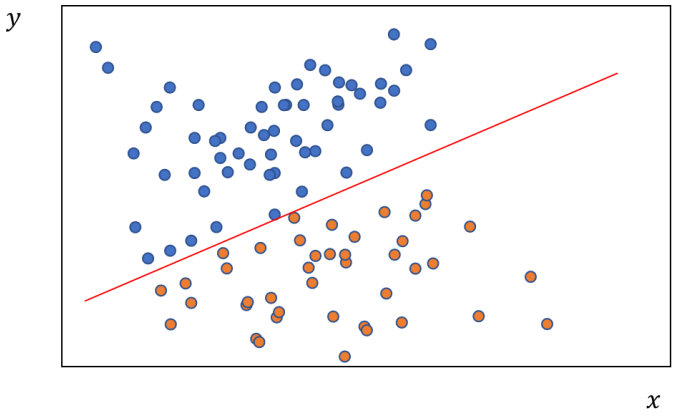


Figura 7: Representación de un problema clásico de clasificación con separación lineal.

La regresión logística no funciona tan bien cuando la separación muestra no linealidad, aunque es posible mejorar los resultados incorporando interacciones entre las diferentes variables o transformaciones (por ejemplo, es posible que si se utiliza el

cuadrado de una variable input el modelo sea capaz de clasificar mejor que si se usa la variable original).

Además, como ya hemos mencionado en la parte de selección de variables, los modelos de regresión logística permiten realizar una selección de las mejores variables a través de los métodos *stepwise*, *forward* y *backwards*.

Estos métodos evalúan de forma iterativa los resultados de la predicción generada por la regresión, incorporando (o quitando) un nuevo parámetro (es decir, una nueva variable input multiplicada por su respectivo peso) en cada paso. Se utiliza un coeficiente de comparación, normalmente el *BIC* o el *AIC*, y si al incluir o quitar dicho parámetro mejoran los resultados dentro de un margen razonable, se continúa con la operación. En el momento en el que los resultados apenas mejoran, el proceso se detiene y el modelo creado se queda con los parámetros definidos hasta dicho momento.

Otra característica muy interesante de estos modelos es que son muy fácilmente interpretables, puesto que los valores y el signo de cada parámetro β nos dan una pista muy clara de en cuanta proporción afecta la variación de cada variable input sobre la variable objetivo (este análisis es orientativo, aunque algo simplista puesto que en los casos reales el aumentar una variable puede afectar a las demás y rara vez varían de forma aislada).

En los modelos de scoring (muy usados en sector bancario, aseguradoras, empresas de recobro de deuda, finanzas…) las regresiones logísticas son muy populares, puesto que la estimación de estos parámetros β permite la construcción de tarjetas de puntuación. Estas tarjetas permiten convertir el valor de una determinada variable input (por ejemplo, el rango de edad del cliente) en una puntuación (por ejemplo, en un seguro de vida, si el cliente tiene entre 20 y 30 años recibirá más puntos que si su edad se encuentra entre los 40 y los 50). La suma de todas las puntuaciones obtenidas de las variables input para cada cliente dará lugar a una puntuación final. Si este valor supera cierto umbral, el cliente podrá acceder a cierto producto o será clasificado dentro de una determinada categoría.

Tras este pequeño paréntesis teórico, se va a pasar a la aplicación de la regresión logística de forma práctica. Para ello, se construirán diversos modelos de regresión, intentando encontrar cuál se comporta mejor.

- El primero de ellos consiste en una regresión básica, incluyendo las trece variables explicativas, sin transformaciones, sin interacciones y sin aplicar ningún método de selección como forward o stepwise. El modelo se ha construido con la libería caret de la siguiente manera:

  ```
  modeloInicial <- glm(Y ~., train, family=binomial)
  ```

  Mediante la función summary hemos obtenido el valor de cada coeficiente asociado a cada una de las variables input:

```
Coefficients:
                   Estimate Std. Error z value Pr(>|z|)
(Intercept)        -6.55342    0.28951 -22.636  < 2e-16 ***
Tags_Revert         6.79509    0.29479  23.051  < 2e-16 ***
Tags_Horizzon       9.36590    1.04107   8.996  < 2e-16 ***
Tags_Lost           8.94334    0.77579  11.528  < 2e-16 ***
Tags_NC             3.61142    0.26818  13.466  < 2e-16 ***
Time_website        1.05431    0.05902  17.864  < 2e-16 ***
Source_Chat         1.12424    0.14563   7.720 1.16e-14 ***
Source_socialmedia  2.62043    0.52019   5.037 4.72e-07 ***
Tags_wrong_N        0.35193    0.28453   1.237 0.216136
Last_Act_SMS        0.69947    0.20453   3.420 0.000626 ***
Origin_Add_Form     1.17809    0.30192   3.902 9.54e-05 ***
Origin_Import      -2.42260    0.84164  -2.878 0.003997 **
Better_career       2.05898    0.13507  15.244  < 2e-16 ***
Last_N_Act_SMS      1.72841    0.21779   7.936 2.09e-15 ***
```

Se puede ver que la variable más influyente es *Tags_Horizzon*, pero en realidad todas tienen un nivel de significación elevado (lo cual era previsible dado el sistema de selección de variables empleado).

Este modelo presenta un valor de AUC en los datos de entrenamiento de 0.9701 y de 0.9713 en el conjunto de prueba.

Con este resultado queda patente que el conjunto de datos está muy preparado para la modelización. Es muy poco habitual obtener valores tan elevados en modelos reales de scoring, salvo que sean casos donde se tenga mucha información sobre los clientes (y en cuyo caso estaría por ver si emplear modelos de machine learning aporta algún valor añadido).

Este resultado también deja entrever que el modelo no genera ningún tipo de sobreajuste. Esto es importante porque puede significar que un número de variables más elevado al del conjunto seleccionado puede sacar todavía puntuaciones más elevadas sin llegar a sobreajustar.

- El segundo modelo de regresión logística incluye una selección manual de las siete variables más influyentes (según el método de selección lineal). Se trata de un modelo más sencillo, que contiene solamente siete parámetros:

```
modeloManual <- glm(Y ~., train[, c(1,2,3,4,5,6,14)], family = binomial)
```

Muchas veces los modelos más sencillos obtienen mejores resultados que modelos más complejos, puesto que son capaces de representar de mejor manera las generalidades del conjunto de datos, sin caer en el exceso de ceñirse a las particularidades de algunos individuos de entrenamiento (definición de sobreajuste).

Sin embargo, como se ha mencionado, el modelo anterior no llegaba a provocar ese sobreajuste, así que es de esperar que este modelo obtenga peor resultado.

El valor del AUC en entrenamiento es de 0.9255. mientras que en prueba es de 0.929. Queda claro, por tanto, que este modelo apunta a ser peor en términos de poder predictivo que el anterior.

- Los siguientes cuatro modelos se han construido con selección de variables mediante métodos forward y stepwise, utilizando las variables input originales sin transformar. Además, se ha utilizado como criterio de selección tanto el AIC como el BIC.

  Estos modelos se han construido de la siguiente forma:

```
null <- glm(Y ~ 1, data = train, family = binomial)
full <- glm(Y ~ ., data = train, family = binomial)
modeloForAIC <- step(full, scope = list(lower = null, upper = full),
direction = "forward")
modeloStepAIC <- step(null, scope = list(lower = null, upper = full),
direction = "both")
modeloForBIC <- step(full, scope = list(lower = null, upper = full),
direction = "forward", k = log(nrow(train)))
modeloStepBIC <- step(null, scope = list(lower = null, upper = full),
direction = "both", k = log(nrow(train)))
```

  Al construirse de forma iterativa en base a una métrica de ajuste, es de esperar que los modelos obtenidos mediante estos métodos resulten ser más eficaces que los anteriores.

  El número de parámetros obtenido mediante estos métodos resulta ser de 13 para los dos modelos construidos con stepwise y 14 para los de forward. Es decir, estos últimos dan como resultado el mismo modelo de regresión logística obtenido inicialmente en este apartado, incluyendo todas las variables originales.

  A nivel de AUC en prueba, los modelos formados por stepwise dan 0.9713, mientras que los de forward dan 0.9712. Queda patente que la diferencia entre ambos modelos es insignificante, aunque los modelos stepwise son ligeramente más sencillos al contener un parámetro menos.

- Por último, se han construido dos modelos con selección de variables mediante stepwise (con criterio AIC y BIC) incluyendo interacciones entre variables.

  Cuando se habla de interacciones en el ámbito de la regresión, nos referimos a la posibilidad de que exista algún tipo de comportamiento oculto que surja al combinar varias variables input y que por separado sea imposible percibirlo. Es decir, consiste en construir nuevas variables que sean producto de dos o más variables originales.

  En la práctica estas interacciones suelen limitarse a dos variables, descartando las de un grado más elevado. Además, esto supone calcular todas las combinaciones de productos entre pares de variables. Al tener un total de 13 variables explicativas, calcular las interacciones supone aumentar el número de

variables de 13 hasta 91 (resultado de $13 + 12 + 11 + 10 + \cdots + 1$). El coste computacional en comparación con el dataset original es evidente.

Aun así, se suelen usar para encontrar relaciones ocultas o definir nuevas variables de utilidad.

En este caso, los modelos construidos mediante este método siguen este código:

```
fullInt <- glm(Y ~ .^2, data = train, family = binomial)
modeloStepIntAIC <- step(null, scope = list(lower = null, upper =
fullInt), direction = "both")
modeloStepIntBIC <- step(null, scope = list(lower = null, upper =
fullInt), direction = "both", k = log(nrow(train)))
```

Donde se ha obtenido el valor de AUC en entrenamiento (0.975 para ambos modelos) y en prueba (0.974 para el BIC y 0.975 para el AIC).

Echando un vistazo al resultado con la función summary, se observa que el modelo construido con el AIC contiene un total de 32 parámetros, mientras que el del BIC contiene 19. Como ya se ha dicho, uno de los aspectos por los que la regresión es muy utilizada es porque son modelos sencillos y rápidos, con elevada interpretabilidad. Por tanto, utilizar un número excesivo de parámetros contradice la razón de ser de este tipo de modelos. Por este motivo, consideramos que el modelo obtenido con BIC es superior, al ofrecer un AUC ligeramente inferior, pero poseyendo muchos menos parámetros.

Una vez construidos los diferentes modelos de regresión, es evidente que calcular el valor del AUC en entrenamiento y prueba una sola vez no basta para comprobar la bondad de estos modelos. Ya se ha mencionado que para tener una visión más completa del error de predicción hay que medirlo en términos de su varianza.

Por tanto, se ha utilizado la función *trainControl* de la librería caret para definir un proceso de validación cruzada repetida que servirá para comparar los resultados de los modelos construidos. Este proceso de validación cruzada contiene 5 repeticiones y 5 grupos y será común para el resto de algoritmos a estudiar de ahora en adelante.

Para realizar la comparación se van a representar los resultados en forma de diagrama de cajas mediante la función *boxplot* (Figura 8). En el eje vertical se representa el valor del AUC obtenido en el proceso de validación cruzada repetida, mientras que en el eje horizontal se muestran, en la parte superior el número de parámetros que contiene el modelo; y en la inferior, los cuatro modelos de regresión logística seleccionados, siendo estos:

- **Modelo 1:** modelo inicial con variables originales.
- **Modelo 2:** modelo con selección manual de las siete mejores variables.
- **Modelo 3:** modelo con método stepwise y criterio BIC.
- **Modelo 4:** modelo con método stepwise, BIC e interacciones.

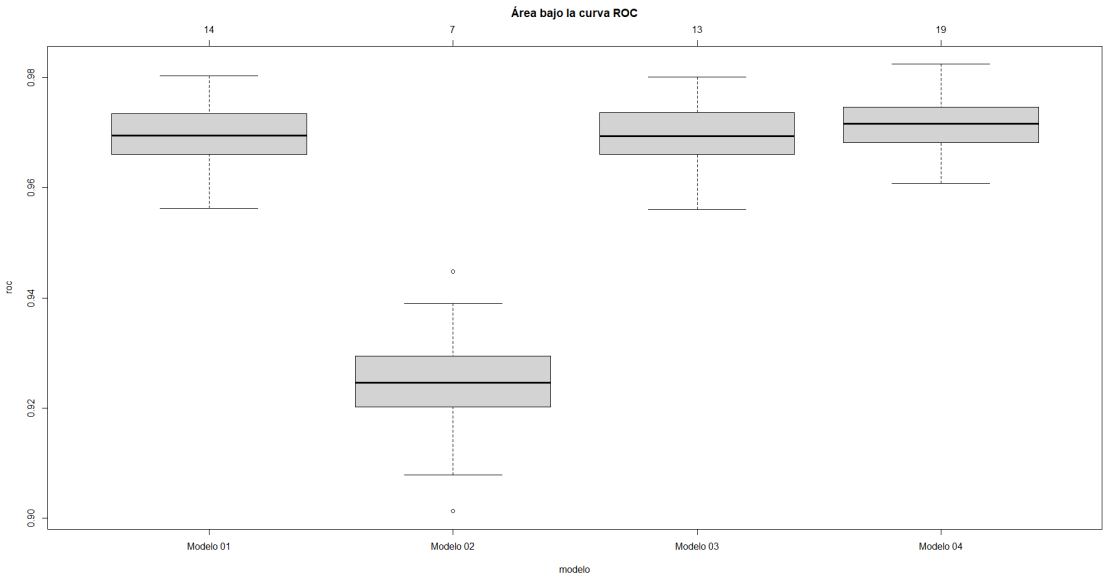


Figura 8: Diagrama de cajas de comparación entre modelos de regresión.

Observando el diagrama parece que los modelos 1, 3 y 4 obtienen resultados muy similares tanto en sesgo como en varianza, mientras que el modelo 2 se encuentra bastante por debajo en cuanto a sesgo.

De estos cuatro modelos solo uno pasará a la fase final de evaluación. En términos de AUC ya se ha visto que cualquier opción excepto el segundo modelo sería un buen candidato. Por tanto, se va a usar como criterio de desempate el número de parámetros incluidos en el modelo. Dicho esto, **la mejor opción resulta ser el modelo 3, que consistía en una regresión logística construida con selección de variables mediante el método stepwise, usando como criterio de selección el BIC.**

6.3. Redes Neuronales

Las redes neuronales artificiales son un tipo de algoritmo de Machine Learning que nació a finales del siglo pasado y han ido evolucionando hasta nuestros días. Se trata de una reproducción del comportamiento del sistema nervioso humano, donde la información accede a cada neurona como un *input*, y ésta a su vez la transforma y la transmite hacia las siguientes neuronas como *outputs.* Cuando la información ha viajado a través de toda la cadena de neuronas, finalmente genera una salida, que es la acción que queríamos realizar.

En Machine Learning la entrada inicial son sencillamente las variables input del conjunto de datos, y el output final es la predicción de la variable objetivo. Las neuronas se representan como nodos situados en capas, donde cada neurona transforma los datos de entrada a través de una función de activación. Existen diversas funciones de activación (tangente hiperbólica, sigmoidea, arco-tangente etc).

El resultado de cada capa pasa a la siguiente capa mediante una función de combinación, que agrega o combina los resultados de la capa anterior.

Existen diferentes tipos de redes neuronales. Las clásicas constan únicamente de una capa intermedia (también llamada capa oculta). En *deep learning* se utilizan sucesivas capas ocultas, aumentando la profundidad de la red. También existen otro tipo de redes como las convolucionales, que se usan especialmente en reconocimiento de imágenes y visión artificial. Con todo, en este trabajo se estudiará únicamente la red clásica con una sola capa oculta.

Por tanto, una red neuronal no es ni más ni menos que una gran función matemática que transforma los datos de las variables de entrada en una predicción final de la variable objetivo.

Para calcular este valor la red estima una serie de parámetros conocidos como pesos y sesgos. Cada nodo de la red (H_i) tiene un peso (w_{ij}) y sesgo (b_i) asociados. El método de estimación de estos parámetros es un método numérico de carácter iterativo, en el que en cada iteración se actualiza el valor de cada parámetro sumando o restando cierta cantidad. Dicha cantidad viene marcada por lo que se conoce como *método de descenso del gradiente*.

Este método evalúa una función de coste (normalmente tomada como una medida del error de predicción en cada iteración) y calcula el gradiente de esta función con respecto a cada peso/sesgo comparando con la iteración previa. El signo en el que cada parámetro evoluciona viene marcado por la dirección en la que dicho gradiente sea negativo. En otras palabras, si aumentar un peso hace que la función de coste disminuya, se seguirá aumentando ese peso. Si hace que aumente, entonces se hará disminuir dicho peso.

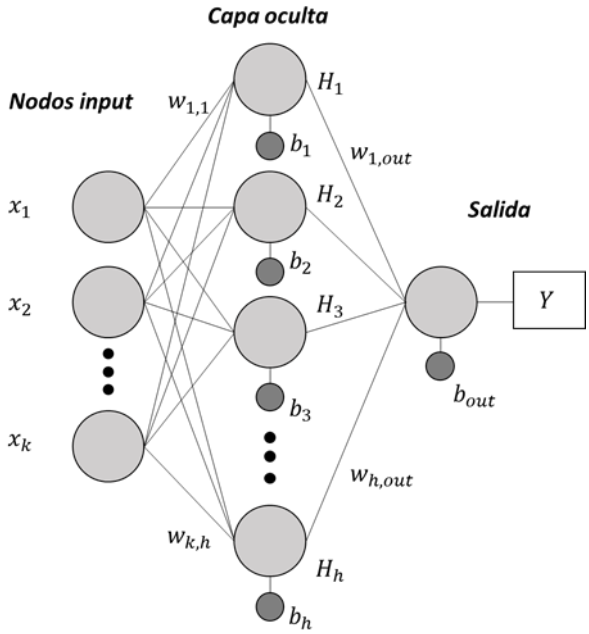


Figura 9: Esquema de la estructura de una red neuronal clásica.

La cantidad en la que cada parámetro aumenta o disminuye es proporcional a la variación de la función de coste con respecto a dicho parámetro y a una constante llamada tasa de aprendizaje (ε, también llamada *learning rate*).

$$w_{ij_{(k)}} = w_{ij_{(k-1)}} - \epsilon \frac{\partial E_{(k-1)}}{\partial w_{ij}}$$

Este es el algoritmo de convergencia más común, pero existen otros que introducen coeficientes de regularización o incluyen términos con la segunda derivada de la función de coste. El algoritmo de convergencia es un hiperparámetro a tener en cuenta a la hora de construir la red.

Pero el hiperparámetro más importante a la hora de crear la arquitectura de la red se trata del número de nodos de la capa oculta. Si se incluye un número de nodos demasiado elevado, el número de parámetros a calcular será también demasiado alto, generando una función muy compleja, que normalmente tenderá a presentar sobreajuste y dará una varianza muy elevada en los datos de validación. Por otro lado, si el número de nodos es muy bajo, la red no conseguirá un ajuste suficiente, dando lugar a un sesgo elevado en validación.

Por tanto, es esencial escoger muy bien el valor del número de nodos, evitando en toda la medida posible la aparición de sobreajuste. En la literatura, se usa como valor orientativo la cifra de 20 o 30 observaciones por parámetro. Es decir, si se consigue que el modelo contenga una cifra superior a 20 observaciones por cada parámetro a estimar, es muy probable que no se presenten problemas de sobreajuste.

Siguiendo esta regla, para saber en qué rango se debe encontrar el número de nodos de la red, se realizará un pequeño cálculo tomando como base un valor de 30

observaciones por parámetro. De esta forma se tratará de minimizar la presencia de sobreajuste.

Siendo h el número de nodos en la capa oculta y k el número de variables input, tenemos:

$$N_{parametros} = h(k+1) + h + 1$$

Al ser un problema de clasificación, la condición de 30 observaciones por parámetro se calcula tomando como referencia la clase minoritaria de la variable objetivo en los datos de entrenamiento. En este caso, el conjunto de entrenamiento contiene un total de 6469 observaciones. De estas 6469, el 39% pertenecen a la clase minoritaria, lo que hace unas 2500 observaciones. Como el conjunto de datos contiene 13 variables input, sustituyendo se obtiene:

$$h(13+1) + h + 1 = \frac{2500}{30}$$
$$14h = \frac{2500}{30} - 1$$
$$h = 5.58 \ldots \approx 5$$

Es decir, con cinco nodos en la capa oculta se puede asegurar que el número de observaciones por parámetro será superior a 30. Esta cifra se usará de forma meramente orientativa, puesto que no deja de ser una "receta" en la que basarse a la hora de construir una red neuronal, pero a la hora de la verdad la aparición de sobreajuste puede estar marcada por otros muchos factores como la distribución del conjunto de datos, el algoritmo de convergencia, la función de activación etc.

A continuación, se abordará la construcción de la mejor red neuronal para el conjunto de datos de estudio. Se empezará afrontando el problema del *Early Stopping*.

Este mecanismo consiste en estudiar la influencia de cada hiperparámetro de la red en el proceso de convergencia de la función de coste. Lo que se intenta es observar el comportamiento del ajuste de la red al aumentar el número de iteraciones, intentando encontrar un punto adecuado en el que detener el proceso de convergencia (punto de early stopping). El objetivo es determinar si se produce algún tipo de sobreajuste a partir de cierto número de iteraciones. En ese caso, estudiar si ese sobreajuste desaparece al limitar el valor del hiperparámetro o si es más efectivo detener el proceso en el punto de early stopping.

En la Figura 10 se presentan dos casos típicos de procesos de convergencia en redes neuronales. En el primero se muestra un proceso normal sin problemas de sobreajuste. En el segundo, se muestra un proceso con sobreajuste excesivo a partir del punto de early stopping. En problemas de clasificación suele abordarse este tema desde el punto de vista del valor del AUC en lugar del error de predicción. Por tanto, las curvas suelen estar invertidas, como se muestra en la Figura 11.

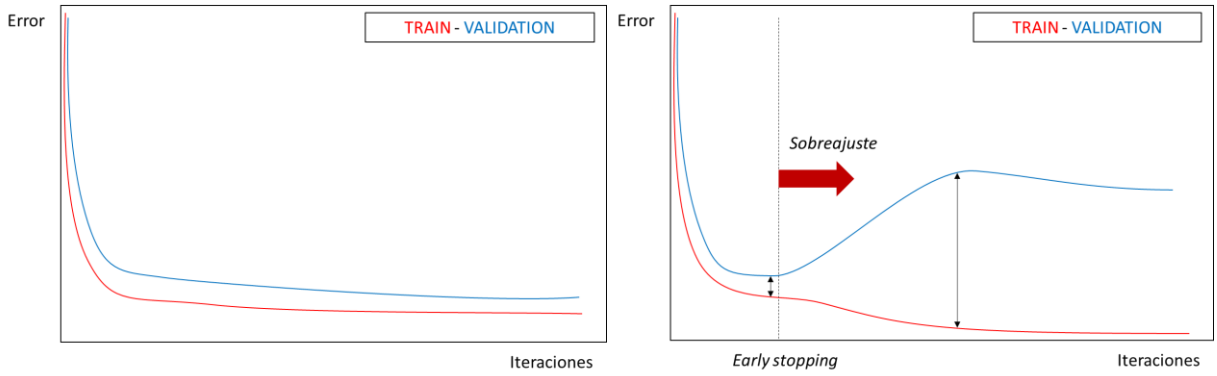


Figura 10: Representación del concepto general de early stopping (izquierda: convergencia sin sobreajuste, derecha: convergencia con sobreajuste tras el punto de early stopping).

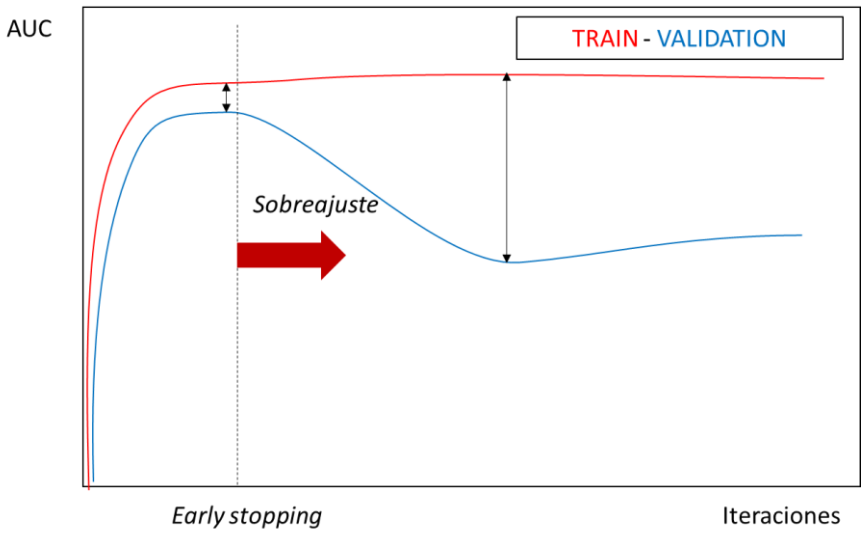


Figura 11: Representación del proceso de convergencia con early stopping en base a la medición del AUC.

Para estudiar el proceso de convergencia y determinar si es necesario aplicar o no early stopping, se han realizado dos ensayos: en el primero se ha evaluado el comportamiento del proceso de convergencia al aumentar el número de nodos, construyendo sucesivas redes y aumentando el número de iteraciones desde 1 hasta 50. El valor de la tasa de aprendizaje es en este caso de 0,1. Para el segundo se ha repetido el ejercicio, pero evaluando la influencia de la tasa de aprendizaje, y se ha probado desde 1 hasta 25 iteraciones. El número de nodos en este caso se ha fijado en 11.

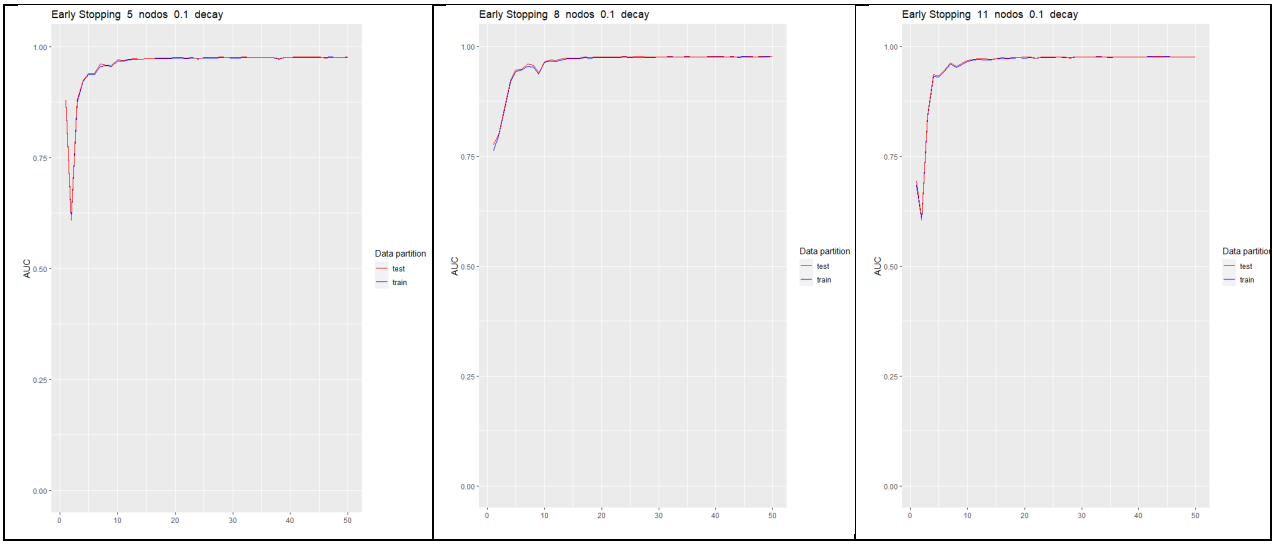
A pesar de que 50 y 25 pueden parecer pocas iteraciones, lo cierto es que en redes neuronales el sobreajuste (de haberlo), suele aparecer en etapas muy tempranas. Aun así, se ha comprobado que aumentar el número máximo de iteraciones hasta un límite de 200 no ofrece ninguna diferencia notable con respecto a los resultados obtenidos con 25 y 50 observaciones en ninguno de los dos casos, así que se asume que el comportamiento es estable.

El paquete caret no dispone de ninguna función diseñada para evaluar el efecto del early stopping. Por tanto, se ha creado la siguiente función dedicada a este propósito:

```
Early_Stopping <- function(maxiter, nsize = c(11), decay = c(0.1)) {
  Control <- trainControl(method = 'none', classProbs = TRUE, verboseIter =
FALSE, summaryFunction = twoClassSummary)
  iter <-  c(1:maxiter)
  ROCresults <- c()
  for (i in iter) {
    fit <- train(Y ~ ., data = train, metric = "ROC", method = 'nnet', maxit =
i, trControl = Control, tuneGrid=expand.grid(size=c(11), decay=c(0.1)))
    predtrain <- predict(fit, train, type = "prob")
    predtest <- predict(fit, test, type = "prob")
    ROCtrain <- roc(train$Y , predtrain$Yes, direction = "<")$auc
    ROCtest <- roc(test$Y , predtest$Yes, direction = "<")$auc
    ROCresults <- rbind(ROCresults, c(i, ROCtrain, ROCtest))
  }
  EarlyStop <- as.data.frame(ROCresults)

  ggplot(data = EarlyStop, aes(x = V1)) +
    geom_line(aes(y = V2, colour = "train")) +
    geom_line(aes(y = V3, colour = "test")) +
    scale_color_manual(name = "Data partition", values = c("train" = "blue",
"test" = "red")) +
    xlab(" ") +
    scale_y_continuous("AUC", limits = c(0, 1)) +
    labs(title= paste("Early Stopping ", nsize[1], " nodos ", decay[1], "
decay"))
}
```

Los resultados se muestran en la Tabla 7. Como se puede comprobar, en ningún caso se produce un sobreajuste notorio. Lo que sí se puede observar es que el proceso de convergencia parece más suave en el caso con 11 nodos, por eso se utilizará como valor de referencia en el segundo ensayo.



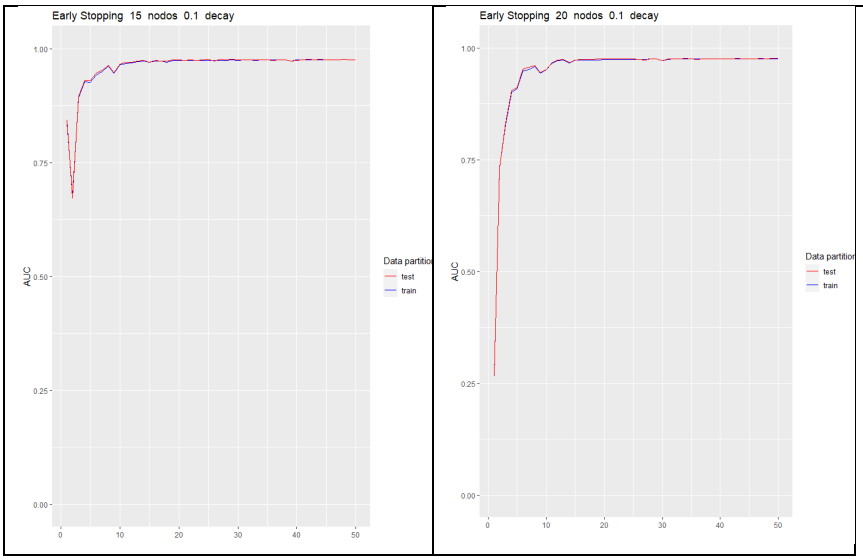


Tabla 7: Resultados del estudio de early stopping variando el número de nodos en la capa oculta.

En cuanto a la influencia de la tasa de aprendizaje, los resultados se muestran en la Tabla 8. Se comprueba que la tasa de aprendizaje tampoco llega a provocar sobreajuste en ningún caso.

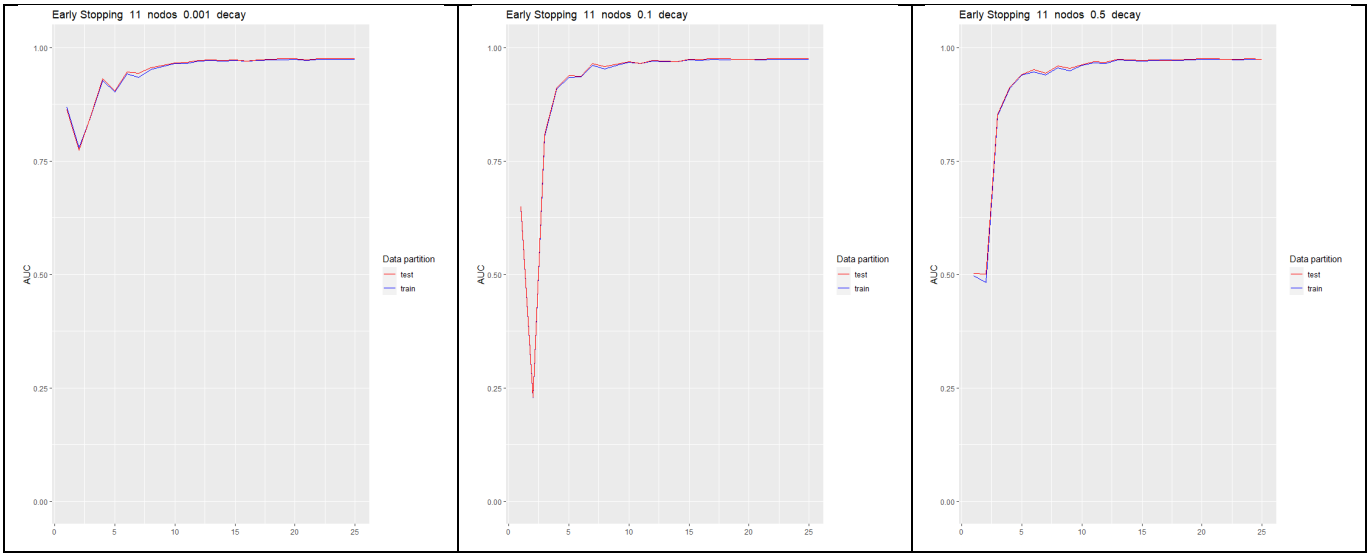


Tabla 8: Resultados del estudio de early stopping variando la tasa de aprendizaje.

Una vez estudiado el proceso de convergencia no parece necesario acudir al early stopping, puesto que ninguno de los casos mostrados presenta problemas de sobreajuste. De haberse localizado algún caso con sobreajuste acusado, se fijaría el número de iteraciones en un valor por debajo o igual al punto de early stopping.

El siguiente paso para construir la mejor red neuronal consiste en fijar el mejor valor de cada parámetro (tuneado). En este caso, dada la extensión del trabajo y la cantidad de pruebas a realizar, se ha decidido limitar este estudio a los dos hiperparámetros ya mencionados: el número de nodos en la capa oculta y la tasa de aprendizaje.

- Para el número de nodos se ha creado una rejilla o *grid* a través de la función de caret *expand.grid*.

A la hora de tunear los hiperparámetros de un algoritmo, es muy común utilizar un grid. Se trata simplemente de una matriz de valores para cada hiperparámetro personalizable de un algoritmo. Al llamar a la función *train* utilizando el grid, el algoritmo se entrena con cada combinación de valores presentes en la matriz. Esta información queda contenida dentro del objeto *train*, en el subobjeto *resample*. De esta forma, se puede comprobar qué combinación de valores ha resultado ser la más efectiva en la métrica definida. En la Figura 11 se muestra un pequeño esquema del uso de este tipo de herramientas.

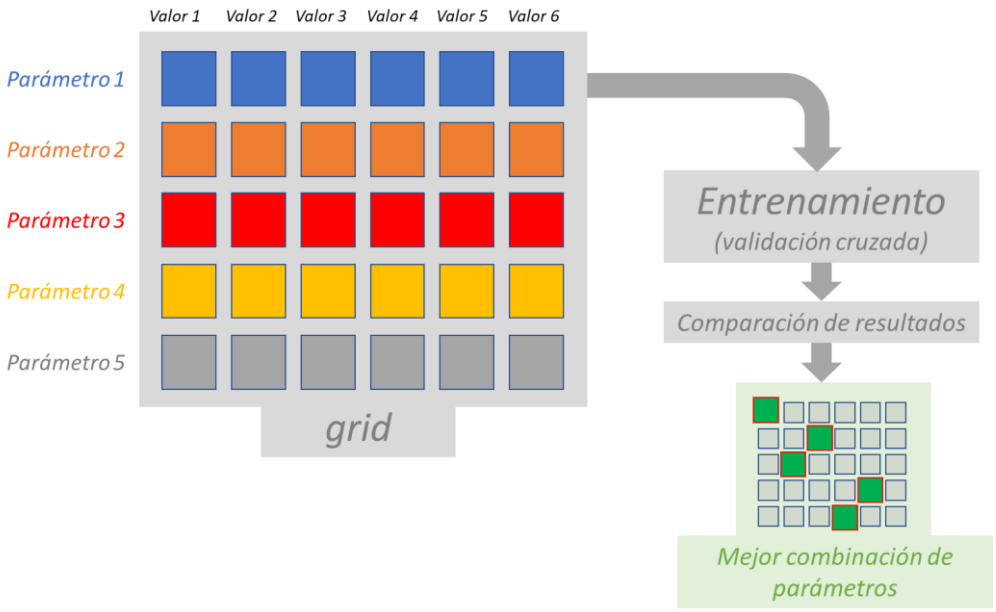


Tabla 9: Esquema del funcionamiento de un grid en el proceso de entrenamiento.

Existen varios tipos de estrategias de búsqueda con un grid. La manera clásica consiste en probar con todas las combinaciones disponibles en la matriz, pero otras, como la *randomized search* toman algunas combinaciones de la matriz de forma aleatoria. Son muy útiles cuando la matriz es muy grande y los tiempos de ejecución son excesivos.

En este caso el grid contiene 7 valores para el número de nodos: 5, 7, 9, 11, 13, 15 y 20. El código empleado es:

```
nodegrid <-  expand.grid(size=c(5,7,9,11,13,15,20), decay = c(0.5))
nodecontrol <- trainControl(method = "repeatedcv", number=5,
repeats=5,savePredictions = "all", classProbs = TRUE, verboseIter =
FALSE, returnResamp = "all", summaryFunction = twoClassSummary)
rednnet_1 <- train(Y ~ ., data = train, method="nnet", metric = "ROC",
maxit = 25, trControl = nodecontrol, tuneGrid = nodegrid)
```

Como se puede comprobar este grid solamente sirve para estudiar el número de nodos como parámetro de forma aislada. Este tipo de acercamiento al problema puede resultar erróneo, puesto que muchas veces un hiperparámetro puede condicionar el comportamiento de los demás, y deben estudiarse de forma conjunta.

Aun así, se ha ejecutado dicho código, que ha realizado el entrenamiento de la red mediante validación cruzada repetida con la rejilla descrita. Después se han representado los resultados para determinar el número de nodos óptimo a través de un diagrama de cajas. El diagrama (Figura 12) muestra el valor del AUC en función del número de nodos seleccionado.

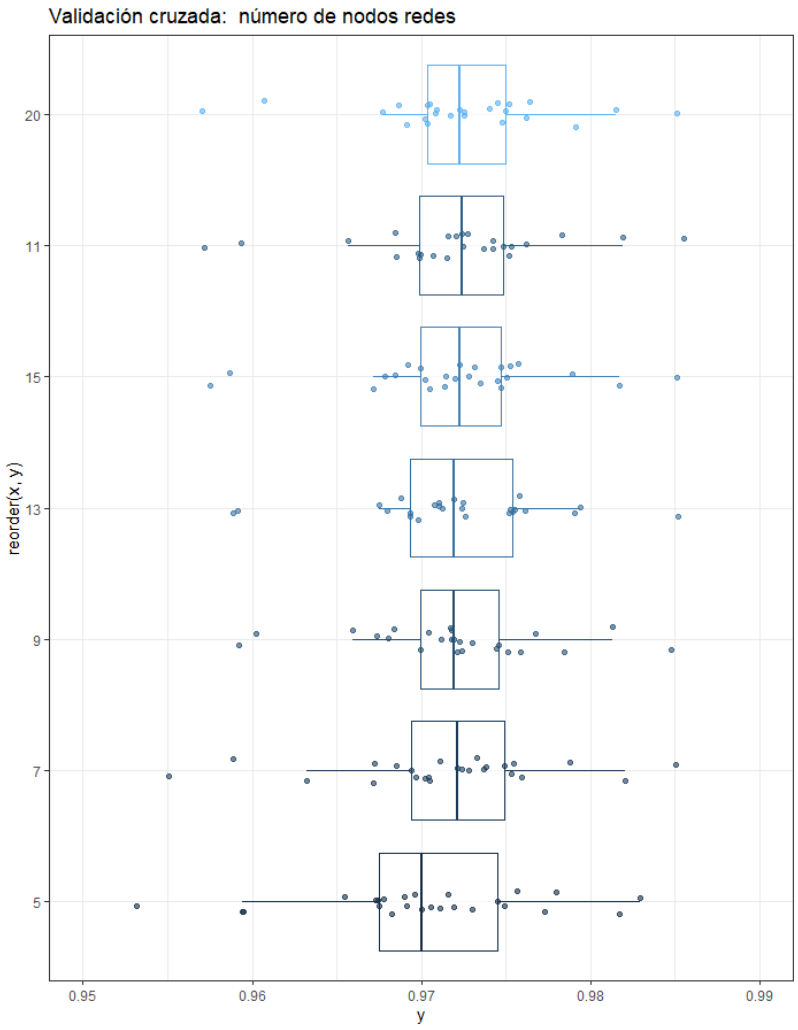


Figura 12: Diagrama de cajas para estudio del número de nodos en redes neuronales.

A la vista de los resultados, todas las redes parecen muy igualadas a nivel de sesgo y varianza. Quizá el caso más equilibrado sea con 9 nodos, así que será elegido como número óptimo.

- En cuanto a la tasa de aprendizaje, se ha realizado el mismo procedimiento, definiendo un grid y utilizando 9 nodos como valor de referencia.

  En este caso el grid contiene un total de cuatro valores para la tasa de aprendizaje: 0.5, 0.1, 0.01 y 0.001. Los resultados se han graficado en la Figura 13, en forma de diagrama de cajas.

A nivel de sesgo los resultados son muy similares. A nivel de varianza parece que la opción con 0.1 es la que mejores resultados obtiene. Por tanto, parece que la opción más adecuada se da con una tasa de aprendizaje de 0.1.

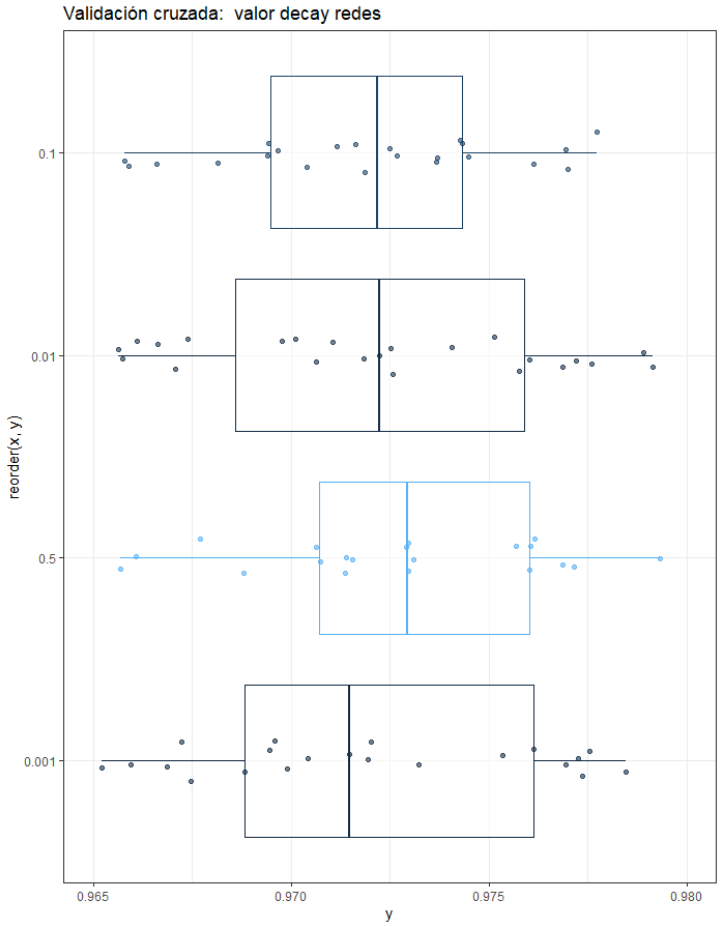


Figura 13. Diagrama de cajas del estudio de la tasa de aprendizaje en redes neuronales.

Por tanto, analizando cada hiperparámetro por separado, da la sensación de que usar 9 nodos y una tasa de aprendizaje de 0.1 es la mejor combinación posible. Sin embargo, como se ha venido diciendo, evaluar la influencia de cada hiperparámetro por separado no es la mejor estrategia.

Por eso, se ha construido un grid de mayor tamaño, incluyendo los siete valores del número de nodos y los cuatro valores de la tasa de aprendizaje. En total, se tiene una matriz con 28 combinaciones posibles de hiperparámetros, que se ha entrenado con validación cruzada repetida con cinco grupos y cinco iteraciones. En total, se han entrenado 700 modelos de redes neuronales. Además, cada red neuronal se ha limitado a un número máximo de iteraciones en el proceso de convergencia de 25, para reducir el tiempo de cálculo. Tras haber estudiado los efectos de early stopping parece que 25 iteraciones serán suficientes.

Se han representado los resultados de las seis mejores combinaciones de parámetros (en términos del AUC medio) a través del siguiente código:

```
REDgrid <-  expand.grid(size=c(5,7,9,11,13,15,20),decay=c(0.5,0.1,0.01,0.001))
REDcontrol <- trainControl(method = "repeatedcv",number = 5, repeats
=5,savePredictions = "all", classProbs = TRUE, verboseIter = FALSE,
returnResamp = "all", summaryFunction = twoClassSummary)
modeloRED <- train(Y ~ ., data = train, method = "nnet", metric = "ROC", maxit
= 25, trControl = REDcontrol, tuneGrid = REDgrid)
resultado_resample_red <- as.data.frame(modeloRED$resample)
resultado_resample_red$concat <- paste(resultado_resample_red$size, " nodes ",
resultado_resample_red$decay, " decay")
resultado_resample_red$Mean <- ave(resultado_resample_red$ROC,
resultado_resample_red$concat)
nnet_results <-
head(resultado_resample_red[order(resultado_resample_red$Mean),], 150)
model_boxplot(data = nnet_results, x = nnet_results$concat, y =
nnet_results$ROC, title = "valor decay redes", ymin = 0.965, ymax = 0.98)
```

El diagrama de cajas resultante se ha representado en la figura.

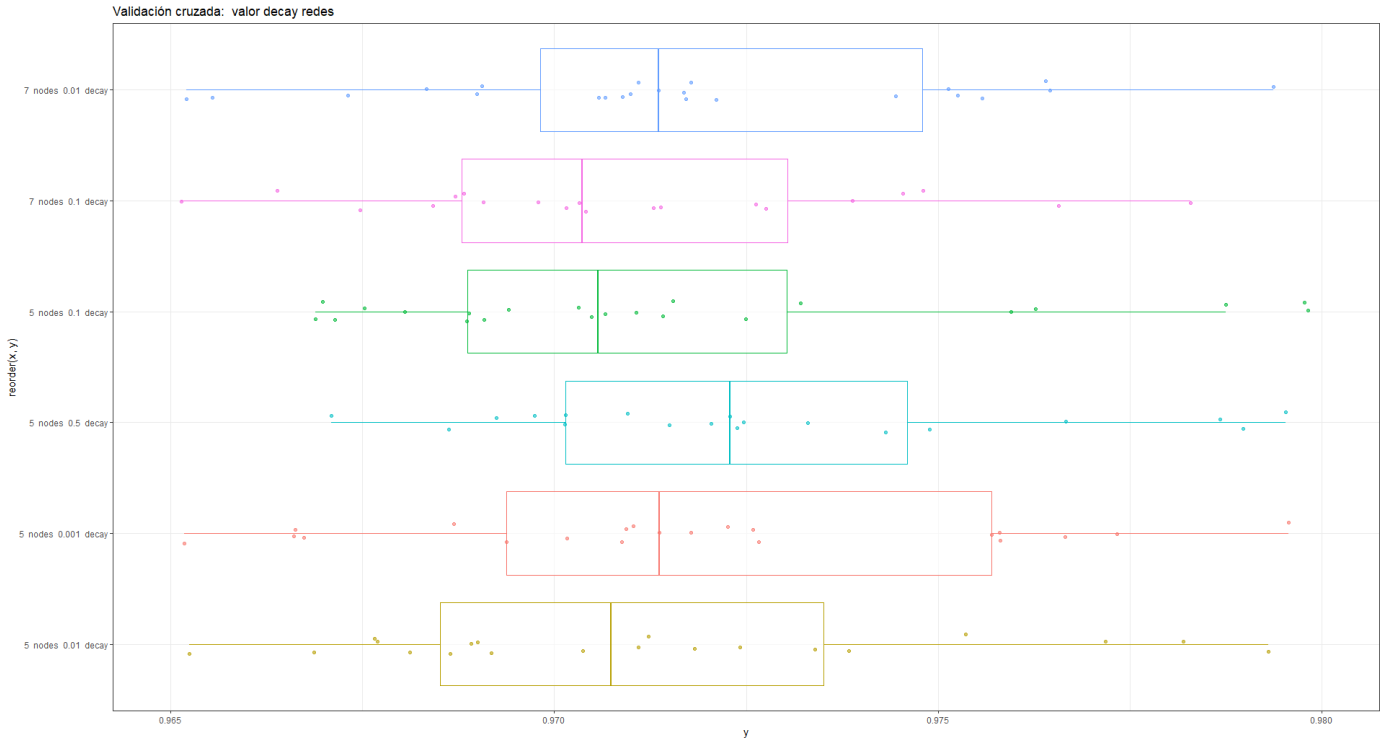


Figura 14: Diagrama de cajas de las seis mejores combinaciones de redes neuronales.

Observando el diagrama, parece que la mejor opción tanto en sesgo como en varianza resulta ser la de color azul claro (la cuarta empezando desde arriba).

Por tanto, la red neuronal que pasará a la fase final de evaluación de modelos está construida con 5 nodos en la capa oculta y una tasa de aprendizaje de 0.5.

6.4. Random Forest

Cronológicamente hablando, el algoritmo de random forest es una evolución de otro algoritmo desarrollado conocido como *Bagging* (aunque en la práctica, bagging es un caso particular de random forest).

La idea bajo el funcionamiento de estos algoritmos consiste en la combinación de las predicciones de múltiples árboles de decisión. Como se comentó en su apartado, el árbol de decisión es un modelo clásico ampliamente utilizado por su sencillez, interpretabilidad y por su adaptabilidad a todo tipo de datos, incluso sin tener que realizar una depuración muy exhaustiva. Sin embargo, también presenta una serie de problemas como un sesgo elevado y la escasa suavidad en las predicciones obtenidas (se obtienen resultados "bruscos", especialmente en problemas de regresión).

La técnica de random forest permite solventar estos defectos por medio de la aleatoriedad y la combinación de árboles. Por tanto, este algoritmo entra dentro de las conocidas como técnicas de ensamblado, pues combina las predicciones de modelos más sencillos para lograr una predicción compleja.

La idea consiste en realizar un muestreo por reemplazamiento promediado con un número determinado de árboles. En cada paso se genera un sorteo sobre las observaciones del conjunto de datos, dando lugar a un subconjunto de entrenamiento y otro de prueba. Se utilizan ambos subconjuntos para entrenar un modelo de árbol y generar predicciones ($\hat{y}_i$), respectivamente. Por último, las predicciones obtenidas se promedian, obteniendo la predicción final ($\hat{y}_f$). Cuanta mayor cantidad haya de árboles generados, más suave será la solución ofrecida, aunque hay peligro de alcanzar sobreajuste.

De esta forma, el resultado es mucho más robusto y suave que usando un simple árbol de decisión, puesto que el método de random forest recoge la variabilidad de numerosos árboles y muestras diferentes.

Además, este método admite la posibilidad de realizar *bootstrap*, es decir, de realizar reemplazamiento en el sorteo de las observaciones. Esto quiere decir que se puede repetir varias veces la misma observación dentro de cada subconjunto de entrenamiento o prueba. Esta técnica mejora bastante los resultados, introduciendo mucha más variabilidad y posibilidades en la combinación de árboles.

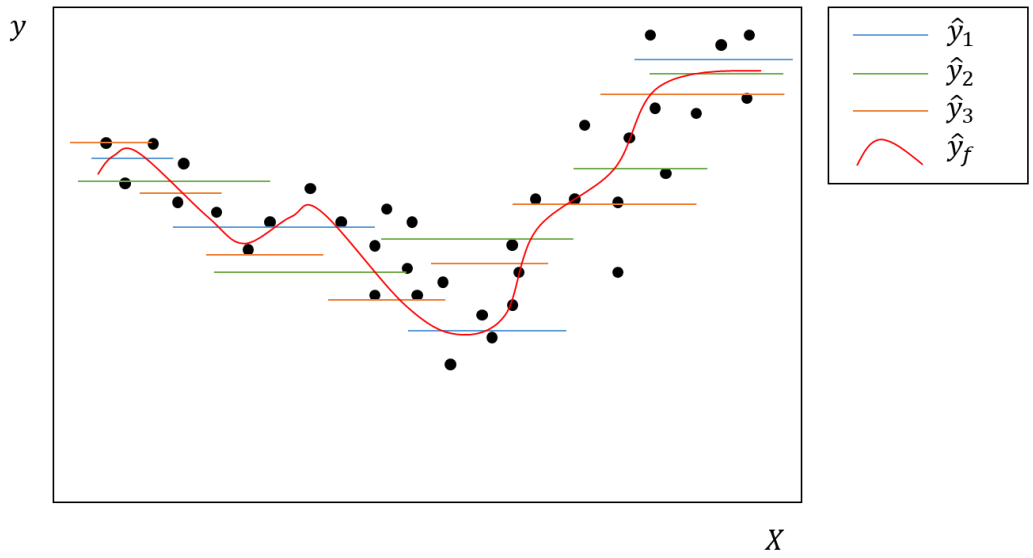


Figura 15: Esquema de funcionamiento de random forest. Puntos negors: representan los valores reales de la variable objetivo. Líneas horizontales: son las predicciones obtenidas con cada árbol de decisión. Curva roja: predicción media.

Además de todo esto, random forest permite aumentar la combinatoria a través del sorteo de las variables implicadas en cada árbol. Es decir, cada árbol generado en el algoritmo de random forest accede a una combinación única de observaciones y variables diferentes, dando resultados únicos. Esto favorece la disminución del sesgo, pero puede provocar mayor sobreajuste.

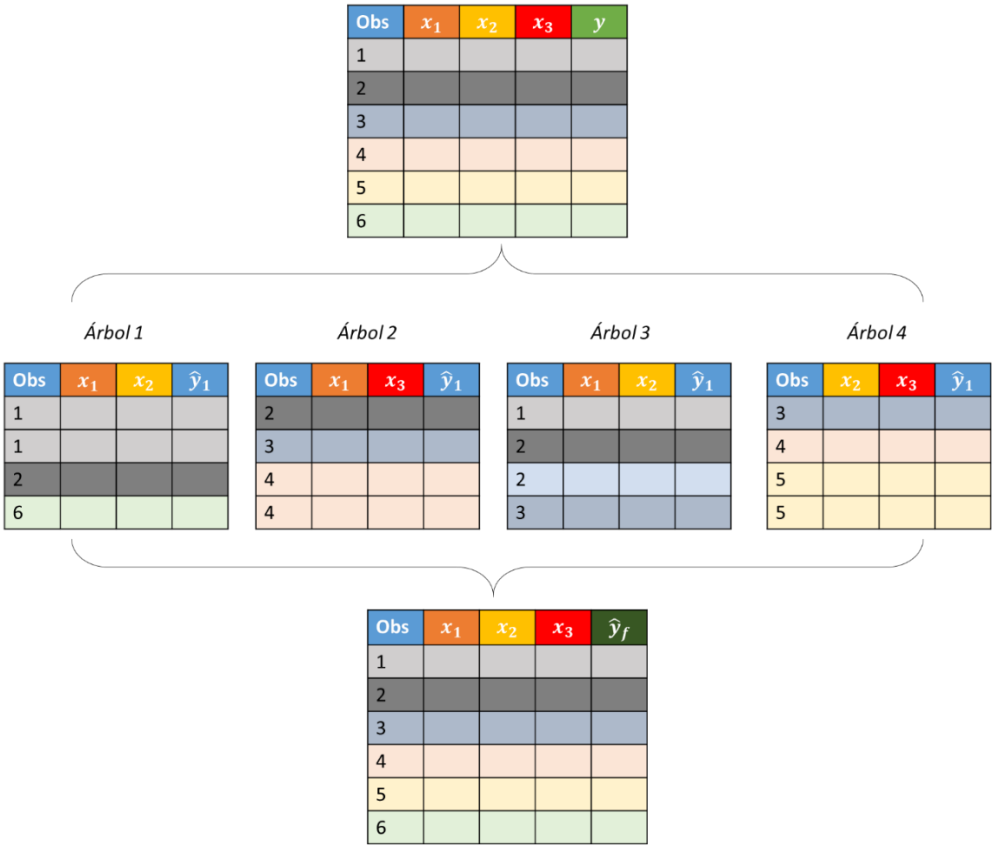


Figura 16. Esquema del sistema de sorteo de observaciones y variables en random forest.

Por tanto, los hiperparámetros a controlar en el algoritmo de random forest son el número de árboles a promediar, el número de variables a sortear, la proporción de observaciones a incluir en cada árbol, la opción de incluir o no bootstrap (reemplazamiento) en el sorteo de observaciones y las opciones de poda de cada árbol.

A continuación, se va a describir el desarrollo de la parte técnica asociada a random forest en nuestro modelo de lead scoring. Para tunear este algoritmo, se va a comenzar analizando cada hiperparámetro de forma separada. Posteriormente, se creará una rejilla para realizar la parametrización conjunta para buscar la mejor combinación posible (*Aplicación de algoritmos Random Forest y XGBoost en una base de solicitudes de tarjetas de crédito*, s. f.).

- El primer hiperparámetro a tunear es el número de árboles. Como se ha mencionado, este algoritmo basa su funcionamiento en promediar las predicciones de múltiples árboles de decisión. Un número muy elevado de árboles puede llegar a provocar sobreajuste, mientras que un número bajo puede hacer que la afectar a la precisión del modelo.

  En este caso, se ha ensayado utilizando un número de árboles comprendido entre 10 y 300. En caret, este parámetro se conoce como *ntree*. El proceso se ha programado través del siguiente bucle:

```
rfcontrol<-trainControl(method = "none", savePredictions = "all",
classProbs = TRUE, verboseIter = FALSE, returnResamp = "all",
summaryFunction = twoClassSummary)
rfROCresults <- c()
for (i in c(1:30)) {
  rf_default <- train(Y~., data = train, method = 'rf', metric = 'ROC',
ntree = i*10, trControl = rfcontrol)
  rfpredtrain <- predict(rf_default, train, type = "prob")
  rfpredtest <- predict(rf_default, test, type = "prob")
  rfROCtrain <- roc(train$Y , rfpredtrain$Yes, direction = "<")$auc
  rfROCtest <- roc(test$Y , rfpredtest$Yes, direction = "<")$auc
  rfROCresults <- rbind(rfROCresults, c(i*10, rfROCtrain, rfROCtest))
}
rfResults <- as.data.frame(rfROCresults)
rfResults
ggplot(data = rfResults, aes(x = V1)) +
  geom_line(aes(y = V2, colour = "train")) +
  geom_line(aes(y = V3, colour = "test")) +
  scale_color_manual(name = "Data partition", values = c("train" =
"blue", "test" = "red")) +
  xlab(" ") +
  scale_y_continuous("AUC", limits = c(0, 1)) +
  labs(title= "AUC - número de árboles RF") +
  theme(title = element_text(size = 20))
```

  Como se observa en el código, los resultados se han mostrado en una gráfica (Figura 17), donde se representa el valor del AUC en función del número de árboles. Además, se ha diferenciado entre el conjunto de entrenamiento y el de prueba.

A la luz de los resultados queda patente que el modelo es muy estable y no presenta sobreajuste. Con menos de 50 árboles ya se obtiene un valor del AUC muy elevado, y se mantiene prácticamente constante.

Por tanto, dada la distribución de los datos, parece que el rendimiento del modelo es, hasta cierto punto, independiente del número de árboles.

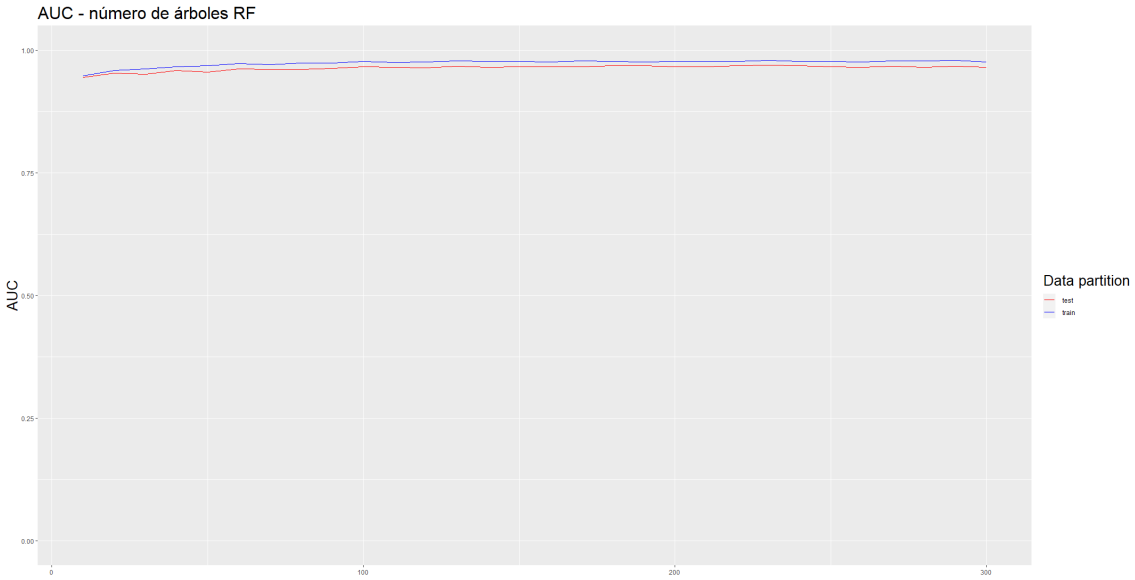


Figura 17: Variación del AUC en entrenamiento y prueba en función del número de árboles en random forest.

- El siguiente parámetro que se va a tunear es el tamaño de la muestra. Se recuerda que este parámetro hace referencia a la cantidad de datos utilizados en el entrenamiento de cada árbol. En general, utilizar muestras muy pequeñas puede aumentar la varianza, mientras que usar muestras muy grandes puede provocar que las predicciones de los árboles sean muy similares, aumentando el sesgo.

  Para determinar el tamaño de la muestra se ha empleado un bucle que evalúa el AUC obtenido en entrenamiento y en prueba, en función de dicho parámetro, entrenando el algoritmo con un valor que va desde las 1000 observaciones hasta las 6000. En caret, el tamaño de muestra se denomina *sampsize*. Este parámetro se puede usar en forma de número entero (lo cual expresa el número de observaciones a incluir) o en tanto por uno (expresando la proporción de observaciones con respecto al total).

```
rfsampresults <- c()
rfgrid <-expand.grid(.mtry = 5)
for (i in 1000*c(1:6)) {
  print(i)
  rf_default <- train(Y ~ ., data = train, method = "rf", metric =
"ROC", ntree = 100, sampsize = i, trControl = rfcontrol, tuneGrid =
rfgrid)
  rfpredtrain <- predict(rf_default, train, type = "prob")
  rfpredtest <- predict(rf_default, test, type = "prob")
  rfROCtrain <- roc(train$Y , rfpredtrain$Yes, direction = "<")$auc
  rfROCtest <- roc(test$Y , rfpredtest$Yes, direction = "<")$auc
  rfsampresults <- rbind(rfsampresults, c(i, rfROCtrain, rfROCtest))
}
```

```
rfsampResults <- as.data.frame(rfsampresults)
rfsampResults

ggplot(data = rfsampResults, aes(x = V1)) +
  geom_line(aes(y = V2, colour = "train")) +
  geom_line(aes(y = V3, colour = "test")) +
  scale_color_manual(name = "Data partition", values = c("train" =
"blue", "test" = "red")) +
  xlab(" ") +
  scale_y_continuous("AUC", limits = c(0, 1)) +
  labs(title= "AUC - tamaño muestra RF") +
  theme(title = element_text(size = 20))
```

Los resultados se han graficado en la Figura 18. Se observa que, de nuevo, el AUC se mantiene casi constante. Por tanto, parece que el dataset es tan homogéneo que ofrece buenos resultados independientemente del valor de este parámetro. Esto rara vez ocurre en un caso real, donde los datos tienden a estar muy desbalanceados y resulta difícil hallar el corte correcto para cada parámetro.

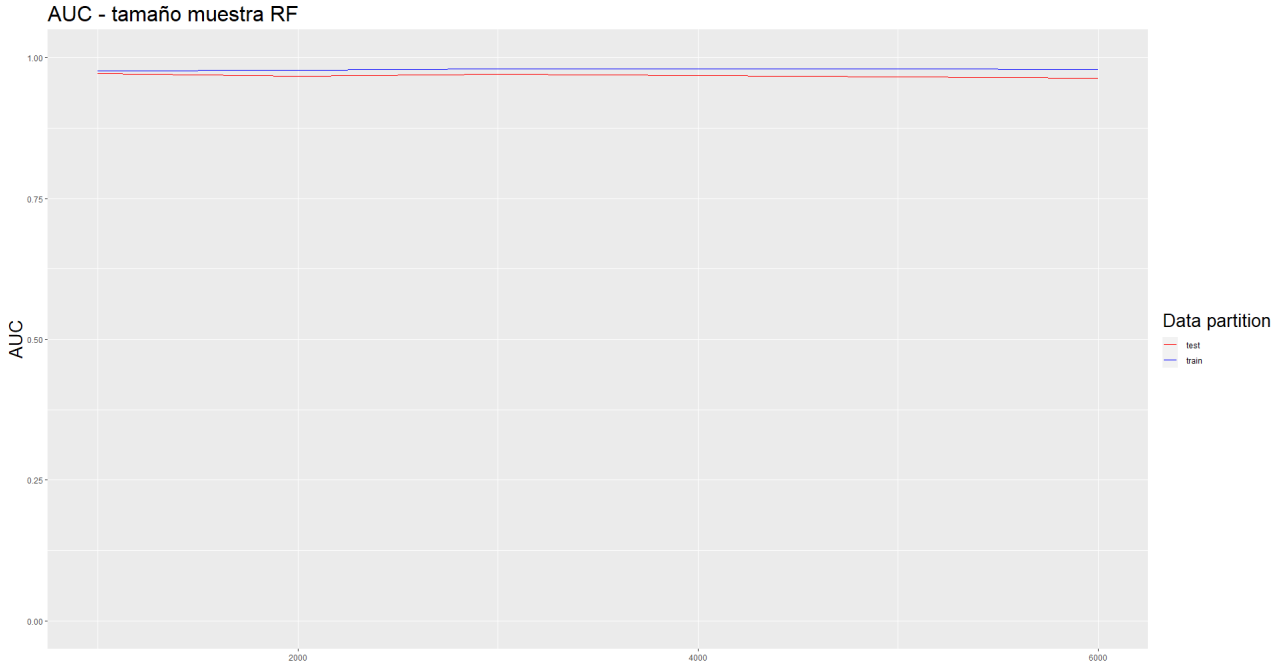


Figura 18: Gráfico de variación del AUC en función del número de observaciones incluidas en cada árbol en random forest.

- En cuanto al número de variables, este hiperparámetro también es fundamental. Dependiendo del conjunto de datos utilizado y de la importancia relativa de cada variable input, puede ser aconsejable utilizar todas las variables o reducir el número para realizar el sorteo. En teoría, reducir el número de variables puede hacer que el algoritmo sea más preciso, pero también puede aumentar la varianza.

  En el paquete caret, este hiperparámetro se llama *mtry*. Para optimizarlo, se ha hecho uso de un pequeño grid, donde se ha probado desde 2 variables hasta el total de las 13 variables. Los resultados se mostrarán en forma de diagrama de cajas (Figura 19).

Los resultados, una vez más, están muy igualados. No parece haber ningún patrón claro en cuanto a la respuesta del AUC en función del número de variables empleadas en entrenamiento. Da la sensación de que, con menos de 5 variables el modelo presenta mayor varianza, pero las diferencias no son muy grandes. La opción más equilibrada parece ser con un total de 9 variables.

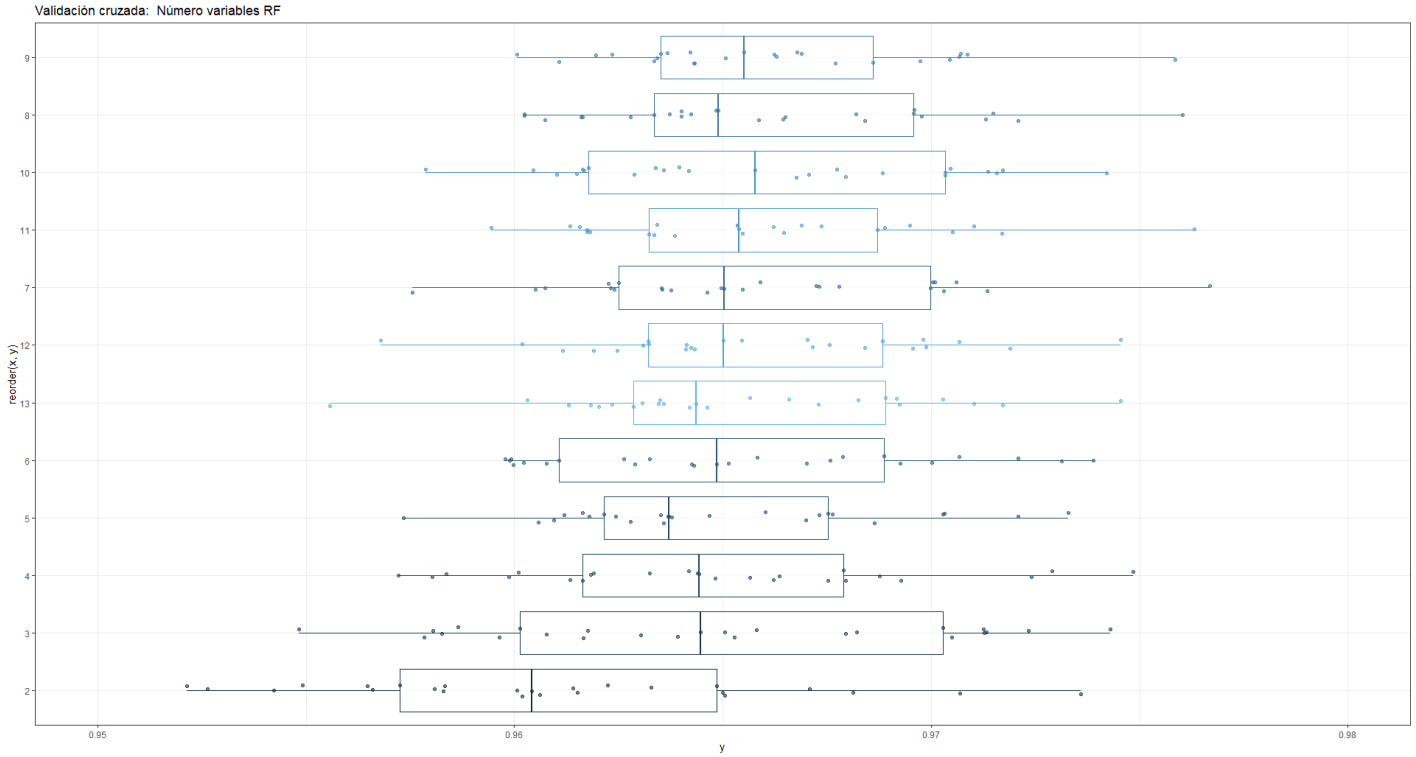


Figura 19: Diagrama de cajas de comparación del AUC frente al número de variables incluidas en cada árbol en random forest.

Dado que los hiperparámetros del tamaño de la muestra y el número de árboles no parecen afectar a la predicción del modelo, se ha omitido la construcción de un grid general con todos los hiperparámetros. **Se ha seleccionado como mejor modelo de Random Forest el construido con un total de 100 árboles, 1000 observaciones como tamaño de la muestra y un total de 9 variables a sortear. Este es el que pasará a la fase final de evaluación.**

6.5. Modelos de Gradient Boosting

6.5.1. Gradient Boosting (GBM)

El algoritmo de *Gradient Boosting* es uno de los más potentes y agresivos en términos de predicción. Se trata de otro método que utiliza una técnica de ensamblado con predictores débiles, como árboles de decisión sencillos (aunque también se pueden usar otras opciones distintas a los árboles) y nació como una solución ingeniosa durante un concurso de predicción.

Se dice que es un modelo agresivo porque ataca directamente al error de predicción, intentando minimizarlo. De forma similar a lo que hace el método de descenso del gradiente usado en redes neuronales, la idea es utilizar una función de coste (como puede ser el error cuadrático medio en regresión o la función *deviance* para clasificación) cuyo valor se hará descender intentando dirigir su avance en la dirección de descenso del gradiente. Para ello se utilizarán árboles para predecir el valor del residuo (es decir, la diferencia entre el valor de la variable objetivo y su predicción) en cada paso del algoritmo, actualizando los valores de la variable objetivo con dicho residuo.

Como se ha dicho, se trata de un método que ataca directamente al error de predicción, con lo que consigue resultados muy buenos, pero también puede caer en problemas de sobreajuste con facilidad. Además, es un método al que beneficia el uso de early stopping puesto, que aumentar el número de iteraciones por encima de cierto umbral en algunas ocasiones aumenta la tendencia al sobreajuste.

A continuación, se va a tunear este algoritmo para encontrar la mejor configuración que se adapte a al dataset de estudio. Para ello se intentará encontrar el valor óptimo de cada hiperparámetro.

- El primer hiperparámetro a tener en cuenta es la constante de regularización o *shrinkage*. Este valor permite escalar el avance del gradiente de la función de coste. Un valor muy elevado acelera la convergencia del modelo, pero puede no encontrar el mínimo. Un valor muy bajo puede encontrar mejor el mínimo, pero también puede provocar que se caiga en mínimos locales.

  En este caso, se ha utilizado un grid para encontrar el valor o rango de valores más adecuado para esta constante. El grid se ha parametrizado con un número de árboles igual a 100, un número mínimo de observaciones por nodo en cada árbol de 20 y una profundidad de 2.

```
shrinkgrid <- expand.grid(shrinkage = c(0.05, 0.1, 0.2, 0.3, 0.4, 0.5),
n.trees = 100, n.minobsinnode = 20, interaction.depth = 2)
shrinkcontrol <- trainControl(method = "repeatedcv",number = 5, repeats
=5,savePredictions = "all", classProbs = TRUE, verboseIter = FALSE,
returnResamp = "all", summaryFunction = twoClassSummary)
shrinktrain <- train(Y ~ .,data = train, method = "gbm", metric = "ROC",
trControl = shrinkcontrol, tuneGrid = shrinkgrid, distribution =
"bernoulli", verbose = FALSE)
shrink_resample <- as.data.frame(shrinktrain$resample)
```

```
model_boxplot(data = shrink_resample, x = shrink_resample$shrinkage, y =
shrink_resample$ROC, title = "Valor shrinkage GBM", ymin = 0.945, ymax =
0.98)
```

Los resultados se han graficado en forma de diagrama de cajas (Figura 20). Observando la figura se puede notar que valores muy bajos de la constante obtienen peores resultados en términos de varianza. Los modelos generados a con un valor igual o mayor a 0,2 parecen prácticamente empatados tanto en sesgo como en varianza, así que cualquier opción dentro de ese rango podría ser válida.

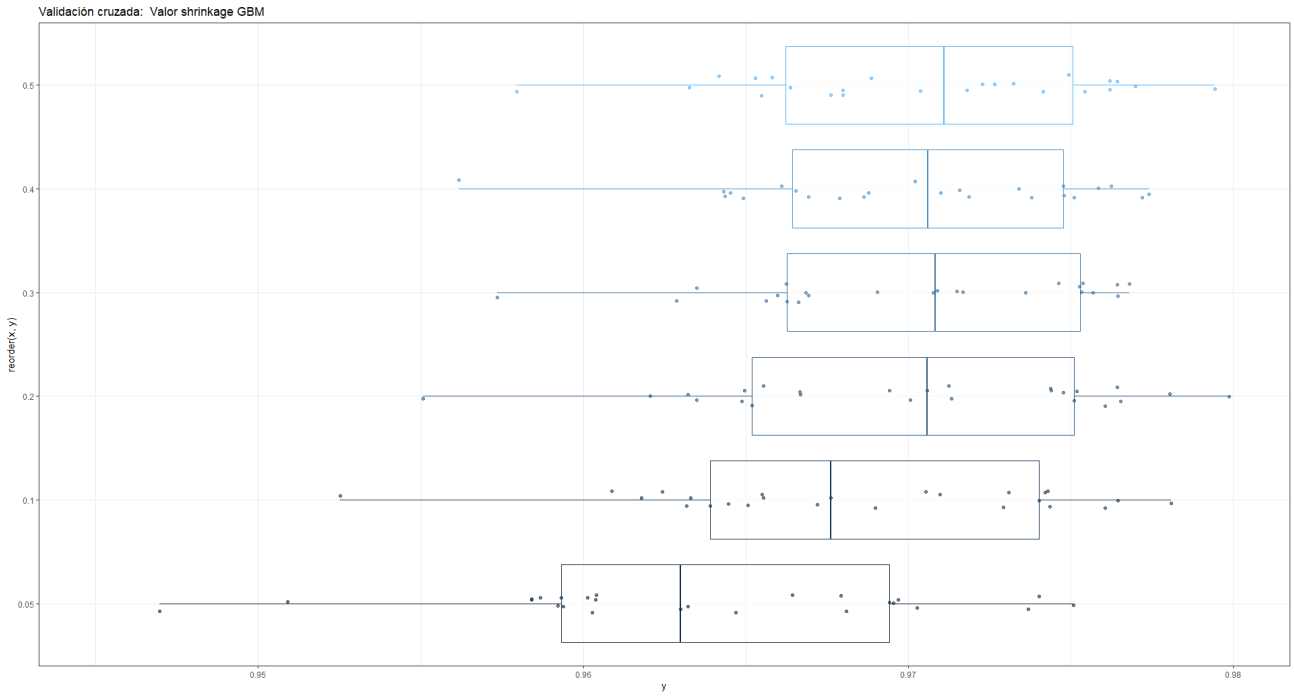


Figura 20: Diagrama de cajas de comparación del AUC frente al valor del shrinkage en gradient boosting.

- El siguiente parámetro a tunear es el número de iteraciones. Este parámetro hace referencia al número de ciclos de predicción (y, por lo tanto, el número de árboles empleados) para realizar el proceso de convergencia de la función de coste. En caret este parámetro se conoce como *n.tree.*

  Para encontrar el número óptimo se ha ensayado con un grid que varía el número de iteraciones desde 10 hasta 200. Se ha empleado un código muy similar al anterior, dejando el resto de parámetros constantes:

```
ntreegrid <- expand.grid(shrinkage = 0.2, n.trees = 10*c(1:20),
n.minobsinnode = 20, interaction.depth = 2)
ntreecontrol <- trainControl(method = "repeatedcv",number = 5, repeats
=5,savePredictions = "all", classProbs = TRUE, verboseIter = FALSE,
returnResamp = "all", summaryFunction = twoClassSummary)
ntreetrain <- train(Y ~ .,data = train, method = "gbm", metric = "ROC",
trControl = ntreecontrol, tuneGrid = ntreegrid, distribution =
"bernoulli", verbose = FALSE)
ntree_resample <- as.data.frame(ntreetrain$resample)
model_boxplot(data = ntree_resample, x = ntree_resample$n.trees, y =
ntree_resample$ROC, title = "Valor ntrees GBM", ymin = 0.945, ymax =
0.98)
```

En la Figura 21 se muestran los resultados obtenidos. A tenor del gráfico se puede concluir que el modelo comienza a estabilizarse a partir de las 70 iteraciones. Utilizar más de 140 iteraciones no parece traer beneficios a nivel de AUC, así que se entiende que el modelo ajusta suficientemente bien con apenas 100-140 iteraciones.

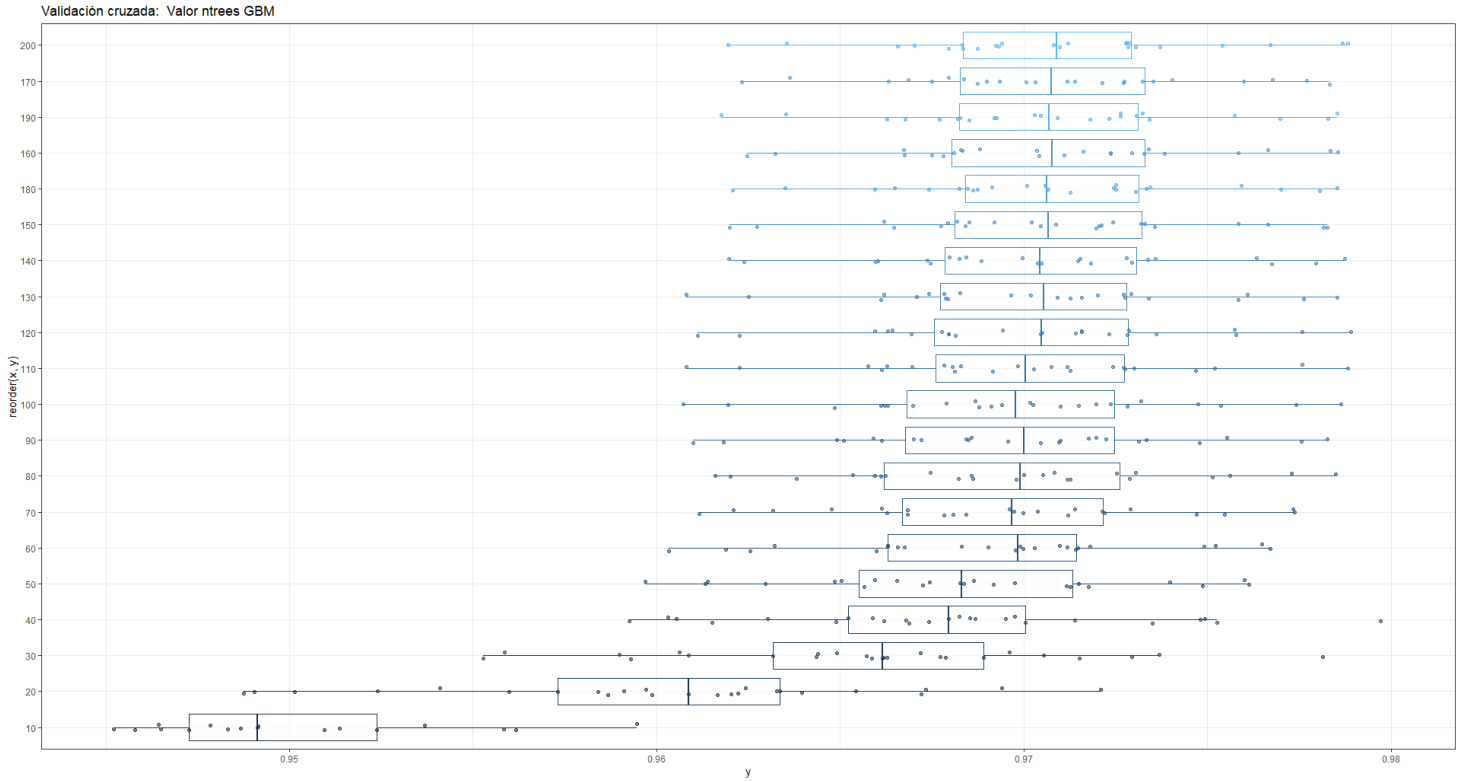


Figura 21: Diagrama de cajas de comparación del AUC frente al valor del número de árboles en gradient boosting.

- El último parámetro que se va a tunar es el número mínimo de observaciones en cada nodo de los árboles de decisión. Este parámetro se refiere a la cantidad mínima de observaciones que pueden incluirse en un nodo de corte en el proceso de generación de árboles que se utiliza en el proceso de convergencia. En los árboles de decisión, usar una cantidad muy baja de este parámetro puede producir sobreajuste, mientras que un número elevado puede no encontrar el corte óptimo.

  Se ha vuelto a usar un grid para encontrar el mejor valor. En este caso, limitando el intervalo a un máximo de 50 observaciones por nodo:

```
nobsgrid <- expand.grid(shrinkage = 0.2, n.trees = 150, n.minobsinnode =
2 * c(1:25), interaction.depth = 2)
nobscontrol <- trainControl(method = "repeatedcv",number = 5, repeats
=5,savePredictions = "all", classProbs = TRUE, verboseIter = FALSE,
returnResamp = "all", summaryFunction = twoClassSummary)
nobstrain <- train(Y ~ .,data = train, method = "gbm", metric = "ROC",
trControl = nobscontrol, tuneGrid = nobsgrid, distribution =
"bernoulli", verbose = FALSE)
nobs_resample <- as.data.frame(nobstrain$resample)
```

```
model_boxplot(data = nobs_resample, x = nobs_resample$n.minobsinnode, y
= nobs_resample$ROC, title = "Valor observaciones mínimas por nodo GBM",
ymin = 0.945, ymax = 0.98)
```

Los resultados se muestran en el diagrama de cajas de la Figura 22. Parece que utilizar un número de observaciones comprendido entre 10 y 40 es lo más adecuado tanto en sesgo como en varianza. Números más elevados hacen que el modelo no ajuste tan bien, aumentando en gran medida sesgo y varianza.

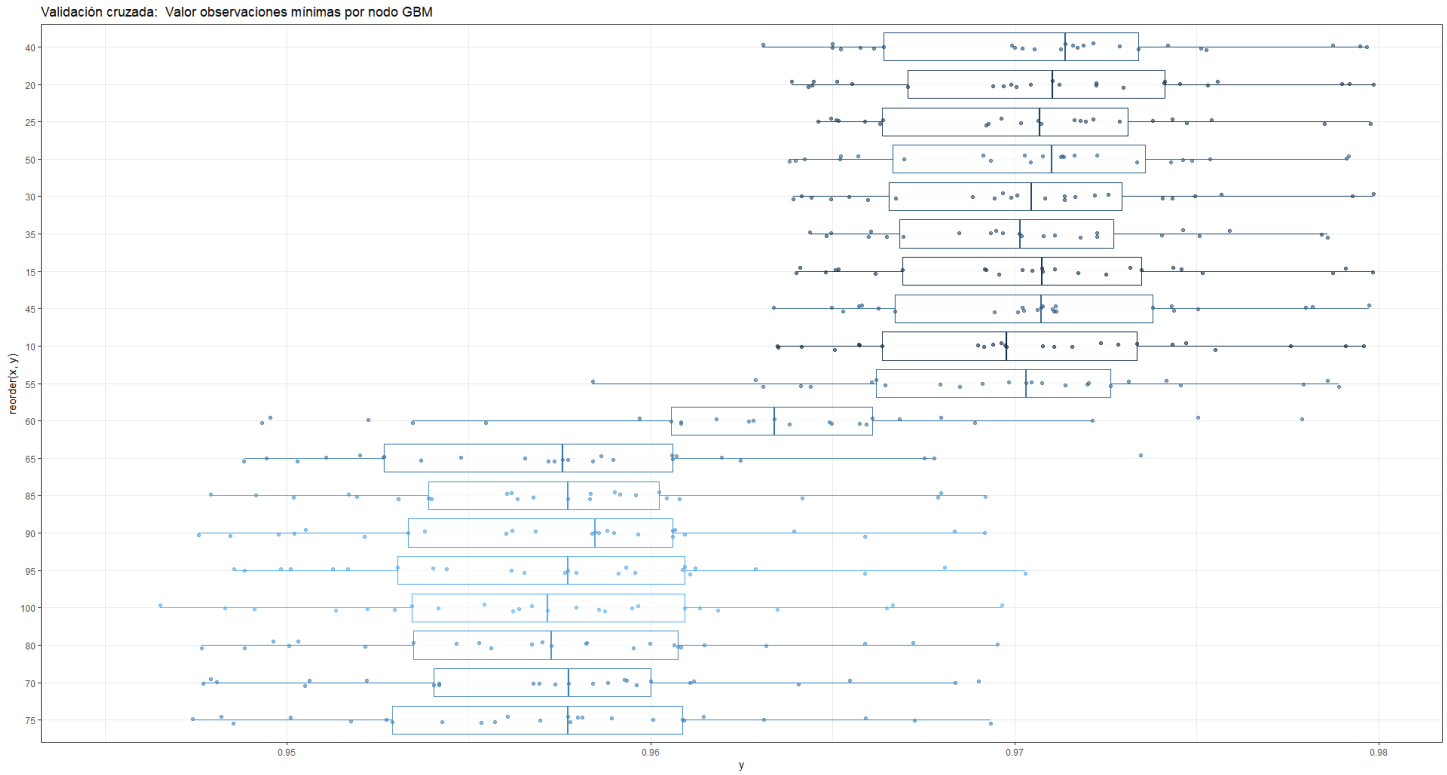


Figura 22: Diagrama de cajas de comparación del AUC frente al número de observaciones mínimas por nodo en GBM.

Se podrían tunear muchos otros parámetros además de los mencionados, pero viendo los resultados parece que el modelo de gradient boosting obtiene resultados muy prometedores, con poco margen de mejora. Además, no hay que olvidar que en este trabajo no se pretende llevar un análisis tan exhaustivo como se haría en un entorno real, donde se dedicarían decenas de horas de esfuerzo en intentar arañar algunas milésimas más en cuanto a precisión.

Pero todavía no se ha seleccionado la mejor combinación de los parámetros mencionados. Para poder tomar dicha decisión se va a utilizar un grid general que encuentre la mejor combinación de parámetros. Cabe señala que se ha decidido fijar el mínimo de observaciones por nodo en 20, puesto que ya hemos visto que en el rango de 10-40, este parámetro no tiene influencia en la calidad de las predicciones. Además, se ha añadido un cuarto parámetro referente a la profundidad de los árboles.

```
refgrid <- expand.grid(shrinkage = c(0.2, 0.3, 0.4), n.trees = c(100, 150,
200), n.minobsinnode = 20, interaction.depth = c(2, 3))
```

```
refcontrol <- trainControl(method = "repeatedcv",number = 5, repeats
=5,savePredictions = "all", classProbs = TRUE, verboseIter = FALSE,
returnResamp = "all", summaryFunction = twoClassSummary)
reftrain <- train(Y ~ .,data = train, method = "gbm", metric = "ROC",
trControl = refcontrol, tuneGrid = refgrid, distribution = "bernoulli",
verbose = FALSE)
ref_resample <- as.data.frame(reftrain$resample)
ref_resample$concat <- paste("shrink ", ref_resample$shrinkage, " ntrees ",
ref_resample$n.trees, " depth ", ref_resample$interaction.depth)
model_boxplot(data = ref_resample, x = ref_resample$concat, y =
ref_resample$ROC, title = "Mejor combinación GBM", ymin = 0.96, ymax = 0.98)
```

De nuevo, los resultados se han representado en forma de diagrama de cajas (Figura 23).

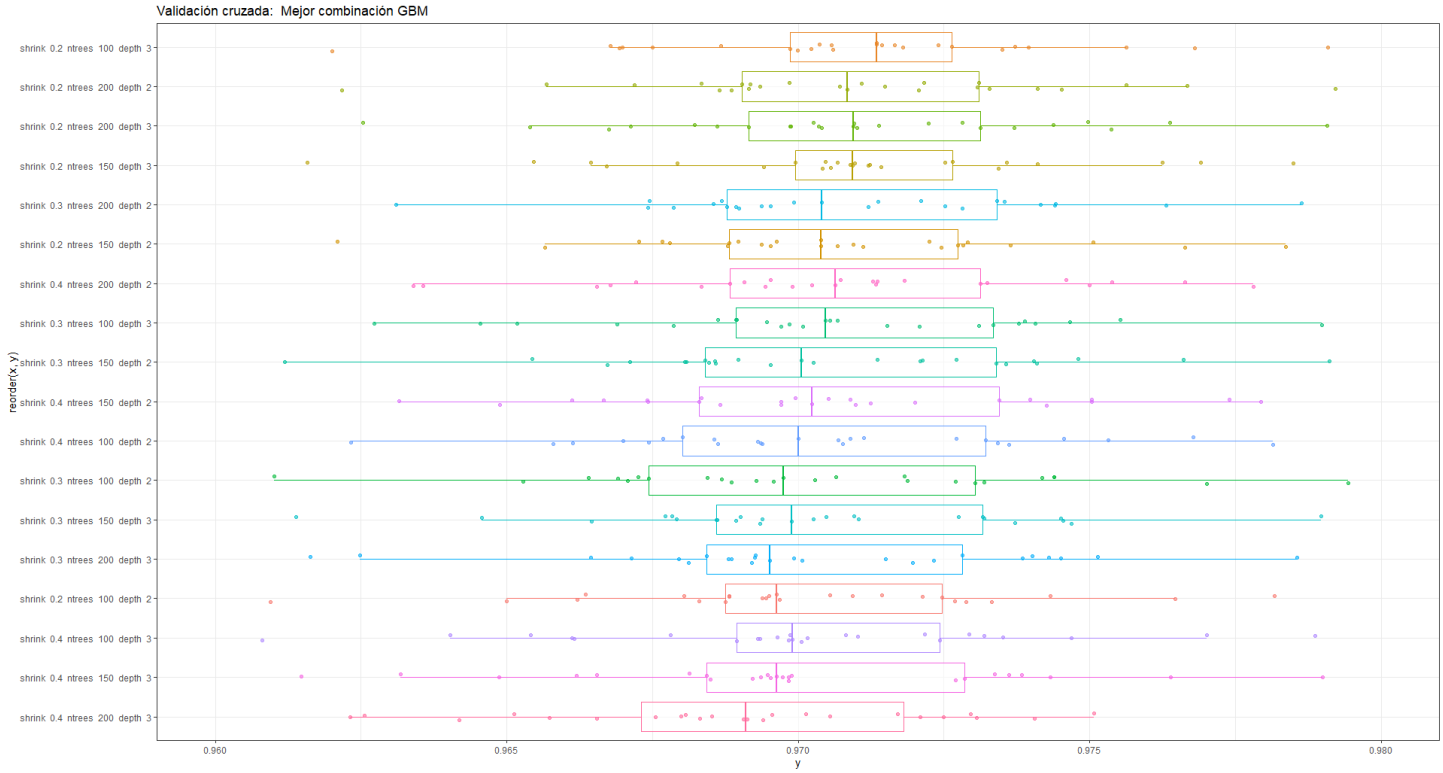


Figura 23: Diagrama de cajas de comparación del AUC frente a las mejores combinaciones de parámetros posibles en modelos de GBM.

A nivel de sesgo, parece que todas las opciones dan resultados muy similares (la mediana del AUC vale en todos los casos en torno a 0,97). La mejor combinación a nivel de varianza parece ser con una constante de regularización de 0,2, un número de iteraciones de 100 y una profundidad de 3.

Se observan diferencias más notorias entre los diferentes modelos a nivel de varianza. En este caso, parece que los mejores modelos son los que poseen una constante de regularización de 0,2 y un número de iteraciones entre 100 y 200. Parece que usar un valor de la constante más elevado produce mayor sobreajuste, disparando la varianza.

Por tanto, a nivel global, el modelo de Gradient Boosting que mejor se comporta (y será finalista en la evaluación) es el que posee 100 iteraciones, un valor de la constante de regularización de 0,2, un número mínimo de observaciones por nodo de 20 y una profundidad de 3.

6.5.2. Extreme Gradient Boosting (XGBoost)

A raíz de la aparición de *Kaggle* y la popularización de la ciencia de datos, en los últimos años han aparecido nuevos modelos muy sofisticados con un poder de predicción y una versatilidad inigualables (Vega, 2020). Aunque recientemente está siendo desbancado por modelos más evolucionados y menos exigentes computacionalmente (como pueden ser *Light GBM* o *Histogram-Based GBM*), el algoritmo de *XGBoost* (*Extreme Gradient Boosting*) sigue siendo muy utilizado y se ha decidido darle un pequeño apartado en este trabajo.

La idea bajo *XGBoost* es, de nuevo, utilizar la técnica de descenso del gradiente para reducir el error de predicción a base de encadenar predicciones con algoritmos débiles (típicamente son árboles de decisión). A grandes rasgos, lo que se hace es ajustar los parámetros del árbol para intentar reducir el valor de la función objetivo. Si el nuevo árbol mejora los resultados del árbol previo, se utiliza como modelo base para seguir mejorando la predicción, en caso contrario se regresa al anterior y se progresa por otra vía.

Para aplicar XGBoost sobre el dataset de estudio se acudirá de nuevo a la librería caret, utilizando el método *xgbTree*. Igual que el resto de algoritmos empleados en caret, se debe utilizar la función train para realizar el entrenamiento, y opcionalmente, se puede definir un grid para la parametrización y usa la función *trainControl* para definir el proceso de entrenamiento a utilizar.

En este caso, se ha creado un grid general para optimizar siete de los principales parámetros usados en XGBoost. Los parámetros incluidos son: *min_child_weight, eta, nrounds, max_depth, gamma, colsample_bytree y subsample.*

```
XGBctrl <- trainControl(method = "repeatedcv",number = 3, repeats =
3,savePredictions = "all", classProbs = TRUE, verboseIter = FALSE,
returnResamp = "all", summaryFunction = twoClassSummary)
XGBgrid <- expand.grid( min_child_weight = c(10, 20, 30) , eta = c(0.001,
0.01, 0.1, 0.2), nrounds = c(100, 200, 300), max_depth = c(6, 8), gamma = c(0,
2, 5), colsample_bytree = c(0.5, 0.7), subsample = 0.8)
Etatrain <- train(Y ~ ., data = train, method="xgbTree", trControl = XGBctrl,
tuneGrid = XGBgrid, verbose = FALSE, alpha = 0,lambda = 0,lambda_bias= 0)
Eta_resample <- as.data.frame(Etatrain$resample)
for (i in 1:7) {
  Eta_resample$concat <- paste(Eta_resample$concat, Eta_resample[,i])
}
Eta_resample$Mean <- ave(Eta_resample$ROC, Eta_resample$concat)
XGB_Results <- head(Eta_resample[order(Eta_resample$Mean),], 250)
model_boxplot(data = XGB_Results, x = XGB_Results$concat, y = XGB_Results$ROC,
title = "mejor combinación XGB", ymin = 0.93, ymax = 0.97)
```

Los resultados se han graficado en forma de diagrama de cajas. Se han construido un total de 432 combinaciones de parámetros distintas, las cuales se han entrenado cada una con validación cruzada repetida con 3 repeticiones y 3 grupos. Eso hace un total de 3.888 modelos de XGBoost diferentes, cada uno con entre 100 y 300 iteraciones en el proceso de convergencia. Eso quiere decir que se han construido entre 388.800 y 1.166.400 árboles.

Queda claro que el proceso de entrenamiento de XGB es largo y conlleva un elevado coste computacional. Además, a diferencia de random forest, en XGBoost, los árboles no tienen unas dimensiones limitadas en la definición del propio algoritmo, sino que crecen hasta su máxima extensión. Para combatir este problema, XGBoost permite trabajar con varios núcleos de forma simultánea. Esto significa que si, por ejemplo, la estación de trabajo posee 16 núcleos, el algoritmo permite paralelizar hasta 16 procesos al mismo tiempo, reduciendo así los tiempos de ejecución.

Aun así, queda claro que utilizar este tipo de algoritmo requiere de un esfuerzo considerable a nivel de hardware. Por este motivo han aparecido algoritmos como LGBM o HGBM, los cuales ofrecen resultados similares o mejores, con un coste computacional más bajo.

Los resultados obtenidos para las 10 mejores combinaciones de XGBoost se muestran en la figura. Observando la gráfica, parece que todos los modelos responden peor que lo visto con los anteriores algoritmos a nivel de sesgo.

Haría falta una tarea de optimización y búsqueda de los mejores parámetros más profunda y sistemática para hallar una combinación que supere estos resultados. Sin embargo, dados los elevados tiempos de cálculo que requiere este algoritmo y el propósito meramente introductorio de este apartado, se dejará este estudio para futuros análisis.

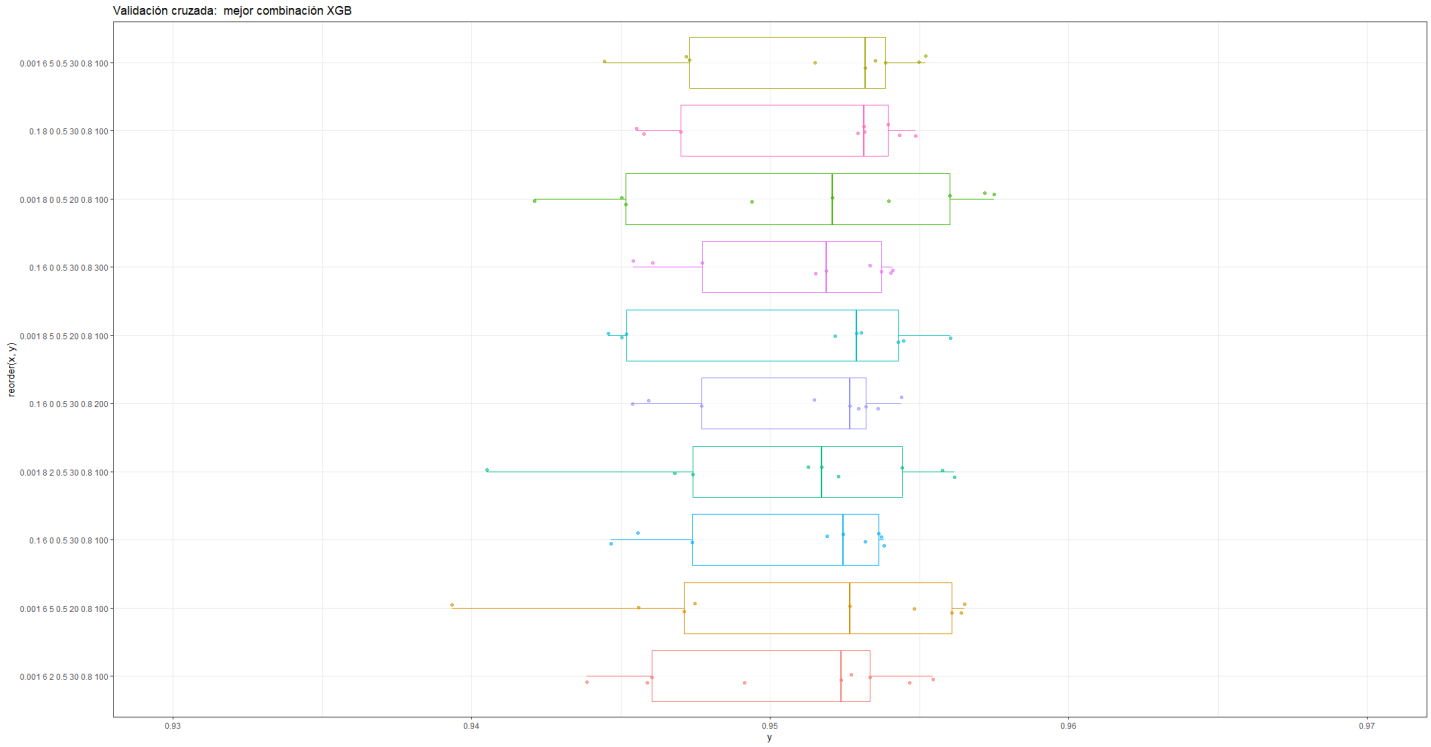


Figura 24: Comparación del AUC entre diferentes combinaciones del modelo XGBoost.

6.6. Support Vector Machines

El último algoritmo a estudiar es *Support Vector Machines* (*SVM*). Este algoritmo se utiliza en clasificación por su gran utilidad gracias a su funcionamiento. Básicamente, se trata de utilizar la idea del "maximal margin", que busca la función (hiperplano) que maximiza las distancias desde el mismo hasta las diferentes clases en las que divide al conjunto de datos (Figura 25).

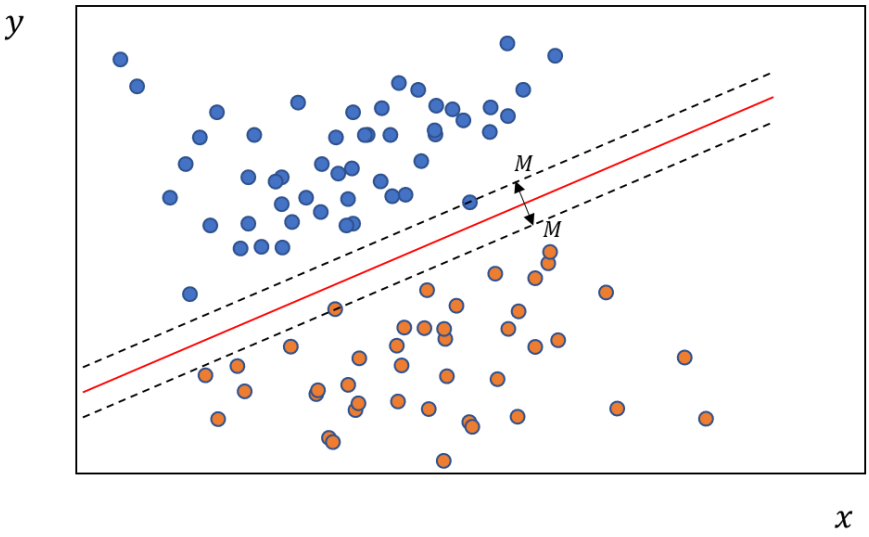


Figura 25: Esquema del "maximal margin" usado en SVM.

No se suele dar ningún caso de clasificación perfecta, sino que lo habitual es que se dé la presencia de observaciones mal clasificadas. El valor del margen admitido repercute en la tendencia al sobreajuste del algoritmo (si se admite un margen muy estrecho, puede que el algoritmo se adapte muy bien a los datos de entrenamiento, pero fallará con datos nuevos).

Para controlar este margen de fallo, se utiliza un parámetro conocido como constante de regularización *C*. A mayor valor de esta constante, menor margen y mayor sobreajuste, mientras que, a menor C, el margen es más amplio.

La otra base sobre la que se sostiene este algoritmo es la idea del *Kernel*. Se trata de un método matemático que sirve para reducir la dimensión del espacio en el que trabaja el algoritmo para simplificar el cálculo. Este proceso se lleva a cabo a través de la aplicación de productos escalares de los vectores formados por las variables input.

Hay diferentes tipos de kernel, y cada uno se adapta mejor a un tipo de estructura en los datos. Los más utilizados son el lineal, el polinómico y el radial (o RBF). Estos dos últimos permiten un grado de parametrización superior, permitiendo personalizar factores como el grado del polinomio a desarrollar, para el caso polinómico; o diferentes constantes de regularización para el caso radial. Por tanto, el otro parámetro a tener en cuenta en la optimización del algoritmo será el tipo de kernel y sus constantes asociadas.

Para aplicar este algoritmo se ha hecho uso del paquete caret, usando como métodos “svmLinear”, “svmPoly” o “svmRadial”, según el tipo de kernel a emplear. Para comenzar, se ha empleado el siguiente código en búsqueda del valor más adecuado de la constante C, probando solamente con el kernel lineal:

```
Ccontrol <- trainControl(method = "repeatedcv",number = 5, repeats
=5,savePredictions = "all", classProbs = TRUE, verboseIter = FALSE,
returnResamp = "all", summaryFunction = twoClassSummary)
Cgrid <- expand.grid(C = c(0.001, 0.01, 0.05, 0.1, 0.2, 0.5))
Ctrain <- train( Y ~ ., data = train,  method="svmLinear", metric = "ROC",
trControl = Ccontrol, tuneGrid = Cgrid, replace = TRUE)
C_resample <- as.data.frame(Ctrain$resample)
model_boxplot(data = C_resample, x = C_resample$C, y = C_resample$ROC, title =
"Valor C SVM", ymin = 0.90, ymax = 0.98)
```

Parece que, a tenor del diagrama resultante (Figura 26), el modelo obtiene los mejores resultados en sesgo y varianza con valores de C comprendidos en un rango entre 0,001 y 0,01. Cabe recalcar que este ensayo solamente es orientativo, puesto que utiliza un único tipo de kernel, pero servirá para decidir en qué rango se encontrará el valor óptimo de esta constante.

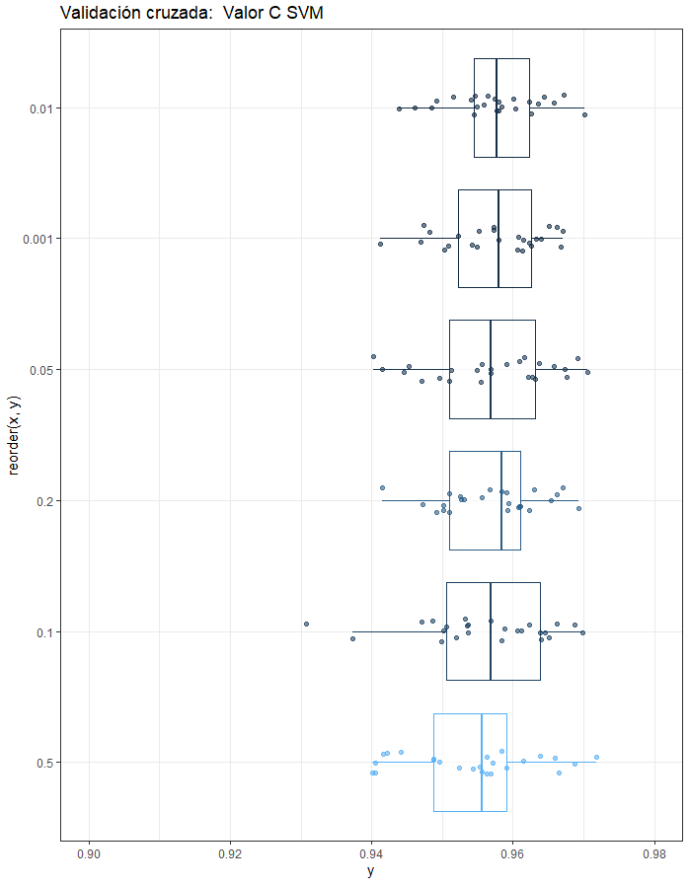


Figura 26: Diagrama de cajas de comparación del AUC frente a la constante C en SVM.

Para determinar qué tipo de kernel es el más adecuado se ha empleado el siguiente bucle:

```
Kernelcontrol <- trainControl(method = "repeatedcv",number = 5, repeats
=5,savePredictions = "all", classProbs = TRUE, verboseIter = FALSE,
returnResamp = "all", summaryFunction = twoClassSummary)
kernels <- c("svmLinear", "svmPoly", "svmRadial")
k_resample <- c()
for (k in kernels) {
  if (k == "svmLinear"){
    Kernelgrid <- expand.grid(C = c(0.001, 0.01, 0.05))
  } else if (k == "svmPoly") {
    Kernelgrid <- expand.grid(C = c(0.001, 0.01, 0.05), degree = 2, scale = 1)
  } else {
    Kernelgrid <- expand.grid(C = c(0.001, 0.01, 0.05), sigma = 1)
  }
  kerneltrain <- train( Y ~ ., data = train,  method = k, metric = "ROC",
trControl = Kernelcontrol, tuneGrid = Kernelgrid, replace = TRUE)
  concat <- paste(k, "C = ", kerneltrain$resample$C)
  k_resample <- rbind(k_resample,
cbind(as.data.frame(kerneltrain$resample[,1:4]), concat))
}
model_boxplot(data = k_resample, x = k_resample$concat, y = k_resample$ROC,
title = "Tipo de Kernel", ymin = 0.90, ymax = 0.98)
```

Los resultados se han representado en el diagrama de cajas de la figura. A nivel de varianza los modelos lineales apuntan a ser los mejores. Sin embargo, en cuanto a sesgo, los modelos radiales se comportan mejor, obteniendo una mediana del AUC en torno a 0,96. En cuanto a los polinómicos, quedan lejos de las otras dos opciones en ambos sentidos, con una varianza de casi cuatro décimas en términos de AUC en el peor de los casos.

Valorando todo en conjunto, se ha decidido seleccionar el kernel radial como la mejor opción, puesto que la pequeña ganancia en varianza del modelo lineal no compensa la pérdida a nivel de sesgo con respecto al caso radial.

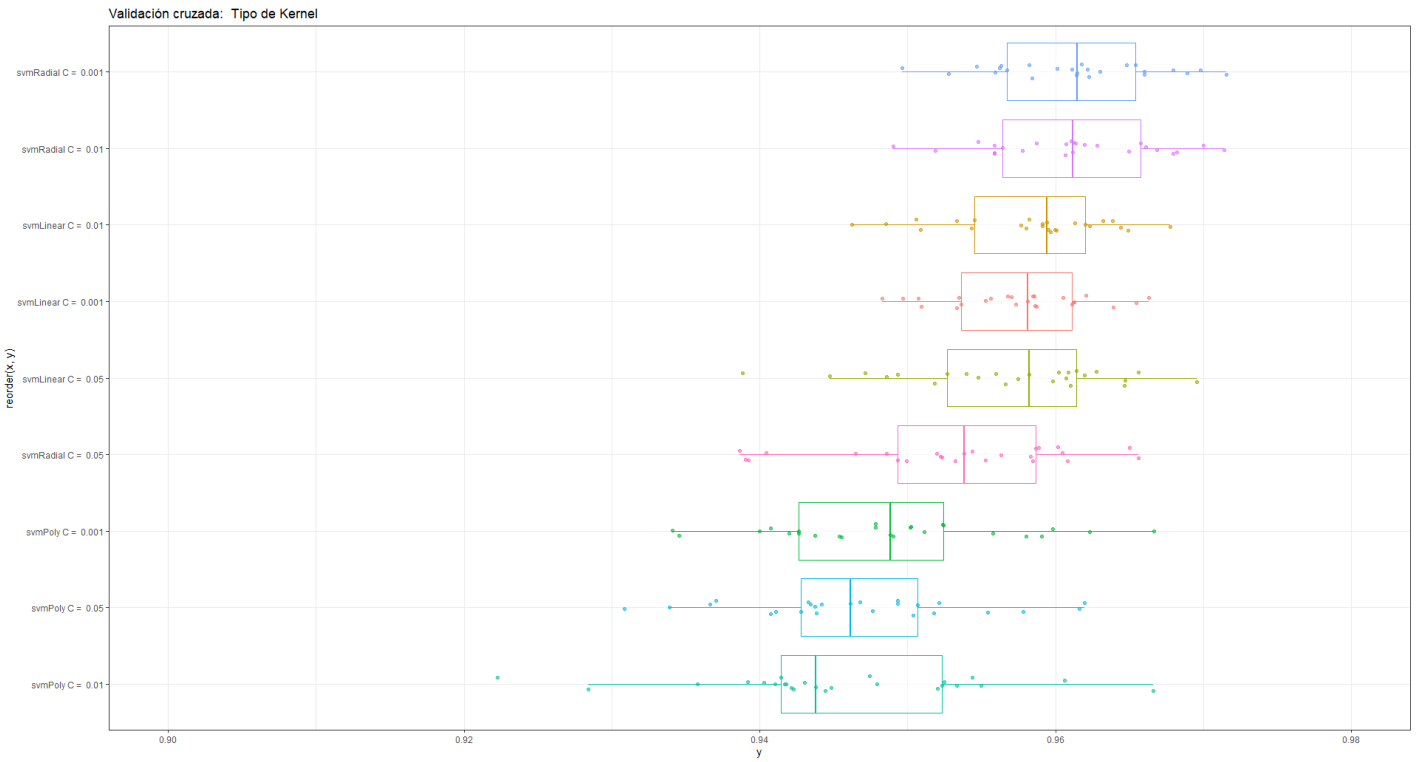


Figura 27: Diagrama de cajas de comparación del AUC frente a los combinaciones con diferentes tipos de kernel en SVM.

Puesto que se ha decidido seleccionar el modelo radial como el óptimo, se va a tunear un hiperparámetro adicional propio de este tipo de kernel: el valor de la constante *sigma*. Esta constante aparece implícita en la función de definición del modelo, y a grandes rasgos, aumentar su valor puede aumentar el sesgo y reducir el sobreajuste, mientras que reducir su valor provoca el efecto opuesto.

En este caso, se ha creado un pequeño script donde se evalúa su valor mediante un grid y se representa el resultado (en AUC) en forma de diagrama de cajas:

```
sigmagrid <- expand.grid(C = 0.001, sigma = c(0.001, 0.1, 0.5, 1, 2))
sigmatrain <- train( Y ~ ., data = train,  method = "svmRadial", metric =
"ROC", trControl = Kernelcontrol, tuneGrid = sigmagrid, replace = TRUE)
sigma_resample <- as.data.frame(sigmatrain$resample)
model_boxplot(data = sigma_resample, x = sigma_resample$sigma, y =
sigma_resample$ROC, title = "Valor de sigma", ymin = 0.93, ymax = 0.97)
```

Los resultados se muestran en la Figura 28. A la vista de los mismos, parece que utilizar valores elevados de sigma mejora notablemente el rendimiento del algoritmo tanto en sesgo como en varianza. Y así se vuelve a corroborar que, con este dataset, apenas se producen problemas de sobreajuste, por lo que las opciones más agresivas parecen ser las más efectivas.

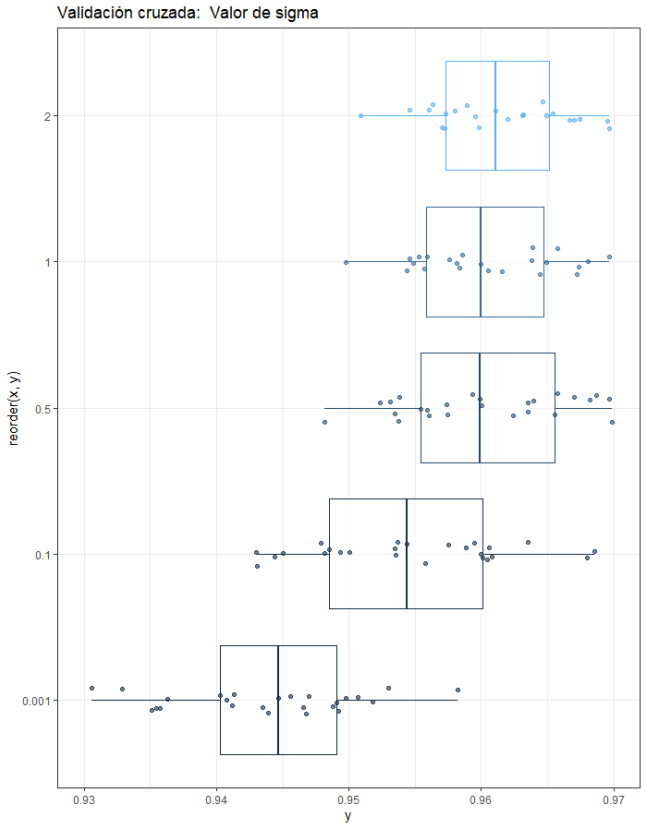


Figura 28: Diagrama de cajas del AUC frente a los efectos del valor de sigma en SVM radial.

Por tanto, se ha construido el modelo el modelo final resultante, que será utilizado en la fase de evaluación. **Se trata de un modelo de SVM radial, con un valor de la constante C de 0,001 y un valor de sigma de 2.**

7. Evaluación de modelos y construcción de un modelo de Stacking

7.1. Selección del mejor modelo

Una vez construidos los seis modelos finalistas (uno por cada tipo de algoritmo quitando el árbol de decisión, aunque XGBoost parte con desventaja al haberse introducido muy someramente) hay que pasar a la parte de evaluación.

Esta parte del proceso es fundamental, pues es aquí donde se pone en valor todo el trabajo de refinamiento y tuneado realizado. Además, **en un entorno empresarial, es en esta etapa donde se tiene que tener muy claro el objetivo final de los modelos, en términos de negocio.**

Dicho de otra forma, no todo se reduce a obtener el mejor valor del AUC, la tasa de clasificación o el R2. **En la etapa de evaluación es muy importante escoger una métrica acorde al problema planteado**, que sea capaz de relacionar el poder de predicción del modelo con un indicador de productividad, una medida económica, una relación de ahorro en horas de trabajo o cualquier tipo de cifra que pueda traducir el lenguaje estadístico en lenguaje empresarial.

También es aquí donde se valida el esfuerzo realizado. Se plantean posibles mejoras a llevar a cabo, nuevas ideas para optimizar el proceso y se realiza un proceso cíclico en el que los modelos se van ajustando hasta dar con la fórmula ganadora.

En este caso, dado que nos encontramos en un entorno meramente académico, nos ceñiremos a la evaluación de los modelos desde el punto de vista estadístico-matemático.

Para ello, como viene siendo habitual a lo largo de todo el trabajo, se va a hacer uso de los diagramas de las cajas y la representación del AUC en el conjunto de datos de prueba.

Para ello, se han entrenado los cinco algoritmos, configurados con la mejor combinación de hiperparámetros estudiados a lo largo de todo el trabajo. Se ha hecho uso de validación cruzada repetida con cinco grupos y cinco iteraciones cada uno (en total, 25 observaciones para cada modelo). Los resultados se muestran en la Figura 29.

A la vista queda, que hay un modelo claramente vencedor en términos de sesgo, y no es otro que el modelo de gradient boosting. Este modelo consigue una mediana para el AUC ligeramente superior a 0,97 y la varianza más baja de todas. Muy cerca se encuentra la red neuronal, seguida del modelo de regresión logística.

Hay dos modelos que obtienen un resultado inferior a los anteriores tanto en varianza como en sesgo, siendo estos random forest y SVM.

Y, por último, a nivel de sesgo, se encuentra el modelo de XGBoost. Se reitera que este modelo no ha sido tuneado tan en detalle como el resto. Se dejará para futuros desarrollos su implementación más en profundidad.

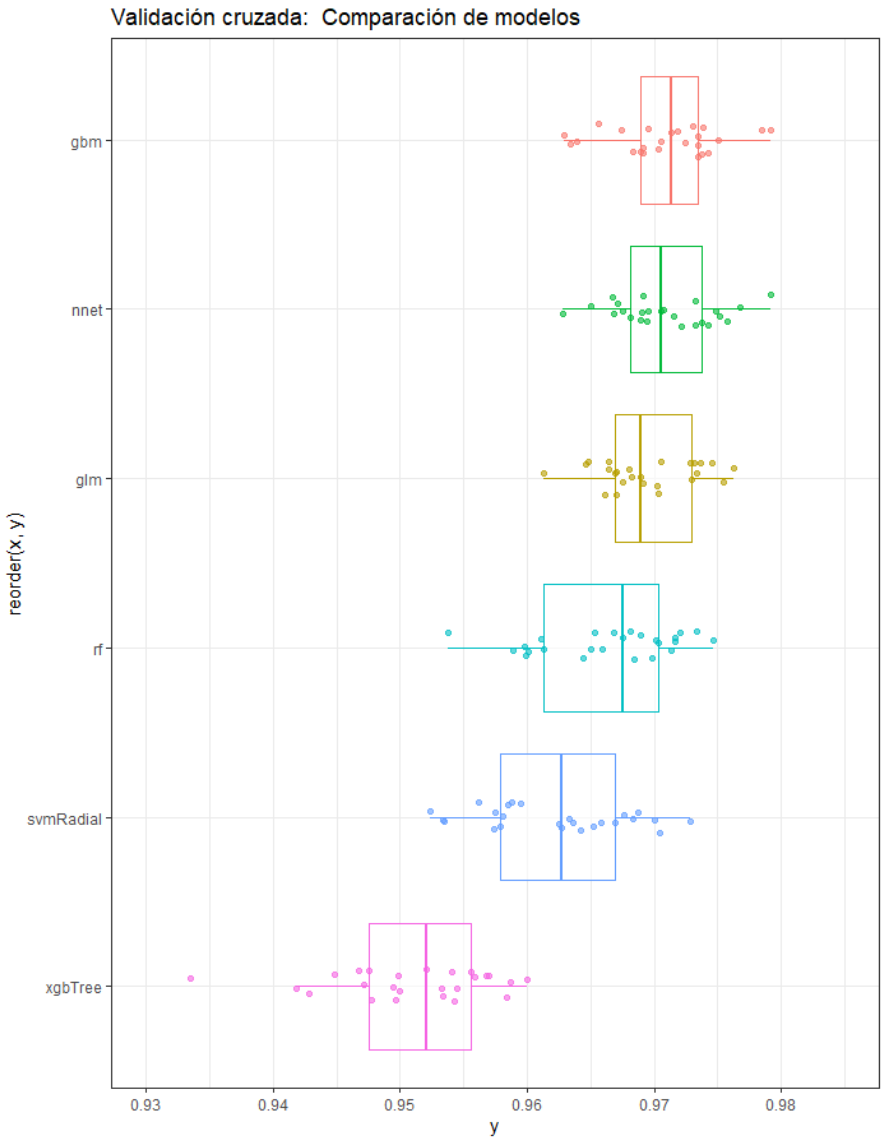


Figura 29: Diagrama de cajas de comparación final entre los mejores modelos construidos con cada tipo de algoritmo.

Por tanto, en términos globales, el mejor modelo de los seis ha resultado ser el modelo de gradient boosting. Este modelo consigue alcanzar un valor del AUC en los datos de prueba muy elevado y con una varianza relativamente baja. Es sorprendente que, a pesar de haber elegido este modelo como vencedor, todos los modelos entrenados dan lugar a valores del AUC muy elevados, dando a entender que el conjunto de datos presenta un comportamiento y variabilidad muy marcadas y fáciles de identificar por cada algoritmo, independientemente de estar enfocados a casos lineales, al uso de árboles de decisión etc.

Cuando ocurre algo así en el entorno empresarial, lo realmente complicado no es encontrar el mejor modelo, sino afinarlo y sacarle el máximo partido posible. Y es que este tipo de situaciones, en la que las predicciones son tan buenas, suelen

corresponder con casos de negocio muy estudiados, en los que apenas se pueden mejorar u optimizar los procesos productivos. La verdadera dificultad, por tanto, se presenta en encontrar esos pequeños matices que pueden mejorar ligeramente el modelo, como encontrar alguna nueva variable que pueda aportar algo más de luz.

7.2. Construcción de un modelo de Stacking

A lo largo de todo el trabajo se han utilizado algunos algoritmos cuya idea fundamental se basa en las técnicas de ensamblado. Como ya se ha mencionado, estas técnicas toman las predicciones de algoritmos sencillos (como los árboles de decisión), las alteran de una determinada forma y después generan una predicción más compleja, que normalmente mejora en cierta medida los resultados, especialmente a nivel de varianza.

Estas mismas técnicas pueden ser utilizadas para generar un modelo de ensamblado en base a los algoritmos complejos construidos a lo largo de todo el trabajo. Hay muchos algoritmos de ensamblado diferentes. Por ejemplo, *Ensembled Voting* se basa (en el problema de clasificación) en ir seleccionando el algoritmo que mejor precisión obtiene en cada región del conjunto poblacional.

En la Figura 30 se muestra un pequeño esquema del comportamiento de este algoritmo. En él se puede ver cómo, en la primera región, el modelo 1 (azul) es el que mejor responde, y por eso sus predicciones son las seleccionadas en el modelo final. En la segunda región el mejor es el modelo 2 (amarillo) y en la tercera es el modelo 3 (verde).

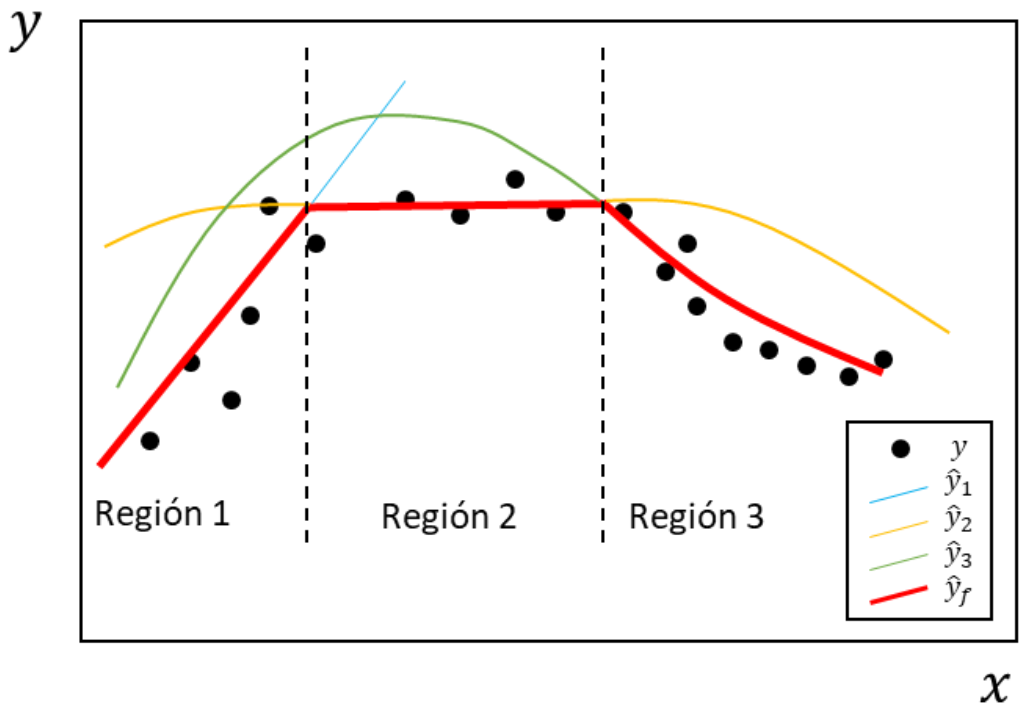


Figura 30: Esquema de funcionamiento de Ensembled Voting

Otro algoritmo de ensamblado muy utilizado es el llamado *Stacking Ensemble*. La idea fundamental detrás de este algoritmo es utilizar un estimador final que procese las predicciones de los estimadores intermedios. Ese estimador final puede ser cualquier modelo de machine learning de los que se han estudiado en este trabajo, o cualquier otro que se nos ocurra. Los estimadores intermedios (*mid-estimators*) también pueden ser cualquier otro algoritmo conocido.

El funcionamiento es el siguiente: cada estimador intermedio entrena y realiza una predicción sobre el conjunto de entrenamiento. Esas predicciones se guardan, y son almacenadas. Después, el estimador final realiza el proceso de entrenamiento en base al dataset formado por las predicciones provenientes de los anteriores estimadores. Tras ese entrenamiento, se produce una predicción final. En la Figura 31 se muestra un esquema del proceso.

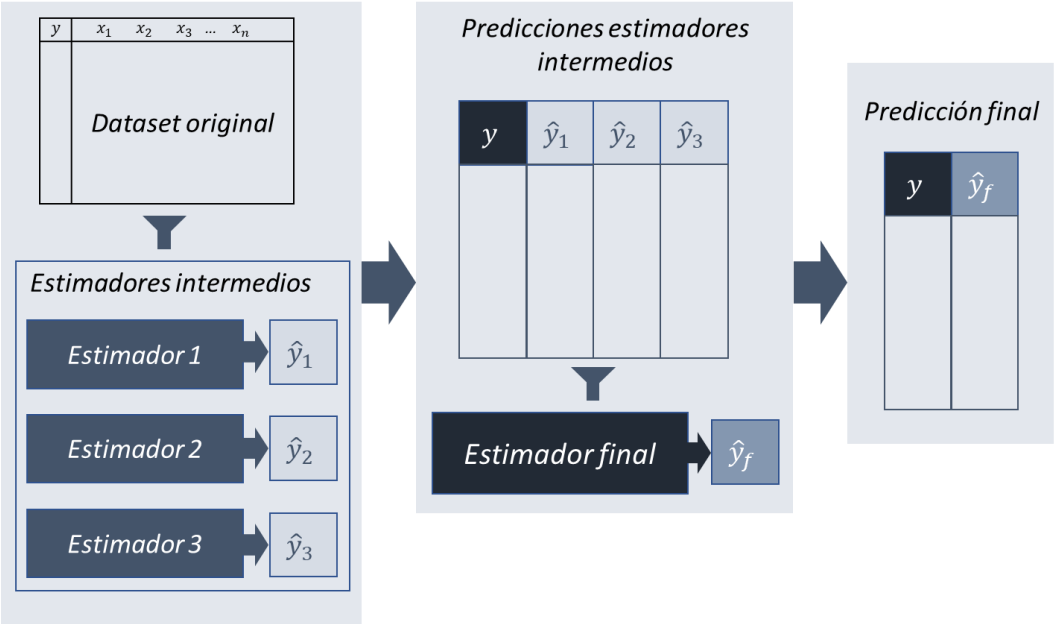


Figura 31: Esquema de funcionamiento de los modelos de stacking.

Una vez presentados algunos de los principales modelos de ensamblado, se va a llevar a cabo la implementación de un modelo de stacking en R, en base a los resultados obtenidos con los mejores modelos construidos hasta ahora (Bhalla, s. f.).

Para llevar a cabo este experimento, se han tomado los tres mejores modelos construidos hasta esta fecha, siendo estos GBM, redes neuronales y regresión logística. Se trata de un conjunto de modelos bastante equilibrado, pues tiene representación un modelo lineal (la regresión), un modelo basado en árboles (GBM) y un modelo a priori no lineal (la red neuronal).

Uno de los factores fundamentales para implementar un modelo de stacking es determinar qué estimador final es el más apropiado para el problema planteado. En teoría, los modelos más sencillos suelen ser los más efectivos, puesto que el dataset

del que parten es muy reducido (se recuerda que solamente se dispone de las predicciones realizadas por los estimadores intermedios). Por tanto, no hay pie a buscar grandes patrones ocultos en dicho dataset, sino que más bien lo que se busca es combinar las predicciones de manera óptima, intentando subsanar los defectos de unos modelos con las bondades de los otros.

Para contrastar dicha hipótesis de sencillez en el estimador final, se van a probar tres estimadores finales: una regresión lineal simple, un modelo de random forest y un GBM.

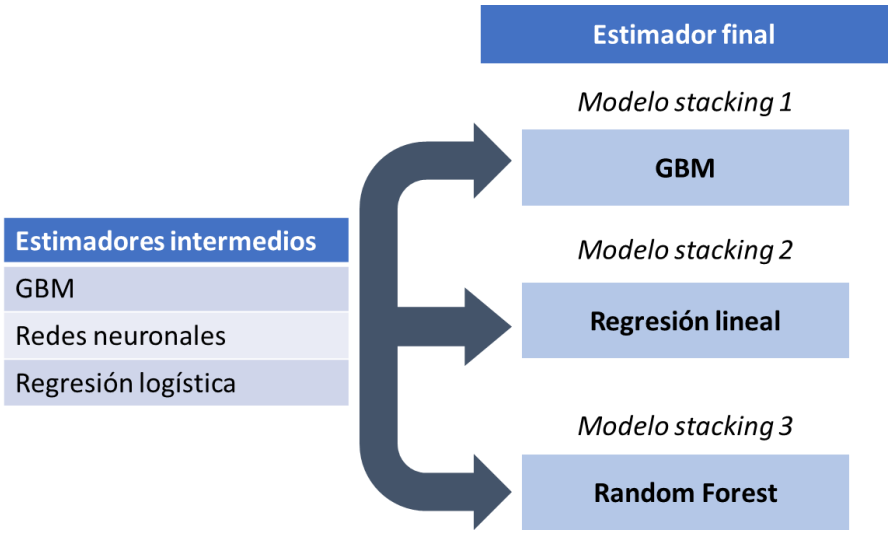


Figura 32: Resumen de la estructura de los modelos de stacking desarrollados.

Existen diversas librerías para implantar modelos de stacking en R. No obstante, se ha optado por desarrollarlos de forma manual, paso por paso (Pipis, 2020). El código empleado es el siguiente:

```
ctrl <- trainControl(method = "repeatedcv",number = 5, repeats
=5,savePredictions = "all", classProbs = TRUE, verboseIter = FALSE,
returnResamp = "all", summaryFunction = twoClassSummary)
REG <- train(as.formula(modeloStepBIC), data = train, method = "glm",
family="binomial", metric = "ROC", trControl = ctrl)
RED <- train(Y ~ ., data = train, method = "nnet", metric = "ROC", maxit = 25,
trControl = ctrl, tuneGrid = REDgrid)
GBM <- train(Y ~ ., data = train, method = "gbm", metric = "ROC", trControl =
ctrl, tuneGrid = GBMgrid, distribution = "bernoulli", verbose = FALSE)
est_results <- resamples(list(REG, RED, GBM))
modelCor(est_results)
test$REG <- predict(REG, test, type='prob')$Yes
roc(test$Y , test$REG, direction = "<")$auc
test$RED <- predict(RED, test, type='prob')$Yes
roc(test$Y , test$RED, direction = "<")$auc
test$GBM <- predict(GBM, test, type='prob')$Yes
roc(test$Y , test$GBM, direction = "<")$auc
length(RED$pred$Yes)

train$REG <- predict(REG, train, type='prob')$Yes
train$RED <- predict(RED, train, type='prob')$Yes
train$GBM <- predict(GBM, train, type='prob')$Yes


model_stacking_gbm <- train(train[,c('RED','REG','GBM')],
```

```
                    train[,"Y"],method='gbm',trControl=ctrl,
                    tuneLength=1)

model_stacking_lr <- train(train[,c('RED','REG','GBM')],
                           train[,"Y"],method='glm',trControl=ctrl,
                           tuneLength=1)

model_stacking_rf <- train(train[,c('RED','REG','GBM')],
                           train[,"Y"],method='rf',trControl=ctrl,
                           tuneLength=1)

Final_CV <- rbind(ROC_CV, cbind("stacking-gbm",
model_stacking_gbm$resample$ROC), cbind("stacking-lr",
model_stacking_lr$resample$ROC), cbind("stacking-rf",
model_stacking_rf$resample$ROC))
summary(Final_CV)
Final_CV$V2 <- as.numeric(Final_CV$V2)
model_boxplot(data = Final_CV, x = Final_CV$V1, y = Final_CV$V2, title =
"Comparación AUC todos los modelos frente a stacking", ymin = 0.93, ymax =
0.98)
```

Este código reproduce el funcionamiento de los modelos de stacking paso por paso y de forma básica. Finalmente representa los resultados en forma de diagrama de cajas, comparando dichos resultados con los que se obtuvieron con el resto de algoritmos en el apartado previo. El resultado tras ejecutar dicho código se muestra en la Figura 33.

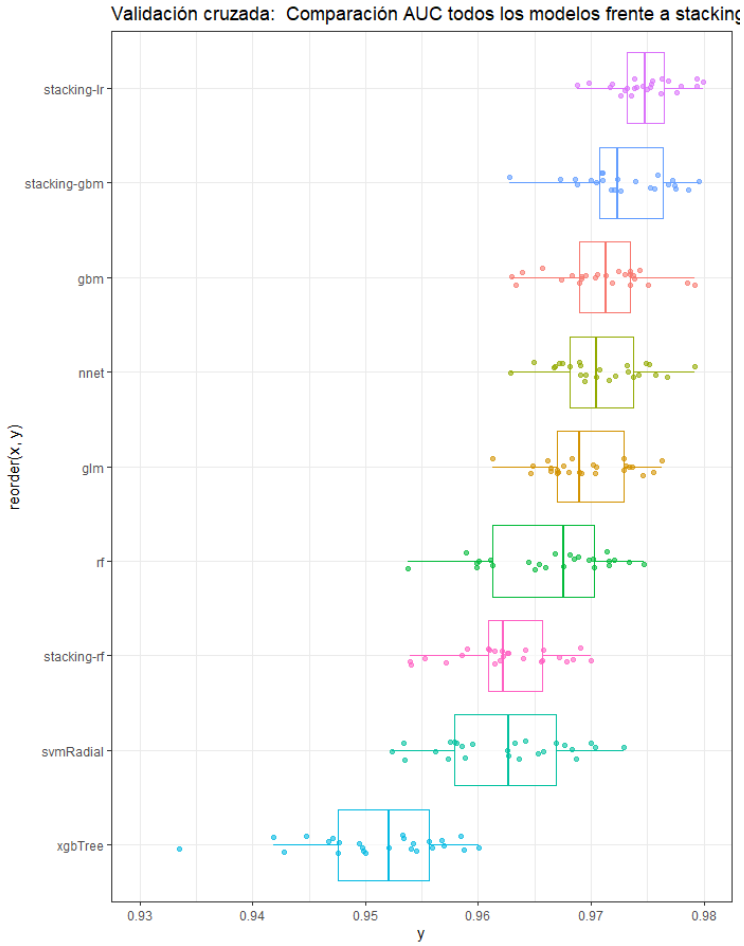


Figura 33: Comparación del AUC entre los mejores modelos desarrollados y los tres modelos de stacking construidos en base a estos.

Como se puede observar, el ensayo se ha realizado bajo las mismas circunstancias que en el resto de modelos (mismo sistema de entrenamiento y validación cruzada).

Fijándonos en el sesgo, se puede ver que el mejor modelo de todos resulta ser el stacking con regresión lineal como estimador final, seguido del stacking con GBM. El otro modelo de stacking desarrollado, que contiene un random forest como estimador final, ha obtenido resultados decepcionantes a nivel de sesgo.

En cuanto a la varianza, todos los modelos de stacking han conseguido buenos resultados, pero destaca el modelo con regresión lineal, que ha logrado la mejor varianza hasta la fecha.

Pero antes de cantar victoria, falta comprobar la bondad del mejor modelo de stacking frente al dataset de prueba. Se va a realizar dicha evaluación tanto en términos de ROC como de la propia matriz de confusión. Para obtener el AUC se ha utilizado este código:

```
Final_prediction <- predict(model_stacking_lr, test[, c('RED','REG','GBM')],
type='prob' )$Yes
roc(test$Y , Final_prediction, direction = "<")$auc
```

Aquí cabe señalar que se está utilizando como dataset el conjunto formado por las predicciones realizadas con los tres estimadores intermedios, las cuales se calcularon en las anteriores líneas de código.

El resultado final del AUC ha resultado ser 0,9736. En cuanto a la matriz de confusión, el resultado obtenido es el siguiente:

	Reference		
Prediction	No	Yes	
No	1642	132	1774
Yes	61	936	997
	1703	1068	2771

Tabla 10: Matriz de confusión obtenida con el mejor modelo de stacking.

Cabe mencionar que esta matriz corresponde a un punto de corte de 0,5. La precisión obtenida es del 93%, mientras que la sensibilidad es de un 96% y la especificidad es de un 87%. Por tanto, parece que este modelo es mejor detectando a los contactos que no realizarán conversión frente a los que sí.

A pesar de que sería interesante realizar un estudio del punto de corte óptimo para el problema de clasificación binaria, lo cierto es que en modelos de lead scoring este estudio carece de sentido. El objetivo de estos modelos no es lograr una clasificación entre dos clases, sino más bien determinar las temperaturas de cada contacto o extraer influencia de las variables más importantes. Por este motivo, se va a prescindir del estudio del punto de corte.

Observando los resultados, queda demostrado que el modelo de stacking funciona correctamente. A nivel de AUC se observa que este tipo de modelos mejoran los

resultados conseguidos con el resto de algoritmos. En concreto, el que usa la regresión lineal como estimador final ha logrado resultados brillantes en todos los sentidos, siendo este el candidato, en un hipotético caso real, a ser el algoritmo puesto en producción como modelo de lead scoring.

Como se ha dicho párrafos atrás, otro punto a tener en cuenta para evaluar un modelo es la métrica utilizada. Hasta ahora se ha utilizado siempre el AUC como métrica de evaluación y comparación, puesto que permite ver con un único número cómo de bien se comporta el modelo con cualquier punto de corte. Se trata de una forma eficaz de comparar diferentes algoritmos desde un punto de vista estadístico. Sin embargo, esta métrica deja de tener valor cuando se quiere medir el valor real del modelo en términos productivos, o comprobar los beneficios de un nuevo desarrollo en comparación con el caso previo.

Para determinar la métrica más adecuada, es muy importante tener claro cuál es el objetivo final del modelo desarrollado en términos operativos. Dicho de otro modo, dependiendo de la finalidad que queramos dar a nuestro modelo, la salida que debe arrojar puede ser de diferente naturaleza. Por ejemplo, si el modelo tuviera como finalidad establecer la "temperatura" de cada contacto para focalizar las estrategias de marketing en un segmento concreto, quizá la salida a tener en cuenta no se trate de las predicciones en sí mismas (medidas en forma de probabilidades), sino el percentil al que pertenece la predicción de cada contacto; el valor medio de las predicciones dentro de un mismo percentil, la segregación en diferentes grupos etc. Al final todo esto se traduce en un KPI que mide la efectividad real del modelo.

Además, la evaluación siempre debe hacerse contra datos que no hayan participado en el proceso de entrenamiento. En un caso real, lo más adecuado sería evaluar este tipo de métricas con un dataset de evaluación, que contenga datos nuevos a ser predichos. Si se trata de un nuevo desarrollo, lo normal es que se traten de datos pertenecientes a un mes anterior, comparando los resultados del nuevo modelo con los que se obtuvieron con el modelo anterior. Sin embargo, dado que no se dispone de más datos aparte del dataset de estudio, la evaluación se ha realizado sobre el dataset de prueba.

Por ejemplo, sería interesante segmentar los contactos del dataset de prueba según su probabilidad de conversión (o temperatura) en *hot leads* y *cold leads*. Una posible forma de hacerlo sería definir en primer lugar a qué nos referimos con estos términos y después, etiquetar a cada individuo según su pertenencia a uno de estos grupos.

Para determinar qué se considerarían *hot leads* podrían utilizarse diferentes estrategias:

- Escoger el 10% de contactos con la probabilidad más elevada de conversión.

- Elegir un punto de corte (por ejemplo, 0,9). Todos los contactos que obtengan una puntuación por encima de 0,9 se considerarán hot leads, mientras que los que queden por debajo no estarán incluidos en este grupo.

- Escoger los X mejores contactos (por ejemplo, 500).

Cada estrategia tiene sus ventajas e inconvenientes, siendo esta una decisión que recae más en la parte operativa de la empresa que en la técnica.

A modo de ejemplo, se ha implementado la primera de las estrategias mediante el siguiente código:

```
cutting_points <- c(0.0, 0.1, 0.3, 0.7, 0.9, 1.0)
h <- c()
predictions <- Final_prediction[order(Final_prediction$Yes),]
for (i in c(1:(length(cutting_points) - 1))) {
  l <- ifelse(i == 1, 1, as.integer(cutting_points[i]*nrow(test)))
  print(l)
  r <- as.integer(cutting_points[i + 1]*nrow(test))
  print(r)
  h <- rbind(h, cbind(paste(toString(cutting_points[i]*100), "% - ",
toString(cutting_points[i + 1]*100), "%"),mean(predictions[l:r, 2])))
}
h <- as.data.frame(h)
h$data_proportion <- as.factor(h$V1)
h$predicted_target <- as.numeric(h$V2)
p <- ggplot(data = h, aes(x = data_proportion, y = predicted_target)) +
  geom_bar(stat="identity") +
  geom_text(aes(label = predicted_target), vjust=1.6, color="white",
            position = position_dodge(0.9), size=3.5)
```

Lo que pretende realizar este código es lo siguiente: se han dividido las predicciones del modelo de stacking en cinco grupos, marcados por seis puntos de corte. Para ello, se han ordenado de forma ascendente las predicciones, y se dividido el conjunto en varios grupos, conteniendo cada uno cierto % de clientes: de 0-10%, de 10-30%, de 30-70%, de 70-90% y de 90-100%. Dentro de cada grupo se ha calculado el valor medio de la predicción en dicho intervalo.

Grupo	Población
Bottom	0 – 10%
Medium-Bottom	10 – 30%
Medium	30 – 70%
Medium-Top	70 – 90%
Top	90 – 100%

En otras palabras, lo que se ha intentado es generar una agrupación de clientes según su probabilidad de conversión, lo que vendría siendo un medidor de temperatura. En teoría, debería verse una progresión gradual desde el grupo más “frío” hasta el más “caliente”. Además, si los grupos están bien formados, el último grupo debería obtener una media claramente más elevada que el resto, puesto que contiene al 10% de contactos con mejor pronóstico de conversión. Son los individuos de este grupo los únicos que se considerarán como hot leads.

Los resultados se muestran en la Figura 34. Se observa que esta estrategia no parece ser demasiado efectiva. Se puede ver que el segundo grupo (*Medium-Top*) es casi tan bueno como el grupo *Top*. Si se lanzase una campaña o acción únicamente contra los

leads calientes del grupo *Top*, habría una gran cantidad de contactos con buen pronóstico de conversión que quedarían fuera.

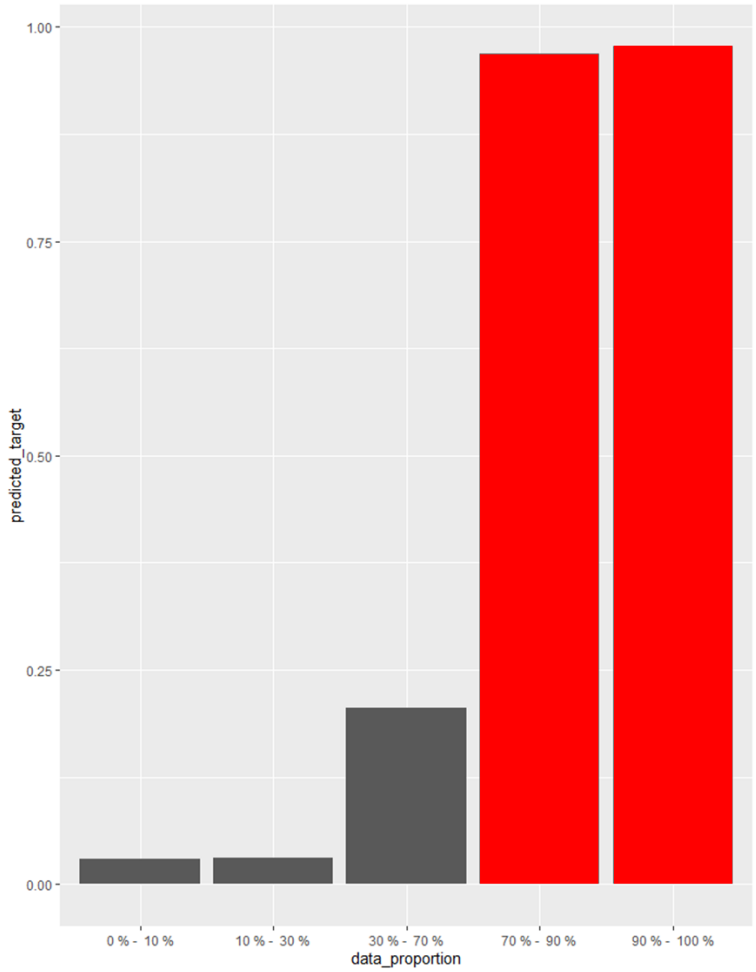


Figura 34: Agrupación de contactos por probabilidad media de cada grupo.

Por tanto, parece que hay que ser un poco más laxos a la hora de determinar el punto de corte para los leads calientes.

Si se quiere seguir otra estrategia para el grupo de los cold leads (*Bottom*), como, por ejemplo, evitar destinar campañas de captación a estos individuos, se observa el mismo problema. Y es que el grupo Medium-Bottom resulta tener prácticamente igual de mal pronóstico de conversión.

Está claro que, entonces, estos cinco grupos no sirven para lograr el objetivo marcado. Quizá tres grupos sean suficientes para obtener una diferenciación más útil desde el punto de vista del departamento de marketing.

Realizando la agrupación en tres grupos (con cuatro puntos de corte), los resultados mejoran sensiblemente (Figura 35).

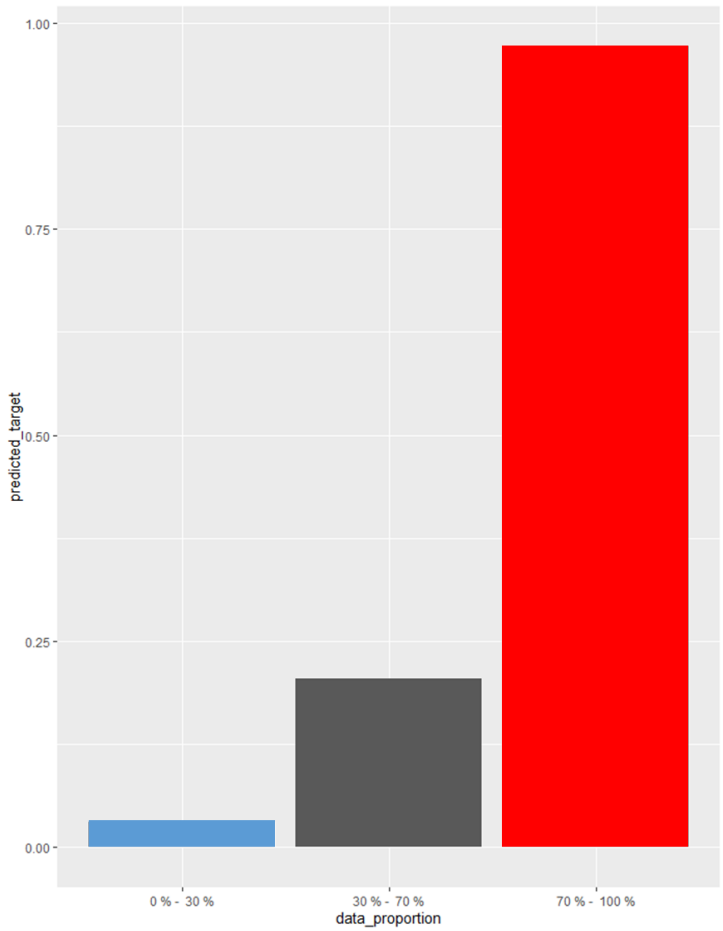


Figura 35: Agrupación de contactos por probabilidad media de cada grupo (tres grupos).

Con estos resultados parece que la diferenciación es mucho más clara. Sería posible seguir ajustando el punto de corte o probar con otras estrategias de segmentación. No obstante, dada la extensión del trabajo se considera suficiente la solución propuesta.

8. Diferencias con la solución propuesta en Kaggle

A pesar de partir del mismo dataset y de intentar construir un modelo con características similares a la solución propuesta en *Kaggle*, hay una serie de diferencias entre ambos desarrollos:

- En primer lugar, la diferencia más obvia es que el esquema seguido en Kaggle utiliza en su totalidad el lenguaje de programación Python (con los paquetes *pandas, numpy y scikitlearn*). Esto incluye tanto la fase de exploración y depuración como la fase de modelización. En el presente trabajo la depuración se ha realizado en su mayoría en SAS Miner, mientras que la modelización se ha llevado a cabo en R Studio.

- En cuanto a la fase de exploración, decir que la propuesta de Kaggle es muy completa, realizando un estudio de la volumetría de cada variable, midiendo con tablas de contingencias y graficando las diferencias entre las dos clases de la variable objetivo para todas y cada una de las variables. En base a estos resultados el autor hace una inferencia con respecto a su influencia sobre el objetivo. En ese sentido, el presente trabajo no ha sido tan ambicioso y se apostado por ahondar más en el detalle de la fase de modelización.

- También el proceso de selección de variables es completamente diferente. Mientras que el presente trabajo apuesta por reducir a la mínima expresión la cantidad de variables a través de la búsqueda de semejanzas mediante dos criterios distintos como pueden ser random forest y las regresiones; el autor del ejercicio de Kaggle opta por usar únicamente el método de random forest como proceso de selección. El autor restringe el dataset a las 15 mejores variables input obtenidas con este método, mientras que en el presente trabajo se han obtenido 13. La mayoría de ellas son comunes, pero hay ligeras diferencias.

- Otra gran diferencia radica en que la solución de Kaggle se centra completamente en la regresión logística, profundizando en los resultados obtenidos con este modelo. Por otro lado, en este trabajo se ha ahondado en múltiples algoritmos, buscando las diferencias entre los resultados obtenidos con cada uno e intentando determinar la influencia de cada hiperparámetro. También se puede destacar como gran aporte de este trabajo la construcción de un modelo de ensamblado como puede ser el stacking.

- La evaluación del modelo propuesto en Kaggle se centra en la matriz de confusión y todas sus métricas (precisión, recall, especificidad, sensibilidad etc). En ese sentido, el autor consigue una precisión de 0,91, frente al 0,93 obtenido con el mejor modelo desarrollado en este trabajo. Esto era de esperar, puesto que, a pesar de contar con menos variables, el modelo de stacking contiene

mucha más capacidad de predicción que una regresión simple, al estar formado por modelos capaces de recrear las no linealidades.

- Con respecto a la evaluación, también cabe comentar que el ejercicio de Kaggle se centra en el estudio del punto de corte para hallar el mejor sistema de clasificación. Ese ejercicio se ha omitido en el presente trabajo. Como se ha comentado, en un problema de lead scoring no es tan importante la propia clasificación como la agrupación de individuos por medio de las probabilidades obtenidas en la predicción. Por esto se destaca como aporte de cierto valor el haber hecho el ejercicio (aunque sea superficialmente) de intentar generar una segmentación de los contactos en base a las predicciones generadas por el modelo.

9. Conclusiones

Una vez seleccionado y evaluado el mejor modelo de lead scoring, es el momento de extraer conclusiones sobre los resultados obtenidos y el proceso llevado a cabo:

- En primer lugar, dados los resultados obtenidos con la totalidad de los algoritmos, se pueden sacar dos conclusiones:

    - El dataset estaba muy preparado para la predicción, presentando un comportamiento muy fácil de reproducir por la mayoría de los algoritmos.

    - También es importante señalar que el proceso de depuración llevado a cabo ha sido exitoso. Se demuestra que, todos los puntos donde había ciertas dudas con respecto a la calidad del proceso realizado, como puede ser el agrupamiento de categorías, la eliminación de variables o el propio sistema de selección han terminado siendo un acierto, puesto que las predicciones obtenidas han sido excelentes.

- También se ha profundizado en los algoritmos de machine learning, habiendo estudiado sus efectos, ventajas, desventajas, parámetros principales y se ha intentado entender el por qué de la calidad de las predicciones obtenidas. A pesar de ello, hay varios algoritmos donde se podría haber profundizado algo más, como se comentará en el siguiente capítulo.

- En general, se puede considerar que el conjunto de datos presenta un comportamiento lineal. Y es que la regresión logística se ha colocado en una sólida tercera posición en el ranking de modelos, muy cerca de los dos primeros (redes neuronales y GBM).

- Aun así, debe existir cierta no linealidad con algunas variables, lo cual podría explicar porque GBM y redes neuronales han rendido tan bien.

- En base a la clasificación realizada en el apartado previo, parece que el dataset tiene también un comportamiento muy simétrico, con una tendencia muy marcada. Hay muchos individuos muy buenos, muchos malos y muy pocos casos intermedios.

- Los modelos de stacking han resultado ser un éxito, consiguiendo resultados sobresalientes en términos de sesgo y varianza.

- Con respecto a los modelos de stacking, también se ha demostrado que un modelo sencillo como la regresión final funciona mejor que modelos más complejos cuando se usa como estimador final.

- El proceso de selección de variables, aunque exitoso, quizá haya sido demasiado agresivo. La ausencia de sobreajuste con la mayoría de modelos, incluso tras haber forzado algunos parámetros, parece indicar que el dataset podría haber funcionado todavía algo mejor de haber incluido más variables.

- En general, la percepción tras haber realizado todo el proceso de depuración y modelización, es que el dataset es poco realista. En un entorno real no es habitual encontrarse con predicciones tan buenas, o no al menos de forma tan rápida. Dar con variables realmente explicativas es un proceso complicado y que requiere de mucho trabajo previo y mucho conocimiento del sector y los patrones de comportamiento de los clientes.

- Tampoco es realista la proporción entre ambas clases de la variable objetivo. En entornos reales la tasa de conversión de este tipo de negocios es muy baja (se está hablando de que habitualmente es inferior al 1%). Esto ha impedido poder enfrentarse al problema más frecuente en esta área, que es el del dataset desbalanceado.

- El modelo de XGBoost tiene mucho potencial y no se ha explorado mucho en sus opciones. Habiendo sido GBM el modelo ganador, era de esperar que un XGBoost bien construido hubiese superado esos resultados, pues en definitiva se trata de una evolución del mismo algoritmo.

10. Trabajo Futuro

El presente trabajo de fin de máster ha servido para introducir el problema de lead scoring y aprender el uso y funcionamiento de diferentes algoritmos de machine learning utilizados hoy en día.

Sin embargo, hay una serie de puntos que no han podido cubrirse en la extensión deseada, que deberían recibir un análisis más profundo o que, directamente, se han dejado para ahondar en ellos en un futuro.

Dentro de estos puntos los más destacados son:

- En primer lugar, no ha podido tratarse de forma práctica todo el proceso previo de ETL, generación de variables y extracción desde las fuentes. Es cierto que estas tareas no corresponden con el contenido principal del máster. Sin embargo, gran parte del trabajo del científico de datos consiste en acompañar al dato desde su inicio, cuando está desordenado y ubicado en múltiples tablas, hasta su forma final en el dataset de entrenamiento. Es cierto que la parte de modelización es donde el científico de verdad muestra todo su conocimiento y aporta valor añadido, y también es cierto que disponer de una arquitectura que permita realizar este tipo de simulaciones en el entorno académico no es nada sencillo de obtener. Sin embargo, en un trabajo futuro sería interesante abordar el problema desde el inicio.

- El dataset escogido puede no haber sido lo suficientemente interesante como para abordar un análisis más detallado con algunos algoritmos. Al disponer de unos datos tan bien diferenciados, y con ambas clases tan equilibradas, la mayoría de ensayos han dado poco margen para personalizar cada modelo, puesto que en pocos o ningún caso se producía sobreajuste o cualquier comportamiento digno de estudio. En este caso, sería interesante disponer de datos reales provenientes del sector privado, que permitan un tratamiento más profundo y den más juego en la evaluación de cada algoritmo.

- El algoritmo de XGBoost se han introducido de forma muy superficial, no pudiendo destacar sus bondades frente al resto. Quizá se podría haber realizado una partición de los datos para generar un dataset más compacto y sencillo, que permitiera hacer frente al reto computacional que supone el entrenamiento de este algoritmo. Se ha dejado para posibles futuros proyectos abordar este tema.

- Del mismo modo, hay algunos aspectos de ciertos algoritmos que se han tratado por encima. Por ejemplo, hay muchos más parámetros aparte de los expuestos que podrían haberse estudiado para la mayoría de los algoritmos (el número máximo de nodos en GBM, la función de pérdida escogida en GBM, el uso de Bootstrap en random forest frente a su no utilización etc). Cabe aclarar que los parámetros estudiados se han elegido por tener una influencia muy grande

sobre el desempeño de cada modelo, pero quizá podría llevarse a cabo una personalización más profunda en futuros análisis.

- También hay algunos algoritmos muy usados hoy en día que podrían haberse añadido al estudio, como LGBM o algunos modelos de ensamblados además de los modelos de stacking desarrollados, como *Ensembled Voting*.

- Otra área en la que no se ha hecho hincapié es en la de los algoritmos no supervisados. Quizá podría haberse abordado un problema de segmentación posterior a la generación de las puntuaciones de cada contacto, utilizando métodos el análisis clúster mediante *k-medias*. Este tipo de algoritmos son muy utilizados en el ámbito del marketing y aportan un gran valor a la hora de crear campañas de segmentación.

- Como punto a seguir desarrollando, se podría expandir la evaluación de las predicciones obtenidas. Quizá se podría probar con más puntos de corte, encontrar el punto óptimo o incluso probar nuevas estrategias para clasificar a los leads calientes.

- Habría sido interesante realizar una evaluación más profunda de los resultados, generando un perfil tipo (*buyer persona*) en base a los hot leads de la empresa. Esto habría permitido ahondar en algunas tareas importantes del área de marketing de la empresa, quizá incluso planteando alguna campaña destinada a ellos. Sin embargo, esto no se ha llevado a cabo porque se aleja de las competencias propias de este máster.

- Otro aspecto en el que no se ha sacado todo el potencial posible es en el de las herramientas utilizadas. Durante todo el tabajo nos hemos ceñido a R y, en menor medida, SAS Miner. Sin embargo, un buen científico de datos es capaz de emplear diversas herramientas, beneficiándose de las ventajas de cada una. Quizá podría haberse llevado el entrenamiento de algún algoritmo en Python, que está más optimizado que R para determinadas tareas. Por ejemplo, podrían haberse introducido las redes neuronales con varias capas (Deep learning) aprovechando las librerías *Keras* y *TensorFlow*. El paquete *scikitlearn* de Python también incorpora muchos otros algoritmos de interés, como pueden ser HGBM (que se ha mencionado durante el trabajo y está aún en estado experimental), métodos lineales como *Lasso* o *Ridge*, métodos basados en *stacking* etc (*scikit-learn: machine learning in Python — scikit-learn 0.24.2 documentation*, s. f.).

En definitiva, el trabajo realizado ha sido satisfactorio desde un punto de vista de aprendizaje e introducción a nuevos conceptos. Sin embargo, hay numerosos puntos por explotar. Desde la explotación del origen de los datos hasta la utilización de algoritmos de ensamblado más avanzados y el uso de algoritmos no supervisados, pasando por la formación del propio dataset y sus variables, la profundización en las tareas propias del área de marketing etc.

11. Bibliografía

Aplicación de algoritmos Random Forest y XGBoost en una base de solicitudes de tarjetas de crédito. (s. f.). Recuperado 18 de agosto de 2021, de http://www.scielo.org.mx/scielo.php?pid=S1405-77432020000300002&script=sci_arttext

Ashish. (2018). *Leads Dataset* [Data Science Community]. Kaggle - Lead Scoring. https://kaggle.com/ashydv/leads-dataset

Barnhard, A. (2020, septiembre 27). *A True End-to-End ML Example: Lead Scoring*. Medium. https://towardsdatascience.com/a-true-end-to-end-ml-example-lead-scoring-f5b52e9a3c80

Bhalla, D. (s. f.). Ensemble Methods in R: Practical Guide. *ListenData*. Recuperado 29 de agosto de 2021, de https://www.listendata.com/2015/08/ensemble-learning-stacking-blending.html

Glm function | R Documentation. (s. f.). Recuperado 21 de febrero de 2021, de https://www.rdocumentation.org/packages/stats/versions/3.6.2/topics/glm

Lead nurturing y lead scoring en el inbound marketing—InboundCycle.pdf. (s. f.). Recuperado 1 de enero de 2021, de https://cdn2.hubspot.net/hubfs/136661/1-Content/offer/Lead-nurturing-lead-scoring/Lead%20nurturing%20y%20lead%20scoring%20en%20el%20inbound%20marketing%20-%20InboundCycle.pdf

Lead Scoring (Logistic Regression). (s. f.). Recuperado 1 de enero de 2021, de https://kaggle.com/ashydv/lead-scoring-logistic-regression

Pipis, G. (2020, mayo 21). *How to build Stacked Ensemble Models in R | R-bloggers*. https://www.r-bloggers.com/2020/05/how-to-build-stacked-ensemble-models-in-r/

Porras Blanco, M. (2018, abril 6). *¿Qué es el Lead Scoring? ¿En qué consiste y para qué sirve?* [Blog]. Semrush Blog. https://es.semrush.com/blog/que-es-lead-scoring/

RPubs—Introducción al paquete Caret. (s. f.). Recuperado 21 de febrero de 2021, de https://rpubs.com/joser/caret

Scikit-learn: Machine learning in Python—Scikit-learn 0.24.2 documentation. (s. f.). Recuperado 28 de agosto de 2021, de https://scikit-learn.org/stable/

Vega, J. B. M. (2020, agosto 12). Tutorial: XGBoost en Python. *Medium*. https://medium.com/@jboscomendoza/tutorial-xgboost-en-python-53e48fc58f73