Trabajo Fin de Máster

Bootstrap y métodos de ensamblado en modelos de regresión y clasificación

Sergio Gómez Vilas

Máster en Técnicas Estadísticas

Curso 2021-2022

II

Propuesta de Trabajo Fin de Máster

Título en galego: Título do Traballo Fin de Máster: Bootstrap e métodos de ensamblado en modelos de regresión e clasificación
Título en español: Título del Trabajo Fin de Máster: Bootstrap y métodos de ensamblado en modelos de regresión y clasificación
English title: Master's Thesis title: Bootstrap and ensemble methods in regression and clasification techniques
Modalidad: A Modalidad A
Autor/a: Sergio Gómez Vilas, Universidad de Santiago de Compostela
Director/a: Director: Manuel Febrero Bande, Universidad de Santiago de Compostela
Breve resumen del trabajo: En el presente trabajo trataremos de realizar una aproximación a los distintos modelos de aprendizaje estadístico supervisado existentes tratando de describirlos de la forma más parsimoniosa y clara posible. También nos acercaremos a las principales técnicas de ensamblado estadístico existentes revisando la bibliografía con el objetivo de explicar sus principales características subyacentes y los casos de uso en los que son de utilidad. En el apartado final se tratará de implementar algunos de ellos con el fin de compararlos a la hora de enfrentarnos a conjuntos de datos reales.
Recomendaciones:
Otras observaciones:

Don/doña Director: Manuel Febrero Bande, Catedrático de Universidad de la Universidad de Santiago de Compostela, informan que el Trabajo Fin de Máster titulado

Bootstrap y métodos de ensamblado en modelos de regresión y clasificación

fue realizado bajo su dirección por don/doña Sergio Gómez Vilas para el Máster en Técnicas Estadísticas. Estimando que el trabajo está terminado, dan su conformidad para su presentación y defensa ante un tribunal.

En Madrid, a 31 de Agosto de 2022.

El/la director/a:	El/la autor/a:
Don/doña Director: Manuel Febrero Bande	Don/doña Sergio Gómez Vilas

---

Declaración responsable. Para dar cumplimiento a la Ley 3/2022, de 24 de febrero, de convivencia universitaria, referente al plagio en el Trabajo Fin de Máster (Artículo 11, Disposición 2978 del BOE núm. 48 de 2022), **el/la autor/a declara** que el Trabajo Fin de Máster presentado es un documento original en el que se han tenido en cuenta las siguientes consideraciones relativas al uso de material de apoyo desarrollado por otros/as autores/as:

- Todas las fuentes usadas para la elaboración de este trabajo han sido citadas convenientemente (libros, artículos, apuntes de profesorado, páginas web, programas,...)
- Cualquier contenido copiado o traducido textualmente se ha puesto entre comillas, citando su procedencia.
- Se ha hecho constar explícitamente cuando un capítulo, sección, demostración,... sea una adaptación casi literal de alguna fuente existente.

Y, acepta que, si se demostrara lo contrario, se le apliquen las medidas disciplinarias que correspondan.

Índice general

Resumen IX

1. Introducción al aprendizaje estadístico 1
1.1. Aprendizaje estadístico no supervisado vs. supervisado . . . 1
1.2. Notación en aprendizaje estadístico . . . 2
1.3. Construcción y evaluación de modelos de aprendizaje estadístico . . . 3
1.3.1. Evaluación del modelo . . . 4

2. Modelos de aprendizaje estadístico 7
2.1. Modelos de regresión . . . 7
2.1.1. Modelo de regresión lineal . . . 8
2.1.2. Modelo lineal generalizado . . . 8
2.1.3. Modelo de regresión no paramétrica . . . 10
2.2. Árboles de Decisión . . . 13
2.2.1. Árboles de Regresión . . . 14
2.2.2. Árboles de Clasificación . . . 15
2.2.3. Sobreajuste u Overfitting . . . 16
2.3. Redes Neuronales . . . 16
2.3.1. ¿Qué es una Red Neuronal? . . . 17
2.3.2. Función de activación . . . 19
2.3.3. Función de coste . . . 20
2.3.4. Entrenamiento e hiperparámetros. . . . 21
2.4. Support Vector Machines (SVM) . . . 22
2.4.1. Hiperplano . . . 22
2.4.2. Clasificadores de máximo margen . . . 23
2.4.3. Clasificadores de margen débil . . . 24
2.4.4. Máquinas de soporte vectorial . . . 25

3. Ensamblado y Bootstrap 27
3.1. Bagging . . . 27
3.1.1. Bootstrap . . . 28
3.1.2. Metología del Bagging . . . 28
3.1.3. Random Forest . . . 29
3.2. Boosting . . . 30
3.2.1. Adaptative Boosting . . . 30
3.2.2. Gradient Boosting . . . 31
3.3. Stacking . . . 32
3.3.1. Stacking multinivel . . . 32

4. Aplicación de los modelos a problemas reales **33**
4.1. Librerías de R . . . 33
4.2. Aplicación . . . 34
4.2.1. Problema 1 . . . 34
4.2.2. Problema 2 . . . 36

5. Conclusiones y reflexiones. **41**

A. Código utilizado **43**

Bibliografía **61**

Resumen

Resumen en español

Los modelos de aprendizaje estadístico supervisado han demostrado ser una poderosa herramienta en el campo de la modelización estadística de tipo predictivo para tratar datos de distinto tipo y campo científico de estudio. Dependiendo de la naturaleza del problema y de los datos que este nos proporciona disponemos de una amplia gama de técnicas estadísticas para extraer información de valor. A su vez, los métodos de ensamblado han aprovechado distintas técnicas matemáticas como el Bootstrap o los algoritmos de optimización para mejorar significativamente las capacidades predictivas de los modelos de aprendizaje estadístico. En el presente trabajo trataremos de realizar una aproximación a los distintos modelos de aprendizaje estadístico supervisado existentes tratando de describirlos de la forma más parsimoniosa y clara posible. También nos acercaremos a las principales técnicas de ensamblado estadístico existentes revisando la bibliografía con el objetivo de explicar sus principales características subyacentes y los casos de uso en los que son de utilidad. En el apartado final se tratará de implementar algunos de ellos con el fin de compararlos a la hora de enfrentarnos a conjuntos de datos reales.

English abstract

Supervised statistical learning models have shown to be a powerful tool in the field of predictive statistical modeling to process data of different types and from distinct science branches. Depending on the nature of the problem and the data it provides us, we have a wide range of statistical techniques to extract valuable information. Simultaneously, ensemble methods have taken advantage of different mathematical techniques such as Bootstrap or optimization algorithms to significantly improve the predictive performance of statistical learning models. In the present document we will try to make an approach to the different supervised statistical learning models trying to describe them as most clear and parsimonious way possible. We will also dicuss the main statistical ensemble techniques reviewing the bibliography focused on explaining their main underlying characteristics and the use cases where they are useful. In the final section we will try to implement some of them in order to compare their results when facing real data sets.

Capítulo 1

Introducción al aprendizaje estadístico

Cada segundo que pasa en el mundo se generan millones de datos de diferentes orígenes y tipos. Un informe médico, un proceso industrial, la última compra que hicimos en el supermercado generan datos que pueden ser explotados por los diferentes sujetos interesados para generar conocimiento y valor. En el caso de la medicina se pueden estudiar relaciones entre ciertas prácticas y sus consecuencias sobre la calidad de vida o la salud de las personas. En el caso de la industria y servicios permiten reducir costes y mejorar la calidad de los servicios ofertados. Podríamos seguir enumerando un largo número de etcéteras que motivan el desarrollo de técnicas que permitan sacar partido a los datos pero empezaremos a adentrarnos y profundizar en las herramientas existentes para dicha tarea.

El aprendizaje estadístico se puede entender como el conjunto de técnicas matemáticas que tratan de entender y explotar, es decir obtener conocimiento, de los datos que se generan en un entorno concreto. Tratar de entender los datos es algo fundamental, ya que sin saber cómo operan y la información que aportan a un determinado objetivo no se podría obtener valor en forma de conocimiento a partir de los mismos. Por lo tanto, la obtención de valor a partir de los datos depende fuertemente de que las técnicas utilizadas para entender los datos sean correctas y no presenten sesgos a la hora de evaluarlos. Cuando hablamos de explotar la información que aporta un determinado conjunto de datos concreto, una de las principales fortalezas de los aprendizaje estadístico es la posibilidad de predecir eventos futuros basándonos en la elaboración de modelos que se entrenan con datos de eventos pasados. Por lo tanto, uno de los principales objetivos del aprendizaje estadístico desarrollar modelos que permitan la obtención de predicciones futuras.

1.1. Aprendizaje estadístico no supervisado vs. supervisado

Dentro de los modelos de aprendizaje estadístico podemos destacar dos grandes bloques: El estadístico no supervisado y el aprendizaje estadístico supervisado.

Aprendizaje estadístico no supervisado

El aprendizaje estadístico no supervisado comprende una serie de técnicas estadísticas cuyo objetivo no es explícitamente definido por quién las implemente. Es decir, trata de explorar y entender un conjunto de datos en bruto para extraer ciertas conclusiones a partir de los datos sin realizar asunciones respecto de los mismos. Por ejemplo dado un conjunto de datos, estas técnicas podrían explorar cómo se distribuyen, si existen patrones subyacentes, si existen relaciones o subconjuntos dentro de los mismos o si se dan comportamientos atípicos respecto al grueso de los datos en ciertos datos. Existen diferentes categorías de aprendizaje no supervisado las principales serían [5]:

- **El análisis descriptivo:** Que englobaría todas las técnicas que pretenden describir y analizar un grupo de datos, sin realizar inferencias sobre la población a la que pertenecen. Incluyendo tanto técnicas puramente numéricas como gráficas.
- **Métodos de reducción de la dimensión:** Son un conjunto de técnicas que pretenden reducir en el conjunto de datos el número de variables que aportan información relevante a cerca del mismo. Se pueden destacar entre otras el análisis de componentes principales (PCA) y el análisis factorial.
- **Análisis clúster:** Comprende un grupo de técnicas que tratan de agrupar el conjunto de datos en subgrupos homogéneos que no son conocidos a priori. Destacan los métodos jerárquicos y el método de los centroides.
- **Técnicas de detección de valores atípicos:** Se trata de un conjunto de técnicas muy utilizadas en el campo de la minería de datos y que pretenden encontrar datos anómalos en el conjunto de datos con el que se trabaja. Los datos atípicos tienen propiedades diferentes respecto a la mayoría del conjunto que pueden generar errores durante los posibles análisis que se realicen y que la información que se extraiga de ellos sea errónea. Aunque algunos de los métodos anteriores de aprendizaje no supervisado se pueden utilizar para la detección de datos anómalos destacamos aquí las técnicas basadas en distancias entre datos cómo la basada en la distancia de Mahalanobis.

Aprendizaje estadístico supervisado

El aprendizaje estadístico supervisado será en el que basemos el presente trabajo de fin de máster. Éste engloba todos aquellos modelos matemáticos en los que se trata fundamentalmente de alcanzar un objetivo predictivo previamente definido. Para ello, el investigador que dirige el estudio, decide previamente a la construcción de uno u otro modelo qué variables se definen como predictoras y qué variable se utilizará como respuesta.

En Estadística se considera la clasificación como un caso particular de la regresión, sin embargo es habitual diferenciar en aprendizaje estadístico entre estos dos tipos de problemas. Más adelante veremos que las técnicas abordadas en el presente trabajo se pueden usar tanto para clasificación como para regresión con pequeñas diferencias en el modelado. Diferenciamos entonces en aprendizaje estadístico entre:

- **Clasificación:** Engloba aquellas técnicas de aprendizaje estadístico en las que la variable de respuesta se define como cualitativa. Por ejemplo, el objetivo de nuestro estudio puede ser, dado un conjunto de predictores relativos a condiciones fisiológicas, determinar en función de los valores de estos si una persona está enferma o no. En este caso la variable respuesta sería una variable cualitativa de tipo binario, tendría dos posibles valores.
- **Regresión:** La regresión trata de construir modelos en los que la variable de respuesta es cuantitativa. Un ejemplo podría ser determinar la velocidad máxima de un coche, que sería la variable respuesta, en función de una serie de predictores como podrían ser la cilindrada del motor, la potencia del mismo y el peso del vehículo. La variable de respuesta en este caso sería una velocidad máxima que podría tomar valores que no están condicionados a categorías cerradas.

1.2. Notación en aprendizaje estadístico

En el presente trabajo trataremos los distintos elementos que componen planteamiento de los modelos de aprendizaje estadístico utilizando terminología proveniente de la estadística clásica aunque también se utilizarán algunos términos relativamente relacionados con el aprendizaje estadístico, y por tanto, más modernos. En la siguiente lista destacaremos los principales:

- **Variables:** Llamaremos a las variables explicativas alternativamente variables predictoras o predictores. Generalmente se representará como $\mathbf{X} = (X_1, \ldots, X_p)$ al vector de variables predictoras, que podrán ser continuas o categóricas. La variable de respuesta se representará como Y y al igual que en el caso de las predictoras podrá ser también tanto continua como categórica.
- **Observaciones:** Se denotarán a las observaciones del modelo como (y_i, x_i). Cabe la posibilidad de que las observaciones para los predictores se presenten como vectores o matrices, en cuyo caso será especificado en el apartado correspondiente.
- **Categorías:** Generalmente en los modelos de clasificación a las distintas categorías categorías a las que puede pertenecer Y se las denotará como $K = (K_1, \ldots, K_n)$.
- **Otros términos de relevancia:** Generalmente en los modelos de clasificación a las distintas categorías a Se denotará como ε al componente de error de los modelos de forma general.

1.3. Construcción y evaluación de modelos de aprendizaje estadístico

A la hora de construir un modelo de aprendizaje estadístico se distinguirá entre parámetros estructurales e hiperparámetros [5]: Mientras que los parámetros estructurales se ajustan en función de los datos durante el ajuste de modelo, los hiperparámetros serán aquellos que imponen restricciones al ajuste del modelo. Por ejemplo en un modelo de ajuste polinómico local el hiperparámetro ventana impone la restricción de cuantos datos se utilizarán en el ajuste local o el número de nodos de un árbol de decisión que permite crecer al árbol sin límite. La selección correcta de los hiperparámetros es importante ya que influyen de forma determinante en la complejidad del modelo y por ende en el balance entre sesgo y la varianza.

Equilibrio entre sesgo y varianza

A la hora de construir un modelo de aprendizaje estadístico dos factores a tener muy en cuenta son el sesgo y la varianza del mismo. Definiremos el sesgo como el error de un modelo de aprendizaje estadístico relativo a la diferencia entre las predicciones de nuestro modelo y sus valores reales. De forma ideal se aspira a ajustar modelos de aprendizaje estadístico con un sesgo bajo, es decir, que se haya determinado de forma correcta cómo se relacionan las variables predictoras y respuesta en los datos de entrenamiento. Por otra parte, el error asociado a la varianza del modelo se refiere a la medida en que la estimación del modelo variará al utilizar para predecir otros datos distintos a los del modelo. Uno de los objetivos del aprendizaje estadístico es que nuestro modelo tenga baja varianza para que se pueda generalizar y una vez probemos el modelo con datos reales este se adapte a los datos nuevos generando predicciones correctas.

Si hablamos en términos de complejidad, un modelo complejo es aquel que incluye una gran cantidad de predictores. A medida que la complejidad aumenta el error asociado al sesgo disminuye, pero como contraparte la varianza del modelo también aumenta, sucediendo lo equivalente al disminuir la complejidad (el sesgo aumenta y la varianza disminuye). Cuando la varianza del modelo aumenta significativamente el modelo corre el riesgo de producir **sobreajuste**. Este fenómeno implica que el modelo se adapta perfectamente a los datos de entrenamiento pero falla sistemáticamente al tratar de hacer predicciones sobre otros datos distintos. En el caso de un modelo con varianza reducida y sesgo elevado estaríamos ante un modelo incapaz de predecir debido a su pobre ajuste.

Se hace pues necesario en todo modelo alcanzar un equilibrio que permita reducir notablemente el sesgo del modelo en base a incrementar la complejidad del mismo pero sin hacerlo excesivamente para

evitar caer en el sobreajuste. En la figura 1.1 podemos observar gráficamente la relación entre el sesgo, la varianza y el error total del modelo en función de la complejidad.

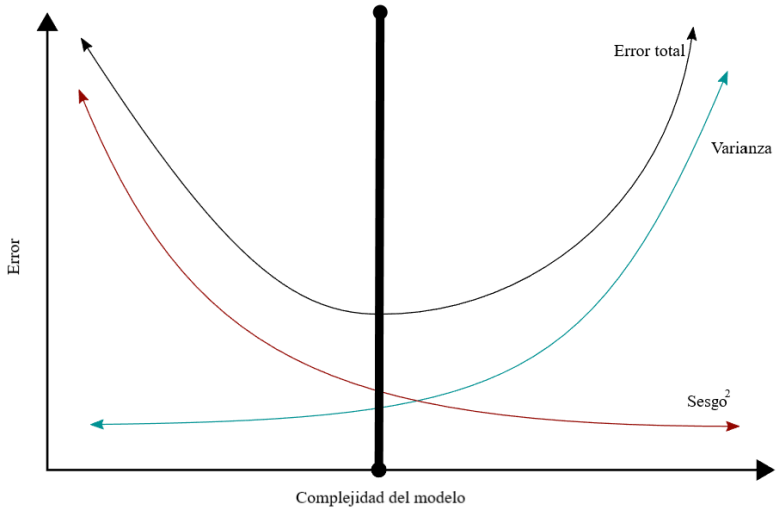

Figura 1.1: Relación entre el error total del modelo y su complejidad. Representadas la curvas de la varianza y del sesgo. Fuente: Elaboración propia a partir de imagen sin copyright

1.3.1. Evaluación del modelo

A la hora de evaluar la precisión un modelo de aprendizaje estadístico, una de las formas más comunes es el dividir la muestra en trozos. Lo más común es dividir la muestra en dos partes que dividan los datos utilizando porcentajes en torno a: 70-80 % datos de la muestra para entrenamiento del modelo y el 20-30 % restante para realizar el test sobre el modelo. Esto conlleva la desventaja de la pérdida de muestra a la hora de entrenar el modelo, pero a cambio nos permite obtener una estimación del rendimiento del mismo y de evaluar si presenta sobreajuste a los datos de entrenamiento o no.

En lo referido a la forma de seleccionar los valores óptimos de los hiperparámetros del modelo se podría realizar una tercera partición de datos para la validación de los mismos, conllevando una pérdida acusada de datos. No obstante existen diferentes técnicas que emplean el remuestreo de los datos de entrenamiento permitiendo obtener submuestras a partir de la muestra de entrenamiento diferentes a la de entrenamiento. Estas submuestras obtenidas por remuestreo permitirán evaluar los hiperparámetros sin necesidad de realizar una ternera partición de datos. Una de las técnicas más utilizada es sin duda la **Validación Cruzada (CV)**, aunque también son punteros métodos basados en remuestreo **Bootstrap**. Aunque este último tiene tiene gran relevancia, postergaremos su descripción al capítulo 4 debido a su gran importancia a la hora de construir modelos de ensamblado.

Validación Cruzada

La validación cruzada es una técnica matemática que se utiliza para cuantificar el error de predicción de un modelo de aprendizaje estadístico mediante el remuestreo de los datos de entrenamiento. Existen varios tipos de validación cruzada de los cuales el más simple es el **LOOCV** *(Leave One Out Cross Validation)* que, como su nombre indica, consiste en realizar un ajuste sobre el conjunto de entrenamiento $n-1$ y utilizar la observación restante para evaluar la predicción. Se iteraría n veces cubriendo el set de entrenamiento completo, lo cual puede ser muy costoso computacionalmente.

Una segunda variante de la validación cruzada es la **K-Fold Cross Validation** que trata de resolver el problema de ajustar un modelo por observación del set de entrenamiento. En lugar de ello, segmenta el conjunto de datos en $K-$grupos y reproduce un procedimiento análogo al anterior de **LOOCV** pero en lugar de ajustar un modelo por observación lo hace por cada uno de los $K-$grupos.

Como comentamos previamente, los modelos de remuestreo y concretamente los de **Validación Cruzada** se usan durante el entrenamiento para poder ajustar múltiples modelos sin necesidad de nuevos datos de entrenamiento. La selección de los hiperparámetros se basa en elegir aquel valor del hiperparámetro que minimiza el error de predicción en el entrenamiento entre los modelos ajustados.

Evaluación de la precisión de las predicciones: Regresión y clasificación

Como comentamos anteriormente una vez el modelo ha sido entrenado y validado se procede a evaluar la precisión de las predicciones con el uso del subset de datos de destinados a ese propósito y con los que no se ha entrenado previamente el modelo. Distinguiremos dos formas particulares de proceder dependiendo si se trata de datos de respuesta continua o categórica, es decir, si se trata de un problema de regresión o de clasificación.

Test en modelos de regresión:

A la hora de testar la capacidad de predicción un modelo de regresión se prueba realizando predicciones sobre el conjunto de test. Estas predicciones deben ser comparadas mediante algún tipo de método para determinar si existe una alta correlación entre las predicciones y los valores reales $(\hat{y_i}, y_i)$, que sería lo deseable ya que significaría que el modelo es generalizable a datos ajenos al entrenamiento. Uno de los métodos más empleados para medir la correlación es utilizar el cuadrado del coeficiente de correlación entre las predicciones $(\hat{y_i})$ y los valores observados y_i, un **pseudo R cuadrado** $(\tilde{R}^2)$ [5]. Este $\tilde{R}^2$ se interpretaría de forma similar al coeficiente R^2 de una regresión lineal:

$$\tilde{R}^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \tilde{y_i})}{\sum_{i=1}^{n}(y_i - \tilde{y_i})^2} \tag{1.1}$$

Test en modelos de clasificación:

A diferencia de los modelos de regresión, en los modelos de clasificación no se necesita comprobar mediante correlación entre observaciones reales y predichas la precisión porque las variables de respuesta son categóricas y se puede hacer fácilmente la comparación de las predicciones con los valores reales. Para estudiar cómo predice el modelo sobre los datos de test se crea una **matriz de confusión** que no es más que una tabla de contingencia con las predicciones frente a los valores reales.

A partir de esta la tabla 1.1 podemos extraer las tasas de acierto de positivos y negativos, **TPR** (también conocido como **Sensibilidad**) y **TNR** (también conocido como **Especificidad**) respectivamente:

$$TPR = \frac{TP}{P} = \frac{TP}{TP + FN} \tag{1.2}$$

Matriz de confusión		
Observación real \ Predicción	Positivo	Negativo
Positivo	Verdaderos positivos (TP)	Falsos negativos (FN)
Negativo	Falsos positivos (FP)	Verdaderos negativos (TN)

Cuadro 1.1: Tabla de confusión. Fuente: elaboración propia a partir de [5]

$$TNR = \frac{TN}{N} = \frac{TN}{TN + FP} \tag{1.3}$$

Una medida global para obtener la tasa de aciertos ACC:

$$ACC = \frac{TP + TN}{P + N} = \frac{TP + TN}{TP + TN + FP + FN} \tag{1.4}$$

Sin embargo cuando las clases no están balanceadas se debe de usar la **Precisión Balanceada (BA)** o la $\mathbf{F_1}$ **Score**:

$$BA = \frac{TPR + TNR}{2} \tag{1.5}$$

$$F_1 = \frac{2TP}{2TP + FP + FN} \tag{1.6}$$

Capítulo 2

Modelos de aprendizaje estadístico

2.1. Modelos de regresión

Los modelos de regresión comprenden un extenso campo en el aprendizaje estadístico. Son un conjunto de modelos que tratan de explicar y predecir el valor de la variable explicativa Y en función de los valores que adopten los diferentes predictores $(X_1, \ldots, X_n)$ que son significativos a la hora explicar esta variable. Podemos definir un modelo de regresión [10] como un problema matemático en el cual existe una relación entre una variable respuesta nombrada como Y, y una o varias variables explicativas, o covariables, identificadas como X. La particularidad que diferencia esto de un problema de clasificación es que en este caso la variable de respuesta Y es cuantitativa.

De forma general podríamos definir la regresión como la esperanza de Y condicionada a ciertos valores que adopte X:

$$\mathbb{E}(Y|X) + \varepsilon = f(X) + \varepsilon \tag{2.1}$$

Donde tendríamos por un lado el componente del modelo controlable, o sistemático si se prefiere, y por otro la parte denotada por ε que correspondería al componente estocástico. Esta última parte, también conocida como componente de error del modelo o residuo, se asume que no depende de las covariables, que no se puede controlar y que proviene de factores externos aleatorios.

En términos generales, por lo tanto, el objetivo del modelo de regresión es utilizar las observaciones medidas de (Y, X) para ajustar un modelo mediante mínimos cuadrados ordinarios que explique la relación entre las mismas y permita predecir una en función de la otra:

- **Objetivo descriptivo**: Determinar en que modo la variable Y depende de la variable X. Esto permite conocer el tipo de dependencia entre variables que tenemos. Por ejemplo una dependencia de tipo lineal u otra de tipo cuadrático.
- **Objetivo predictivo**: con un modelo ya construido determinando de que manera la variable X modula los valores de la variable Y podremos utilizar dicho modelo para realizar predicciones del valor que tome la variable respuesta en función de los valores conocidos de la variable explicativa.

Un ejemplo claro de problema de regresión podría ser explicar el valor de la vivienda en función del porcentaje de pobreza de la población. Este ejemplo sería modelable y comprobable gracias a que se encuentra en el data set *Boston* de la librería *MASS*. de R En este caso, nuestra Y sería el valor de la vivienda el cual vendría determinado por el porcentaje de pobreza de la zona en la que se sitúa dicha vivienda que sería nuestra variable X, o variable predictora, en función de la cual variarían los valores de Y observados.

2.1.1. Modelo de regresión lineal

El modelo de regresión lineal es el más básico y más utilizado por su relativa sencillez en los diversos campos de la investigación científica. Hablaremos de regresión lineal múltiple debido a que este modelo permite explicar los valores de la variable respuesta Y en función de una o más variables explicativas $\boldsymbol{x} = X_1, \ldots, X_p$

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_p X_p + \varepsilon \tag{2.2}$$

donde β_0 es el intercepto, que se corresponde con los valores de Y cuando $\boldsymbol{x} = 0$. $\beta_1, \cdots, \beta_p$ representa el vector de parámetros que se deben determinar y que otorgarán los pesos a cada predictor del modelo determinado por el vector de variables explicativas $\boldsymbol{x}$. Además ε es el componente de error que está sujeto a una serie de hipótesis que serán mencionados en el siguiente apartado en el que se repasan las hipótesis que debe seguir el modelo.

Para este modelo se asumen las hipótesis[10] de:

- **Independencia**: Las variables aleatorias que representan los errores son mutuamente independientes.
- **Linealidad**: la función de regresión debe ser una recta adoptando así la fórmula antes mencionada:
$$Y = \beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p + \varepsilon \tag{2.3}$$
- **Homocedasticidad**: La varianza del error es constante para cualquier valor de las variable explicativas.
- **Normalidad**: el error tiene distribución normal de media 0 y varianza constante:
$$\varepsilon \in N(0, \sigma^2) \tag{2.4}$$

El modelo de regresión lineal múltiple se ajusta por el método de método de mínimos cuadrados minimizando la suma de los cuadrados de los residuos del modelo:

$$\min_{\beta_0, \beta_1, \ldots, \beta_p} \sum_{i=1}^{n} (y_i - \beta_0 - \beta_1 X_{1i} - \cdots - \beta_p X_{pi})^2 \tag{2.5}$$

donde X_{pi} corresponde a los valores de cada predictor enumerado con p para cada observación i dada y β_p es el valor de los coeficientes.

2.1.2. Modelo lineal generalizado

Los modelos lineales generalizados constituyen una extensión de los modelos lineales en los que la distribución de la variable respuesta no sigue una distribución normal. Por lo tanto sus errores no cumplen las hipótesis asumidas para poder aplicar un modelo lineal. Por consiguiente, en estos modelos se introduce una **función de enlace** o ***función link*** que transforme el modelo ajustado de la escala lineal a la escala original en la que se distribuyen los datos realmente. Si nos paramos a observar la regresión lineal presentada en el apartado anterior podemos determinar que es un caso particular de los modelos lineales generalizados en el que la distribución de la variable respuesta es normal, asumiendo las hipótesis planteadas, y la *función link* que se aplica es la identidad, es decir, que no requiere ser transformada.

A continuación presentamos un cuadro elaborado a partir de Thiele, J., & Markussen, B. (2012) [22] :

Cuadro 2.1: Funciones link y detalles utilizadas en modelos lineales generalizados.

Familia	**Nombre**	**Función de la media**
Gaussiana	Identidad 2	$\mu_i = x_i^\top \beta$
Binomial	Logit	$\mu_i = \frac{exp(x_i^\top \beta)}{1+exp(x_i^\top \beta)}$
Poisson	Log	$\exp(x_i^\top \beta)$
Poisson	Identidad	$x_i^\top \beta$
Poisson	Raíz cuadrada	$x_i^\top \beta^2$
Gamma	Inversa	$x_i^\top \beta^{-1}$

De estos consideraremos de especial relevancia el segundo, en el que la distribución de la variable Y corresponde con una función de la familia binomial, concretamente de Bernouilli. A partir de ello se construye la regresión logística.

Regresión logística

La regresión logística un modelo muy utilizado en estadística aplicada y en aprendizaje estadístico debido a su gran sencillez en la implementación a la gran cantidad de problemas que se presentan con respuesta binaria. Por ejemplo, en el caso de la bioestadística la clasificación entre enfermo o no enfermo atendiendo a una serie de variables presentaría una distribución de la variable de respuesta que encaja en lo que comentamos anteriormente.

Cuando nos encontramos ante un problema en el que la variable respuesta puede caer en dos categorías cerradas A o B (presencia o ausencia, éxito o error...) que generalmente se representan como 0 o 1 la variable de respuesta Y se distribuye de acuerdo a una función de distribución Bernouilli, donde podríamos llamar a 1 éxito y a 0 fracaso. Dada esta situación, lo que se hace con la regresión logística es modelar la probabilidad de que Y pertenezca a cada categoría. Es decir, en lugar de predecir un valor como se hacía en la regresión lineal, se predice la probabilidad de que una variable respuesta pertenezca a una categoría.

A diferencia de lo que sucedía con la regresión lineal, la variable respuesta del modelo de regresión logística deberá adoptar valores entre 0 y 1. Si para un problema en el cual la respuesta debe tomar un valor entre 0 y 1, tratamos de modelarlo con una recta de regresión es demostrable que nuestra variable dependiente tomará valores fuera del rango del espectro de probabilidad. Por este motivo, la regresión logística utiliza la propiamente llamada **función logística**, la cual establece una relación entre la variable respuesta y la variable explicativa de la forma $E(Y|\boldsymbol{x}) = p(\boldsymbol{x})$, donde $p(\boldsymbol{x})$ es:

$$P(\boldsymbol{x}) = \frac{e^{\beta_0+\beta_1 X_1+\cdots+\beta_n X_n}}{1 + e^{(\beta_0+\beta_1 X_1+\cdots+\beta_n X_n)}} \tag{2.6}$$

donde $\boldsymbol{\beta} = \beta_1 + \cdots + \beta_n$ representa los pesos del modelo y β_0 representa el sesgo del modelo. Los parámetros del modelo son ajustados mediante optimización. Debido a que la **función logística** permite obtener resultados en formato probabilístico, respuestas en el rango [0,1], lo cual facilita la interpretación, necesitamos una función inversa para poder calcular ajustar los coeficientes de regresión, aquí entra en juego la **función logit**. Se puede ajustar el modelo de forma sencilla usando esta transformación logarítmica de la forma:

$$logit(p) = \log\left(\frac{P(\boldsymbol{x})}{1 - P(\boldsymbol{x})}\right) = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_P X_P \tag{2.7}$$

Para estimar los coeficientes de la regresión logística se usa el estimador de máxima verosimilitud que consiste en buscar aquellos parámetros β para los cuales la probabilidad predicha para cada individuo $\hat{P}(\boldsymbol{x_i})$ usando 2.6 sea lo más verosimil posible. Se trata de hallar los $\hat{\beta}$ que sustituyendo con sus valores los parámetros del modelo 2.7 este arroje unas probabilidades $P(\boldsymbol{x_i})$ próximas a 1 si la característica dicotómica está presente y 0 si no lo está en el individuo i.

Esto se consigue con el set de parámetros β que maximiza la función de verosimilitud de los datos de entrenamiento que es expresada como el producto de las probabilidades predichas para las n observaciones:

$$\ell(\boldsymbol{X}|P) = \prod_{i=1, y_i=1}^{n} P(\boldsymbol{x_i}) \prod_{i=1, y_i=0}^{n} (1 - P(\boldsymbol{x_i})) \tag{2.8}$$

Dónde $(\boldsymbol{X}, \boldsymbol{y})$ son los datos de entrenamiento. $\boldsymbol{X}$ es una matriz de observaciones de las variables predictoras donde cada columna corresponde con un predictor y las observaciones de la primera columna tienen valor 1. $\boldsymbol{y}$ es el vector de respuestas de dimensión n, $\boldsymbol{x_i}$ es el vector de observaciones para los predictores (correspondiente a una fila i de la matriz $\boldsymbol{X}$) e y_i son las observaciones de la variable respuesta.

2.1.3. Modelo de regresión no paramétrica

Los modelos de regresión no paramétricos no asumen una forma particular para la función de regresión. En lugar de ello, se aproxima la forma de la función de regresión construyéndola directamente de acuerdo con la información derivada de los datos. Estos métodos no suponen ninguna forma concreta de la media condicional de la variable respuesta respecto de los predictores, sino que adaptan la forma de la función a partir de los datos. Esta forma de estimar necesita de tamaños de muestra mucho más elevados que en el caso de los modelos paramétricos porque los datos deben dar soporte a la estructura del propio modelo.[17]

La forma de la función de regresión se expresaría de la forma:

$$Y = f(X_1, \ldots, X_p) + \varepsilon \tag{2.9}$$

donde f es una función *suave* de las variables predictoras del modelo. Gran parte de estos modelos se los conoce como modelos locales debido a que ajustan el modelo para una determinada observación en función de los datos cercanos a la misma. Para que la definición de un modelo local sea correcta y permita predecir correctamente deberemos entrenarlo con numerosas observaciones.

Regresión local

La regresión local es una técnica que se utiliza para ajustar modelos a los datos en una ventana de tamaño fijo en lugar de en el conjunto de datos completo. Se puede hacer de forma manual, especificando

el tamaño de la ventana y el número de datos que se utilizarán en cada paso, o de forma automática, utilizando un algoritmo que seleccione el tamaño de la ventana y el número de datos de forma dinámica. Una de las formas de modelar la regresión local más destacable es la **regresión polinómica local ponderada** .

Propuesta por Cleveland (1979)[7], la regresión polinómica local ponderada es una técnica que trata de ajustar curvas polinómicas para pequeñas ventanas del total de los datos (curvas polinómicas locales) centradas en cada paso en un punto x_0 distinto. Con cada uno de los ajustes locales se estima el valor predicho para x_0 y así se va construyendo la curva de regresión. Las curvas polinómicas locales se ajustan utilizando mínimos cuadrados ponderados por pesos (por sencillez usamos el caso univariante, es decir, con un sólo predictor):

$$\min_{\beta_0,\beta_1,\ldots,\beta_d} \sum_{i=1}^{n} (Y_i - \beta_0 - \beta_1(x_0 - X_i) - \ldots - \beta_d(x_0 - X_i)^d)^2 K_h(x_0 - X_i) \tag{2.10}$$

Donde d es el grado del polinomio utilizado para la estimación local, x_0 es el punto donde se centra el ajuste local y el término $K_h(x_0 - X_i)$ corresponde a la función que otorga los pesos:

$$K_h(x_0 - X_i) = \frac{1}{h} K\left(\frac{x_0 - X_i}{h}\right) \tag{2.11}$$

donde K es una función tipo núcleo (Kernel) que por norma general suele ser una función de densidad de media 0. El hiperparámetro ventana, h, se usa para suavizar la curva y es el que regula el número de observaciones que entran en cada ajuste polinómico local. Este hiperparámetro de ventana es uno de los hiperparámetros más importantes y para su selección existen distintos procedimientos entre los que destacan la validación cruzada y validación cruzada generalizada (CV y GCV por sus siglas en inglés) o los selectores plug-in.

Cabe destacar debido a a la importancia del algoritmo en el aprendizaje estadístico, que si usamos como función núcleo una densidad uniforme centrada en el 0, por ejemplo $\mathcal{U}(-1, 1)$, (otorgando los mismos pesos a cada dato) y usando un polinomio de grado $d = 0$ (tomando la media local) estaremos ante el algoritmo ***K-NN*** o ***K-vecinos más próximos*** para construir una regresión de forma no paramétrica. Este algoritmo también se utiliza al igual que los métodos aplicados en este apartado para construir clasificadores, aunque nos estamos centrando en la regresión en el presente apartado.

Splines

Los **Splines** son un conjunto de funciones polinómicas sobre las que imperan restricciones en los puntos de unión entre las mismas denominados **nodos**. Estos nodos dividen el rango del predictor X en regiones para las cuales se ajusta un polinomio en cada una, lo que implica que se ajustará una función polinómica en cada uno de los segmentos delimitados por los nodos. Con lo cual la función de regresión se conformará por un conjunto de funciones continuas en sus intervalos de definición.

- **Regresión con Splines**:

  En vez de ajustar un polinomio, que posiblemente acabase siendo una función bastante compleja, a lo largo de todo el espacio definido por la matriz de predictores X, se divide el espacio de predictores en subintervalos y se ajustan polinomios de menor grado paso a paso. Los puntos que dividen el espacio de predictores se conocen como **nodos**. La función se construye ajustando polinomios de orden d e incluyendo ciertas restricciones que permitan que los Splines sean derivables hasta el orden $d - 1$. Habitualmente se suele optar por el conocido **Spline cúbico**

natural debido a que es continuo en el espacio delimitado por los nodos, en los que se anula su segunda y tercera derivada que implica que en los subintervalos extremos es lineal, lo cual reduce notablemente la variabilidad del spline en los extremos.

Para un espacio de datos situado en el intervalo $[a, b]$ donde $a < t_1 < t_2 \ldots < tn < b$ un Spline cúbico natural tendría una forma:

$$g(t) = a_i + b_i(t - t_i) + c_i(t - t_i)^2 + d_i(t - t_i)^3 \tag{2.12}$$

donde $t \in [t_i, t_{i+1}]$.

Un parámetro que se debe tener muy en cuenta a la hora de construir el modelo es el número de nodos, elegidos generalmente por validación cruzada. La regresión con Splines se ajusta con mínimos cuadrados pero tiene el problema de que la selección de nodos puede ser un proceso costoso.

- **Splines de Suavizado**:

  Los **Splines de Suavizado** o ***Smoothing Splines*** siguen una estrategia similar a la seguida para calcular la regresión con Splines introduciendo un componente de restricción y así evitar un modelo demasiado flexible y por consiguiente el sobreajuste del mismo:

$$\sum_{i=1}^{n}(y_i - g(x_i))^2 + \lambda \int g''(t)^2\, dt \tag{2.13}$$

  donde $\sum_{i=1}^{n}(y_i - g(x_i))^2$ es la función de coste, en este caso **RSS** (suma de los residuos al cuadrado), que ajusta los datos mientras que el término $\lambda \int g''(t)^2\, dt$ es el que penaliza $g(x)$ haciendo que disminuya la variabilidad y suavizando por consiguiente la curva resultante. Dado que la segunda derivada de la función g en el punto t mide la variación de la pendiente de la función que al ser integrada arroja la variación total de la pendiente. Si la curva ajustada es suave, la pendiente que coincide con la primera derivada $g'(t)$, tendrá un valor constante y por consiguiente la penalización será menor, ocurriendo al contrario en caso de que la variación de la pendiente sea grande. Por último el término λ es el parámetro que regula la penalización. Para valores de λ cercanos a 0 no se introducirá apenas penalización en la función y tenderá a ajustarse totalmente a los datos de entrenamiento mientras que para valores altos tenderá a suavizar la función.

- **P-Splines**: Una forma de construir la regresión con Splines es a través de los **Splines Penalizados**. Los más utilizados son los **P-Splines**[9] son un tipo de Splines que tratan de ajustar la función de regresión utilizando el menor número de parámetros (menor que los splines de suavizado) y en los que la selección del número de nodos no afecta tanto como en los Splines de regresión. Son Splines de bajo rango en los que el tamaño de la base es mucho menor que el número de datos lo que los hace computacionalmente más eficientes. La introducción de las penalizaciones disminuye la importancia de la elección de localización y cuantía de los nodos. Para los P-Splines la penalización empleada suele ser de orden $d = 2$. Esta forma de modelar utiliza una base de **B-Splines** que son un conjunto de splines que se unen en p nodos internos con propiedades bastante deseables entre las que destaca no ser afectados por el efecto frontera que incrementa la variabilidad de las predicciones en estos puntos y que padecen otros métodos de suavizado.

Modelos aditivos generalizados

Los modelos aditivos constituyen un tipo regresión en el que el ajuste del peso de cada predictor viene mediado por una función a la vez que se trata de mantener la aditividad del modelo. Por lo tanto estos modelos resultan en una combinación lineal de funciones no lineales:

$$Y = \beta_0 + f_1(X_1) + f_2(X_2) + \ldots + f_i(X_i) + \varepsilon \tag{2.14}$$

donde f_i para $i = 1, \ldots, n$ son las funciones que modelan cada variable predictora X_i.

Al igual que sucedía con los modelos lineales generalizados podemos utilizar una función link para transformar la escala del modelo ajustado a la escala real de los datos. En este caso estaríamos ante los conocidos como *GAM*, o modelos aditivos generalizados [11]. Para el caso de la regresión logística que describimos en el apartado de modelos lineales generalizados:

$$\log\left(\frac{p(X)}{1-p(X)}\right) = \beta_0 + f_1(X_1) + f_2(X_2) + ... + f_p(X_p) \tag{2.15}$$

donde en la regresión logística, utilizando la función logit (2.15), los parámetros β son sustituidos una serie de funciones no lineales para cada predictor $f_p(X_p)$.

Los GAM permiten ajustar funciones no lineales en cada predictor X_p y al ser modelos aditivos permiten estudiar el efecto de estos en la variable Y. Su mayor limitación viene de que, debido a su carácter aditivo, no contemplan las interacciones entre predictores de forma automática y habría que introducir esta interacción en el modelo de forma manual mediante un nuevo predictor que contemple esta interacción.

2.2. Árboles de Decisión

Los árboles de decisión son modelos de aprendizaje estadístico supervisado que, aunque se podrían incluir dentro de los modelos no paramétricos, por su importancia para la construcción de otros modelos consideramos necesario dedicarles un apartado separado. Los árboles construyen mediante reglas de decisión binarias, la idea subyacente es: modelar las sucesivas decisiones conformando un árbol, en el que cada nodo representa una decisión, y cada rama representa el resultado de tomar esa decisión. Los árboles de decisión se pueden utilizar tanto para problemas de clasificación como de regresión.

En el caso de los problemas de clasificación, cada nodo del árbol contiene una pregunta sobre los datos, y cada rama representa una respuesta posible a esa pregunta. El árbol se construye a partir de los datos de entrenamiento, de modo que cada pregunta se selecciona de tal manera que maximice la separación entre las clases de datos.

En el caso de los problemas de regresión, cada nodo del árbol contiene una decisión sobre los datos, y cada rama representa un valor posible de la variable objetivo. El árbol se construye de modo que cada pregunta se seleccione de tal manera que minimice la varianza de los datos.

Sus principales ventajas son:

- Los árboles de decisión son fáciles de interpretar y de explicar.
- Los árboles de decisión pueden modelar datos no lineales y variables categóricas.
- No se ven muy influenciados por outliers.

Sus principales desventajas son:

- Pueden ser muy sensibles a los cambios en los datos de entrada, lo que puede dar lugar a resultados impredecibles.
- Pueden crear modelos muy complejos que no se pueden interpretar fácilmente.
- Pueden ser propensos al sobreajuste de los datos.

Cuando queremos predecir una variable respuesta, ya sea continua o categórica, en función de una serie de variables explicativas podemos aplicar un árbol de decisión. El árbol de decisión es un modelo de aprendizaje estadístico que consiste en distribuir las observaciones de un determinado set de datos mediante bifurcaciones en las que se toman decisiones, generando una estructura de árbol que finaliza al alcanzar el nodo en el que no se puede dividir más, conocido como nodo terminal. Existen dos subtipos de árboles de decisión: aquellos en los que la variable respuesta es continua, conocidos como árboles de regresión, y el otro subgrupo que corresponde a aquellos en los que la variable respuesta es categórica, conocidos como árboles de clasificación.

2.2.1. Árboles de Regresión

Este árbol de decisión (Figura 2.1) ilustra con un ejemplo de la base de datos *Hitters* (con datos de bateadores de baseball) el salario que obtendrían en función del recorrido por el árbol de regresión conformado a través de los predictores de su modelo. Cada decisión del modelo implica una bifurcación que hace que el camino se ramifique en dos posibles direcciones. A estos puntos donde se determina que en base a unos valores u otros del predictor se deba tomar una u otra dirección en base al valor de corte establecido se les llama nodos internos. A los segmentos que conectan estos nodos descritos se les conoce como ramas. Los nodos que nos encontramos al final y en los que ya no existen más divisiones son los conocidos como nodos terminales, que se suelen denotar como $(R_1, R_2, \ldots, R_n)$. Para llegar a ellos se debe hacer un recorrido u otro por el árbol, para llegar a un valor predicho de nuestra variable respuesta. Como podemos apreciar en la imagen que al recorrer el árbol llegamos a los valores medios para el salario de los bateadores que es la variable respuesta del modelo de regresión.

Durante el entrenamiento de un árbol de regresión se crean múltiples caminos conformando un esquema de árbol en base a los datos de entrenamiento. Cuando se desea realizar una predicción en función de los valores de los predictores, éstos se evaluarán en cada nodo correspondiente mediante una decisión relacionada con el valor de dicho predictor. Superadas las distintas decisiones, se llegará a un nodo terminal en el cual la variable respuesta toma el valor de la media de las variables observadas durante el entrenamiento para ese nodo terminal.

El fin último de los árboles de decisión es llegar a los terminales $(R_1, R_2, \ldots, R_n)$ y que estos sean lo más fiables, en términos predictivos, que sea posible. A pesar del sencillo proceso que describimos antes, en el que para valores clave de los predictores se iban generando sucesivas bifurcaciones hasta llegar a nodos terminales, necesitamos un método de ajuste que nos permita identificar el valor óptimo en el cual generar la división en un nodo dado.

Para el caso de los árboles de regresión el criterio usado con más frecuencia es la suma de residuos al cuadrado **(RSS)**:

$$RSS = \sum_{j=1}^{J} \sum_{i \in R_j} (y_i - \hat{y}_{R_j})^2 \tag{2.16}$$

En la ecuación 2.16 el objetivo es encontrar las J regiones que minimizan el RSS, donde $\hat{y}_{R_j}$ es la media de la respuesta de la variable en la región R_j e y_{R_i} es una observación para la variable respuesta.

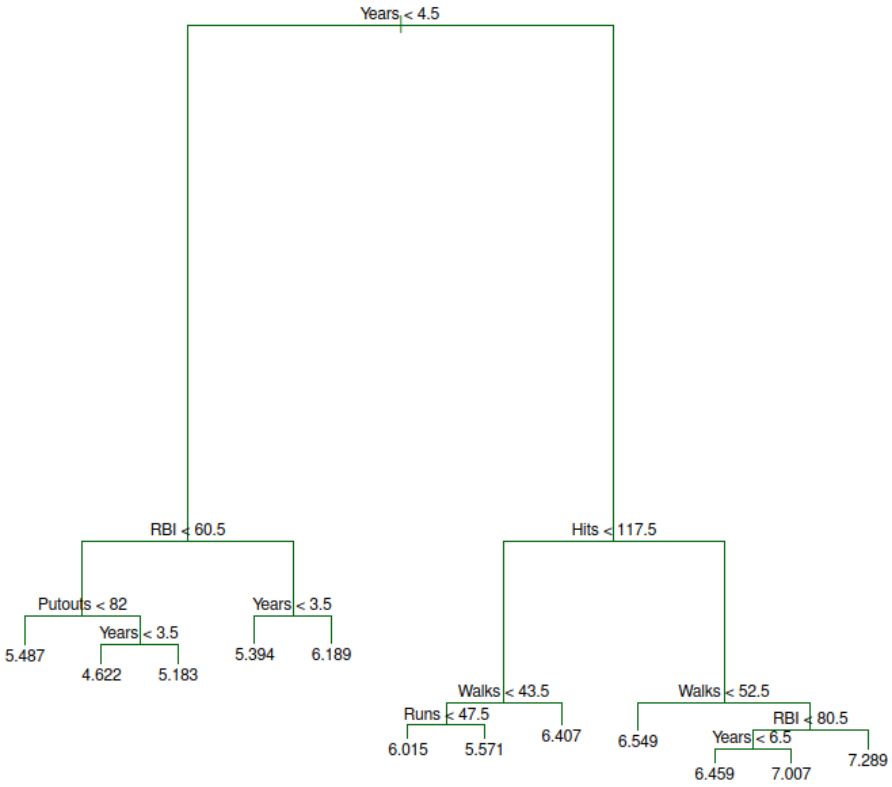

Figura 2.1: Arbol de regresión. Fuente: James, Gareth, et al., 2013. [17]

Para ello, y como no es posible encontrar todas las posibles divisiones para el conjunto de variables predictoras se recurre al **Recursive Binary Splitting**.

El Recursive Binary Splitting es un algoritmo que trata de encontrar en cada una de sus iteraciones el predictor X_j y el punto de corte s tal que las observaciones en las regiones $\{X|X_j < s\}$ y $\{X|X_j \geq s\}$, se obtiene la mayor reducción posible del RSS. Cabe destacar que en cada división el algoritmo evalúa un único predictor mediante una decisión binaria (*¿Es $X_1 > 4$? Si/No)* y que no tiene en cuenta la influencia de la división actual sobre las divisiones futuras.

2.2.2. Árboles de Clasificación

Los árboles de clasificación siguen un el mismo patrón de sucesos que mencionamos antes para el caso de los árboles de regresión pero con ciertos matices. Para empezar, la variable respuesta en este caso es categórica al igual que en otros modelos de clasificación. A pesar de que al igual que los árboles de regresión la construcción del árbol se hace siguiendo el algoritmo de Recursive Binary Splitting que describimos con los árboles de regresión, por la naturaleza de la variable respuesta la consecución de las divisiones óptimas no se puede hacer usando el método de RSS, si no que en este caso se dispone de varias opciones para llegar al árbol óptimo:

- **Gini Index**: Es una medida de pureza del nodo, mide la varianza total en el conjunto de las K clases del nodo m:

$$G_m = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk}) \tag{2.17}$$

Donde $\hat{p}_{mk}$ es la proporción de varianza para la clase k en el nodo m. Cuando esta proporción es cercana a 0 o a 1 el nodo contiene mayoritariamente observaciones de una determinada clase por lo que $\hat{p}_{mk}(1-\hat{p}_{mk})$ será muy pequeño. Por lo tanto el índice de Gini (G) informa de la pureza del nodo a razón de cuanto más puro sea, menor será (G).

- **Information Gain, Cross Entropy**: Es una medida para cuantificar el desorden del sistema. Haciendo referencia a la explicación anterior en este caso el desorden se corresponde con la impureza del nodo.

$$D = -\sum_{k=1}^{K} \hat{p}_{mk} \ \log(\hat{p}_{mk}) \tag{2.18}$$

Por lo tanto si D es cercano a cero estaremos ante un nodo de alta pureza. Podemos observar cómo, al igual que en la fórmula para el índice de Gini, para valores $\hat{p}_{mk}$ la pureza del nodo aumenta.

- **Chi Cuadrado**: Es un método que trata de comparar los nodos hijo con los nodos padre mediante un contraste de bondad de ajuste Chi Cuadrado. Para ello se usa como hipótesis nula la distribución esperada para cada clase en el nodo parental.

$$\chi^2 = \sum_{k} \frac{(\text{Frecuencia Observada}_k - \text{Frecuencia Esperada}_k)^2}{\text{Frecuencia Esperada}_k} \tag{2.19}$$

Cuanto mayores sean los valores de χ^2 mayor evidencia de la diferencia entre los nodos comparados.

2.2.3. Sobreajuste u Overfitting

Una de las principales limitaciones de los árboles tanto de regresión como de clasificación es que de forma natural, si no se limitan el número de decisiones, se terminan ajustando por completo a las observaciones de entrenamiento y creando por consiguiente un nodo terminal por observación, esto es el sobreajuste u overfitting. Para controlarlo existen dos estrategias:

- **Early stopping o parada temprana**: Mediante una serie de hiperparámetros se limita el árbol para detener su crecimiento y y el número de nodos que crea. Entre los hiperparámetros que controlan el tamaño del árbol están establecer un número de observaciones mínimas para generar una división, establecer un número de divisiones máximo para la rama más larga, limitar el número de nodos terminales máximo a un número o proponer una reducción mínima del error para que una bifurcación se lleve a cabo. Dependiendo de las distintas implementaciones y algoritmos para crear árboles de decisión se incluirán estas u otras medidas para controlar los árboles. El inconveniente de esta forma de controlar el crecimiento es que no tiene en cuenta las decisiones que se podrían tomar tras la división actual con lo cual es complicado llegar al árbol óptimo.

- **Tree prunning o podado**: Esta segunda estrategia consiste en generar un árbol mínimamente limitado en cuanto a tamaño y posteriormente proceder a evaluarlo en busca de la estructura óptima que mantiene la estructura robusta que consigue un test de error bajo para llegar al sub-árbol óptimo.

2.3. Redes Neuronales

Dentro del aprendizaje estadístico uno de los modelos que ha experimentado mayor crecimiento y ha suscitado mayor interés es el relativo a las redes neuronales. Los modelos de redes más sencillos con

perceptrones simples y arquitectura poco compleja se pueden implementar de manera relativamente sencilla y sin costo computacional elevado en R a través, por ejemplo, del paquete **nnet**. Los modelos más complejos y cuyas arquitecturas ya involucran desde perceptrones multicapa hasta modelos más complejos todavía como redes neuronales convolucionales, redes recurrentes entre otros tipos se consideran un paso más avanzado del aprendizaje estadístico a menudo nombrado como *deep learning*. Este último ya requiere de una mayor optimización de recursos o de mayor poder computacional para ser implementado. Se pueden implementar con diversas librerías como por ejemplo usando la API para redes neuronales complejas *Keras* desarrollada a partir de la librería *TensorFlow* que, aunque inicialmente se desarrolla en Python, también está disponible para R.

2.3.1. ¿Qué es una Red Neuronal?

En la Figura 2.2 primera capa de la red neuronal (color amarillo) se conoce como capa de entrada o *input layer* y recibe los datos de las observaciones, es decir, el valor de los predictores. La capa intermedia (color malva), conocida como capa oculta o *hidden layer*, recibe los valores de la capa de entrada, ponderados por los pesos (flechas grises). La última capa, llamada *output layer*, combina los valores que salen de la capa intermedia para generar la predicción.

A grandes rasgos cada uno de los puntos representa una neurona que realiza una operación sencilla y está conectada las neuronas de la capa anterior y a las de la capa siguiente de forma ponderada mediante pesos para regular la importancia de la información propagada. Estos pesos representados por las flechas grises se podrían equiparar a un modelo de regresión lineal, como los tratados en el apartado 1.1 a los coeficientes de regresión para cada variable predictora de modo que entre la neurona X_1 y la neurona A_1 se podría estar ponderando el envío de la información de la forma:

$$\boldsymbol{Input} = w_1X_1 + w_2X_2 + w_3X_3 + w_4X_4 + \varepsilon \tag{2.20}$$

donde representamos la información llegada a la neurona A_1 (Figura 2.2) como *Input*. Esta entrada es la suma ponderada de los pesos w para cada variable predictora de la anterior capa (neuronas $\{X_1, X_2, X_3, X_4\}$). Por último, el término ε representa el sesgo. Si a lo anterior le añadimos que dentro de la neurona se realiza un segundo proceso sencillo llamado función de activación tendríamos explicado el funcionamiento de la neurona.

La neurona por lo tanto es un elemento de la red que recibe la suma ponderada por los pesos de cada una de las neuronas de la capa anterior y en función del resultado de esta suma se aplica una función de activación que producirá una salida con destino a la siguiente capa. De forma pormenorizada:

El valor de entrada de cada neurona se puede definir matemáticamente como definimos en la ecuación 2.20 o de forma matricial como:

$$\boldsymbol{Input} = \mathbf{XW} + \varepsilon \tag{2.21}$$

Donde en lugar se usar el sumatorio de de los pesos ponderados se representa el producto vectorial en el que $\mathbf{X}$ es el vector de valores de entrada, $\mathbf{W}$ el vector de los pesos y ε el sesgo.

Como relatábamos anteriormente a la suma ponderada de inputs procedentes de la capa anterior se le aplica una función de activación a la que llamaremos $\mathbf{g}$. Esta función transformará la suma ponderada de inputs en el valor de activación, que llamaremos $\mathbf{a}$ que será la salida de esta neurona:

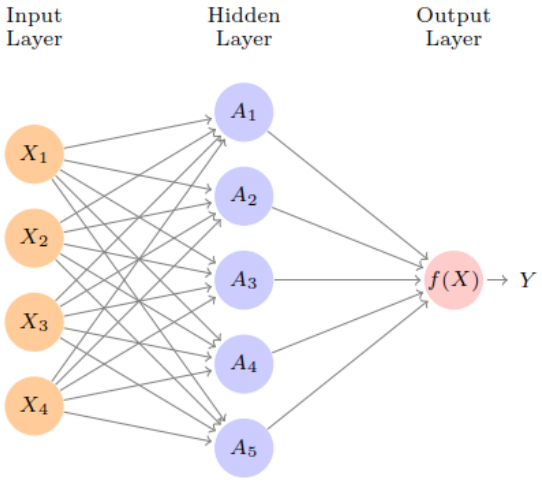

Figura 2.2: Red neuronal de una sola capa oculta. Fuente: James, Gareth, et al., 2013. [17]

$$a = g(\mathbf{X}\mathbf{W} + \epsilon) \tag{2.22}$$

En la capa de entrada, o *input layer*, la función de activación es sencilla: sale lo mismo que entra ya que lo que se quiere es únicamente pasar al siguiente nivel el valor de los predictores. En las siguientes capas (intermedias y de salida) se suelen utilizar otras funciones de activación cómo la función *Identidad* en problemas de regresión, ya que devuelve la entrada, o como la función *Softmax* en problemas de clasificación ya que transforma las salidas a una representación en forma de probabilidades lo cual es muy útil.

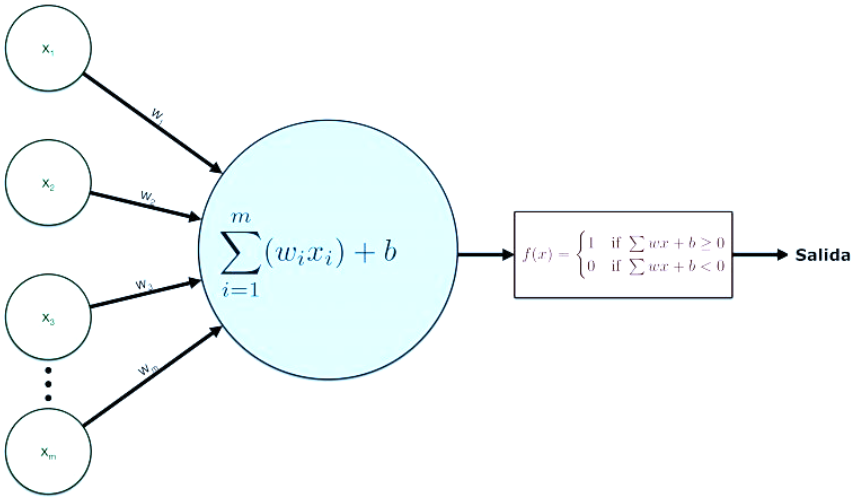

Figura 2.3: Representación de la neurona artificial. Fuente: https://magiquo.com/redes-neuronales-o-el-arte-de-imitar-el-cerebro-humano/

Aunque la red neuronal simple supuso un gran avance en el aprendizaje estadístico, sólo se podía

usar para el procesamiento de patrones sencillos. Sin embargo, la investigación demostró que se podían combinar múltiples capas intermedias ocultas y que al hacerlo la red era más efectiva a la hora de de modelar patrones muchos más complejos que con el perceptrón simple. De hecho se considera a la red neuronal multicapa como un aproximador universal para cualquier función [15]. A esta adición de múltiples capas para modelar problemas más complejos se la considera el inicio de lo conocido como *deep learning*.

La estructura de un perceptrón multicapa consta de varias capas ocultas. Cada neurona está conectada a todas las neuronas de la capa anterior y a las de la capa posterior. Las neuronas que forman parte de una misma capa suelen emplear la misma función de activación. A base de combinar múltiple capas y funciones de activación no lineal, los modelos de red neuronal multicapa pueden aproximar cualquier función aunque existen limitaciones como el coste computacional.

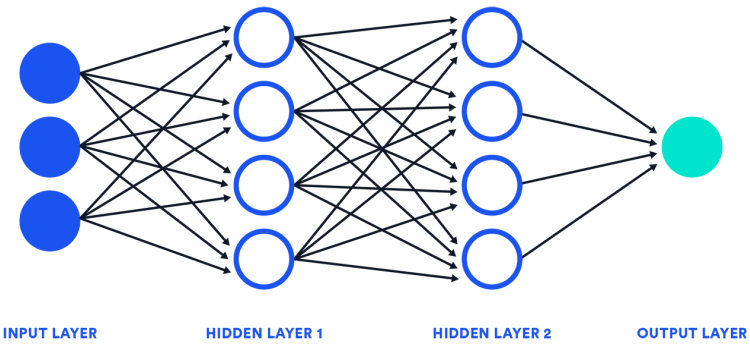

Figura 2.4: Representación de una red neuronal multicapa. Fuente: Banco de imágenes sin copyright https://www.pngegg.com/

2.3.2. Función de activación

Las funciones de activación, como anteriormente explicamos, modulan la forma en que la información se propaga de una capa a la siguiente. Mediante estas funciones se consigue transformar un valor determinado de entrada a otro valor transformado distinto del de entrada. Las funciones de activación pueden ser tanto lineales, como no lineales, lo cual permite que las redes neuronales puedan modelar relaciones no lineales.

La mayoría de las funciones de activación suelen ser sencillas y convertir el valor insertado a la neurona salidas que toman valores en los rangos (0,1) o (-1,1). Cuando el valor de salida de la neurona es 0 se dice que está inactiva debido a que no transmite ninguna información a la siguiente capa. Las funciones de activación más empleadas son:

- **Rectified linear unit (ReLU)**: Sólo si el input está por encima de cero. Mientras el valor de entrada está por debajo de cero, el valor de salida es cero, pero cuando es superior, el valor de salida aumenta de forma lineal con el de entrada.

$$\text{ReLU}(x) = \text{máx}(x, 0)$$

- **Softmax**: Transforma las salidas a una representación en forma de probabilidades, de tal manera

que el sumatorio de todas las probabilidades de las salidas de 1.

$$\text{Softmax}(z) = \frac{e^z}{\sum_{k=1}^{K} e^{z_k}}$$

- **Sigmoide**: La función sigmoide transforma valores en el rango de $(-\infty, +\infty)$ a valores en el rango (0, 1).

$$\text{sigmoid}(x) = \frac{1}{1 + \exp(-x)}$$

- **Tangente hiperbólica (Tanh)**: Se comporta de forma similar a la función sigmoide, pero su salida está acotada en el rango (-1, 1).

$$\tanh(x) = \frac{1 - \exp(-2x)}{1 + \exp(-2x)}$$

2.3.3. Función de coste

La función de coste (**L**), también llamada función de pérdida (Loss function en inglés) mide la desviación entre el valor real de una observación y el valor predicho por la red como sucede en otros modelos de aprendizaje estadístico y como hemos tratado en el prefacio. No es difícil pues llegar a la conclusión de que lo mejor para nuestro modelo es que el valor de esta función se acerque a cero lo máximo posible. Dependiendo del tipo de problema, regresión o clasificación, se suelen usar el error cuadrático medio y absoluto o la función log Loss respectivamente.

- **Error cuadrático medio**: MSE por sus siglas en inglés, es la más usada para problemas de regresión. Se calcula para una observación i como la diferencia al cuadrado entre el valor predicho $\hat{y}$ y el valor real y:

$$L_{(i)}(\mathbf{w}, \epsilon) = \left(\hat{y}_{(i)} - y_{(i)}\right)^2$$

  La función se escribe cómo $L(w, \epsilon)$ para expresar que depende de los pesos y del sesgo del modelo. Para calcular el valor que comete el conjunto de datos se promedia el error del conjunto de observaciones de la siguiente forma:

$$L(\mathbf{w}, \epsilon) = \frac{1}{n} \sum_{i=1}^{n} (\hat{y}_i - y_i)^2$$

- **Error medio absoluto**: MAE, por sus siglas en inglés, consiste en promediar el error absoluto de todas las predicciones del modelo. Es más robusto frente a los outliers que el MSE y se formula de la siguiente forma:

$$L(\mathbf{w}, b) = \frac{1}{n} \sum_{i=1}^{n} |\hat{y}_i - y_i|$$

- **Log loss**: Es la función de coste utilizada para problemas de clasificación. Cuando la clasificación es de tipo binaria, donde la variable respuesta es 1 o 0, y $p = Pr(y = 1)$, la función de coste log loss se define como:

$$L_{\log}(y, p) = -\log \Pr(y|p) = -(y \log(p) + (1 - y) \log(1 - p))$$

  Para problemas de clasificación con más de dos clases, esta fórmula se generaliza a:

$$L_{\log}(Y,P) = -\log \Pr(Y|P) = -\frac{1}{N}\sum_{i=0}^{N-1}\sum_{k=0}^{K-1} y_{i,k}\log p_{i,k}$$

Minimizar la función log Loss en redes neuronales de clasificación multiclase implicaría intuitivamente que la probabilidad para la clase correcta tiende a 1 mientras que a las clases incorrectas tenderían a 0.

2.3.4. Entrenamiento e hiperparámetros.

El **entrenamiento de una red neuronal** seguiría los siguientes pasos que a grandes rasgos consiste en ajustar el valor de los pesos y el sesgo de tal forma que se minimice el error para identificar los predictores que tienen mayor influencia y su relación con la variable respuesta:

- 1) Iniciar la red con valores aleatorios de pesos y sesgo.
- 2) Para cada observación de entrenamiento, calcular el error que comete la red al hacer su predicción. Promediar los errores de todas las observaciones.
- 3) Identificar la responsabilidad que ha tenido cada peso y sesgo en el error de las predicciones.
- 4) Modificar ligeramente los pesos y sesgos de la red (proporcionalmente a su responsabilidad en el error) en la dirección correcta para que se reduzca el error.
- 5) Repetir los pasos 2, 3, 4 y 5 hasta que se minimice lo suficiente la función de coste.

En el pseudo algoritmo anterior intervienen una serie de métodos matemáticos que enumeraremos y que sin los cuales la optimización de la red sería imposible de alcanzar:

- **Backpropagation**: La propagación hacia atrás es el algoritmo que permite identificar y cuantificar la influencia que tiene cada peso y sesgo en las predicciones finales de la red. Este algoritmo usa la regla de la cadena se calcula el gradiente (derivadas parciales de la función) lo cual permite ver que pesos o sesgos influyen más en el error de la red.
- **Descenso de gradiente**: Este algoritmo de optimización permite, que una vez identificados mediante backpropagation, el descenso de gradiente permite ir actualizando los pesos y sesgo del modelo para reducir su error. Debido al alto coste computacional que tiene el calcular el error para todas las observaciones de entrenamiento se utiliza una modificación llamada método de descenso de gradiente estocástico. Esta variante consiste en dividir las observaciones en lotes , *batch* en inglés, y actualizar los parámetros de forma progresiva con cada uno. Cabe destacar que cada vez que se procesan los todos *batch* de la red se conoce como época.
- **Preprocesado**: El preprocesado constituye las transformaciones previas que deben aplicarse sobre las variables del modelo. Destaca la binarización, utilizada sobre variables categóricas, para transformar variables multinivel en variables *dummy* de forma que por ejemplo una variable con 4 niveles se transformaría en 4 variables de tipo binario informando de la presencia o no del que informa cada una. Debemos destacar también, en este caso en el campo de las variables numéricas, el centrado y la estandarización para evitar diferencias entre escalas de la variables. El centrado consiste en en restarle a cada valor la media de la variable predictora a la que pertenece. La estandarización, por su parte, consiste en consiste en transformar los datos de forma que todos los predictores estén aproximadamente en la misma escala.

Por último debemos hablar de los principales hiperparámetros que intervienen en la construcción de redes neuronales en las librerías de R o Python que podríamos tener que ajustar en el momento de implementación:

- **Número y tamaño de capas**: Mientras que el tamaño de las capas de entrada y salida son parámetros fáciles de definir, ya que el número de neuronas en estas capas se define en base al número de predictores y al número de salidas (una en problemas de regresión y tantas como clases en problemas de optimización) la tarea complicada en este caso sería definir el número y el tamaño de las capas intermedias. Es una tarea complicada ya que a mayor número de capas, mayor número de relaciones y complejidad de las mismas, pero también mayor coste computacional y por tanto mayor tiempo de entrenamiento.

- **Learning rate**: La tasa de aprendizaje del modelo se refiere a cómo de bruscos pueden ser los cambios en los parámetros del modelo a medida que se optimiza. Es el parámetro más complicado de establecer ya que si es muy grande el algoritmo puede ir saltando de una región a otra del problema de optimización sin llegar al mínimo nunca, por el contrario si es muy pequeño el tiempo de entrenamiento se puede prolongar y no llegar a completarse nunca. Generalmente se utiliza la tasa más pequeña posible en relación al tiempo de entrenamiento y no se utiliza un valor constante, sino que se suelen usar valores mayores al principio y menores al final.

- **Algoritmo de optimización**: Los algoritmos antes mencionados (descenso de gradiente y el descenso de gradiente estocástico) presentan problemas a medida que se aumentan el tamaño de las redes (tanto en número como en capas), se pueden dar regiones en las que el gradiente es muy próximo a cero y el algoritmo se estanque. Para ello se han desarrollado modificaciones del algoritmo de descenso de gradiente que modifican la tasa de aprendizaje en función del gradiente y subgradiente adaptándolo (aumentando o disminuyendo) en función de la región del espacio en la que se encuentre. Destacamos el algoritmo *l-bfgs*[18] utilizado para conjuntos de datos pequeños y *adam*[16] para conjuntos de datos grandes.

- **Regularización**: Los métodos de regularización tratan de prevenir el sobreajuste y por tanto la incapacidad de hacer buenas predicciones del modelo. Destacamos las regularizaciones L1, L2 y el Dropout. Las regularizaciones L1 y L2, conocidas también como weight decay consisten en evitar que los pesos tomen valores excesivamente elevados evitando que ciertas neuronas tomen un peso exacerbado en el modelo. el Dropout [21] en cambio consiste que en cada iteración del entrenamiento, se ponen a cero los pesos de una fracción pequeña de neuronas por capa.

2.4. Support Vector Machines (SVM)

Las **máquinas de vector de soporte** o SVM, por sus siglas en inglés, son un tipo de modelos de aprendizaje estadístico que se desarrollaron inicialmente para solventar problemas de clasificación mediante la separación de los conjuntos de datos pertenecientes a las distintas categorías con hiperplanos. Inicialmente se desarrollaron para clasificación binaria, pero se han extendido a clasificación con más de dos categorías, regresión e incluso detección de outliers. Para entender cómo opera el modelo de SVM necesitamos comprender qué son los hiperplanos, los clasificadores de máximo margen y los clasificadores con vectores de soporte.

2.4.1. Hiperplano

El hiperplano se define, dado un espacio $p-$dimensional, cómo un subespacio plano y afín de dimensiones $p-1$. Por ejemplo para un espacio de dos dimensiones el hiperplano sería un recta.

Matemáticamente un hiperplano para un espacio de $p-$dimensiones se define con la siguiente ecuación:

$$\beta_0 + \beta_1 x_1 + \beta_2 x_2 + ... + \beta_p x_p = 0 \tag{2.23}$$

donde para todos los parámetros β_p y x_p se cumple la ecuación anterior si pertenecen al hiperplano.

En caso de que $\boldsymbol{x} = (x_1, \ldots, x_p)$ no satisfaga la ecuación planteada, significará que $\boldsymbol{x}$ cae a un lado u otro del hiperplano. Con lo cual se podría probar que el hiperplano de dimensión $p-1$ divide el espacio $p-$dimensional en dos mitades. Para saber en cual de las dos mitades cae $\boldsymbol{x}$ sólo habría que determinar si:

$$\beta_0 + \beta_1 x_1 + \beta_2 x_2 + ... + \beta_p x_p > 0$$

$$o$$

$$\beta_0 + \beta_1 x_1 + \beta_2 x_2 + ... + \beta_p x_p < 0$$

2.4.2. Clasificadores de máximo margen

Los clasificadores de máximo margen tratan de usar un hiperplano para clasificar los datos en dos categorías. Además, precisan de que exista una frontera lineal que separe los datos de entrenamiento de sendas categorías. Si tomamos la ecuación del hiperplano (2.23) clasificar cada $\boldsymbol{x_p}$ en cada una de las categorías que separa es tan fácil como ver el signo que toma la ecuación en cada valor de $\boldsymbol{x}$:

$$f(\mathbf{x}) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_p x_p \tag{2.24}$$

El valor de $f(\mathbf{x})$ además informa de la distancia del dato al hiperplano que separa las categorías. Dado que si existe un hiperplano que separa de forma perfecta un conjunto de datos, está probado que existen infinitos hiperplanos que también lo hacen.

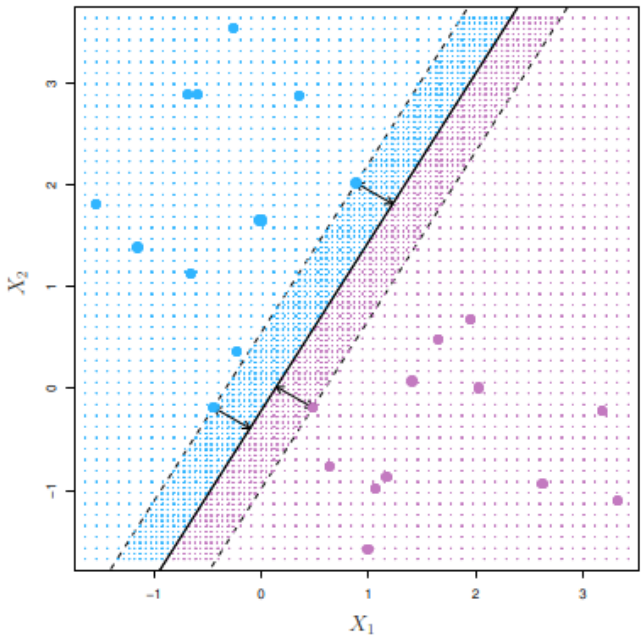

Figura 2.5: Representación de un hiperplano clasificador de máximo margen y sus vectores de soporte. Fuente: James, Gareth, et al., 2013. [17]

El método de clasificación de máximo margen se basa en seleccionar, de todos los hiperplanos que separan los datos en dos categorías, aquel hiperplano que tiene el mayor margen. Entendemos margen

como la distancia mínima de los datos de entrenamiento al hiperplano. Por consiguiente el clasificador de máximo margen será aquel hiperplano que tenga la distancia mínima de los datos de entrenamiento al hiperplano mayor.

Las observaciones ($\boldsymbol{x}$) cuya distancia al hiperplano corresponde al margen son conocidas como vectores de soporte ya que son vectores en un espacio $p-$dimensional y definen al hiperplano con mayor margen (figura 2.5).

Pero los clasificadores de máximo margen a nivel práctico carecen de interés debido a que son contadas las ocasiones en las que las clases a las que están sujetos los datos son perfecta y linealmente separables. Y aún cumpliéndose a nivel práctico estas improbables condiciones se presentan otros inconvenientes como la escasa robustez, por ejemplo una nueva observación introducida podría romper la separación perfecta necesaria para separar las clases. Otro inconveniente destacable es que el hiperplano se ajusta perfectamente a las observaciones de entrenamiento lo cual lleva casi inevitablemente al sobreajuste.

2.4.3. Clasificadores de margen débil

En el contexto de los inconvenientes mencionados para los clasificadores de máximo margen, se desarrolla un método que trata de solventar la problemática de que sea necesario un hiperplano que separe de forma perfecta los datos por categoría. Esta forma de enfocar el problema pasa por admitir que ciertos datos del conjunto de entrenamiento se van a clasificar de forma errónea, es decir que ambas categorías no estarán perfectamente separadas.

La formulación matemática del problema vendría sería:

$$\max_{\beta_0,\beta_1,\ldots,\beta_p,\epsilon_1,\ldots,\epsilon_n} M \tag{2.25}$$

sujeto a:

$$\sum_{j=1}^{p} \beta_j^2 = 1$$

$$y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \ldots + \beta_p x_{ip}) \geq M(1 - \epsilon_i) \quad \forall i$$

$$\sum_{i=1}^{n} \epsilon_i \leq C$$

$$\epsilon_i \geq 0 \quad \forall i$$

tenemos pues un problema de optimización convexa sujeto a una serie de restricciones donde $\beta_0, \beta_1, \ldots, \beta_p$ son los parámetros del hiperplano, $\epsilon_1, \ldots, \epsilon_n$ son variables de holgura y el parámetro C, de especial relevancia, controla el número y severidad de las violaciones del margen (y del hiperplano) que se toleran en el proceso de ajuste. Si $C = 0$ estaríamos ante el caso particular de un clasificador de máximo margen (si y sólo si las clases son perfectamente separables). A medida que el parámetro C se relaja, o se aproxima a ∞ , menos se penalizan los errores y más observaciones de entrenamiento podrían ser clasificadas incorrectamente durante el entrenamiento. Cuando C es grande el margen se expande y caen más observaciones en el pasando así a ser vectores de soporte. Ocurre al contrario para valores de C pequeños, el margen se contrae reduciendo el número de vectores que definen el hiperplano dando como resultado un clasificador con menor sesgo pero mayor varianza. El parámetro C se suele obtener por validación cruzada.

Se debe destacar de este tipo de clasificador que al ser definido por los vectores que caen en el margen y no tener en cuenta datos al muy alejados es muy robusto frente a datos anómalos. Una vez más este método presenta limitaciones, la más destacada es su utilización depende de que el límite de separación entre las clases a las que se adhieren los datos sea lineal.

2.4.4. Máquinas de soporte vectorial

En el contexto en el que los grupos no son linealmente separables (Figura 2.6 izquierda) la utilización de las técnicas anteriormente mencionadas no es viable. Sin embargo se desarrolló una estrategia que soluciona este problema: expandir las dimensiones del espacio original en el que se encuentran los datos hasta una dimensión mayor en la que estos datos sean linealmente clasificables (Figura 2.6 derecha) . Este es el fundamento de las máquinas de soporte vectorial, que implementando de partida una solución similar a la de los clasificadores de margen débil añaden el aumento de la dimensión de los datos.

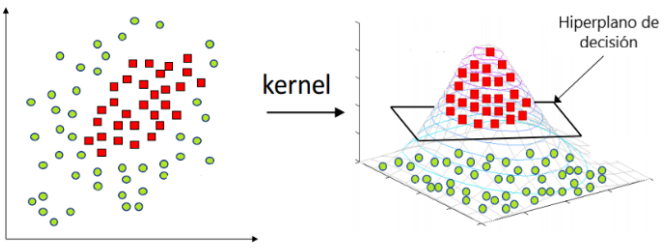

Figura 2.6: Representación de cómo el Kernel permite al aumentar la dimensión de los datos clasificarlos. Fuente: medium.com/@apurvjain37/support-vector-machine-s-v-m-classifiers-and-kernels-9e13176c9396

Metodología

Para explicar las operaciones que permiten al clasificador de soporte vectorial convertirse en una máquina de soporte vectorial, debemos presentar un pequeño detalle. El clasificador de soporte vectorial se puede expresar como una combinación lineal de productos escalares:

$$f(\mathbf{x}) = \beta_0 + \sum_{i \in S}^{n} \alpha_i \mathbf{x}^\top \mathbf{x}_i \tag{2.26}$$

donde $\mathbf{x}^\top \mathbf{x}_i$ del vector $\mathbf{x}$ del dato a clasificar y el vector $\mathbf{x_i}$ del $i-$ésimo dato de entrenamiento. Además debemos de tener en cuenta que finalmente sólo se van a tener en cuenta los índices que formen parte del soporte del hiperplano ($i \in S$). Dada esta forma de expresar los datos podemos mostrar cómo las máquinas de soporte vectorial transforman la dimensión de los datos. Para ello contamos con unas funciones denominadas Kernel (K) que lo que hacen es sustituir el producto vectorial original $\mathbf{x}^\top \mathbf{x}_i$ por una función Kernel de los datos que devuelve este producto realizado en una dimensión superior a la original. Al hacer esta sustitución conocida como **truco del Kernel** obtendremos los vectores de soporte y el hiperplano que separa los datos en la dimensión correspondiente a la transformación del Kernel.

Los Kernels más utilizados son:

- **Kernel lineal** No se cambia la dimensión de los datos, por lo que que el clasificador pasa a ser un clasificador de vector de soporte:

$$K(\mathbf{x}, \mathbf{x'}) = \mathbf{x}^\top \cdot \mathbf{x'} \tag{2.27}$$

- **Kernel polinómico** Los parámetros c y d controlan el comportamiento de la función Kernel, al aumentar el parámetro d se generan límites de decisión no lineales. Para valores $d > 5$ este Kernel empieza a dar problema de sobreajuste en el modelo:

$$K(\mathbf{x}, \mathbf{x'}) = (\mathbf{x}^\top \cdot \mathbf{x'} + c)^d \tag{2.28}$$

- **Kernel Gausiano o radial** El valor de γ controla el comportamiento del Kernel de la forma de que valores pequeños adoptan un comportamiento similar al Kernel lineal y para valores mayores la flexibilidad aumenta en el modelo final:

$$K(\mathbf{x}, \mathbf{x'}) = \exp(-\gamma||\mathbf{x} - \mathbf{x'}||^2) \tag{2.29}$$

Generalización y otras aplicaciones

El concepto sobre el que se construyen las máquinas de soporte vectorial no es generalizable de forma directa para K clases. Por el contrario, se han desarrollado distintos métodos que tratan de generalizar el modelo a k-clases entre las que destacan:

- **Uno contra uno** Cuando tenemos un escenario en el que se requiere clasificar datos en $K > 2$ clases, esta estrategia consiste en generar $K(K-1)/2$ clasificadores y comparar todos los posibles pares de clases. La observación tras haber pasado por los distintos clasificadores se asigna a la clase en la que ha sido asignada con mayor frecuencia. Este método tiene el problema de que se vuelve costoso computacionalmente a medida que aumenta el número de clases.

- **Uno contra todos** A diferencia del método anterior, en este se ajusta un modelo para cada K clase que considera las demás categorías $K - 1$ como una única. Una nueva observación es clasificada como perteneciente a la clase en la que mayor distancia a la frontera entre categorías se presente.

- **DAGSVM** De las siglas en inglés *Directed Acyclic Graph SVM* es una forma optimizada del método uno contra uno pero reduciendo las comparaciones necesaria mediante el empleo de los grafos acíclicos direccionados (DAG). Consiste en no realizar nuevas comparaciones cuando una observación ya se ha asignado a una clase concreta, reduciendo así las comparaciones y el coste computacional.

Aunque las máquinas de soporte vectorial son principalmente utilizadas para clasificación, ha habido trabajos que adaptaron la metodología al ajuste de modelos de regresión, de los cuales destaca el de Drucker, Harris, et al. (1996)[3].

Capítulo 3

Ensamblado y Bootstrap

Independientemente del los modelos de aprendizaje estadístico que nos planteemos utilizar, ya sea regresión o clasificación, la elección del modelo concreto a utilizar en cada caso es muy importante. En todo problema de aprendizaje estadístico tratamos de reducir el sesgo y la varianza del mismo. Para que un modelo sea flexible y el sesgo se reduzca, tratamos de incorporar el mayor número de variables que tengan un peso significativo, sin embargo, la incorporación de variables al modelo aumenta la varianza del mismo, implicando esto último el sobreajuste. Este equilibrio entre varianza y sesgo se debe tener en cuenta en todos los problemas para tratar de obtener un modelo robusto. Una ventaja que introducen los modelos de ensamblado es que permiten equilibrar este problema si somos coherentes en su aplicación. Por ejemplo, en caso de que usemos una serie de modelos de base con una alta varianza y un sesgo bajo, debemos utilizar un método de ensamblado que entre sus características esté la tendencia a reducir la varianza en el modelo de aprendizaje fuerte como es el caso del Bagging del que hablaremos en los siguientes apartados.

Los modelos de ensamblado explicados de forma sencilla buscan acoplar, o ensamblar, una serie de modelos denominados modelos de aprendizaje débil, o modelos de base, para tratar alcanzar un modelo en el que se consiga un equilibrio entre sesgo y varianza óptimo. Este modelo conseguido a partir del ensamblado de múltiples modelos débiles se denomina modelo de aprendizaje fuerte. Para conseguir los modelos de aprendizaje fuerte se pueden usar diferentes técnicas, una forma es usar un único modelo de base ajustado de distintas formas (ensamblado homogéneo). Otra forma consiste en usar como modelos de base distintos modelos de aprendizaje de características distintas que combinados dan lugar un ensamblado heterogéneo.

Los modelos de ensamblado a los que dedicaremos las siguientes secciones serán el **Bagging** el cual resulta de especial interés, ya que combina el ensamblado y el Bootstrap, el **Boosting** el cual realiza una estrategia en la que cada modelo depende de los anteriores y el **Stacking** crea un modelo de predicción basado en las predicciones de los modelos de base.

3.1. Bagging

El nombre de Bagging propuesto por Breiman, L. (1996)[1] proviene de la amalgama de términos en inglés Bootstrap Aggregation y es un método de ensamblado que se centra en tratar de reducir la varianza de los modelos de base que lo conforman. Estos modelos de base podrían ser tanto de regresión como de clasificación que presenten una elevada varianza. Previamente a explicar cómo funciona este método de ensamblado tocaremos las principales características del método utilizado para la obtención de diferentes modelos, el Bootstrap.

3.1.1. Bootstrap

El Bootstrap es una técnica estadística de remuestreo que fue propuesta por Efron (1979)[8] . Esta técnica trata de solucionar el problema de no tener disponibilidad de una gran cantidad de muestras de la población a la hora de realizar inferencia estadística. En caso de tener una muestra representativa de la población los valores para la variable aleatoria que componga esta muestra deberían aparecer en igual proporción que en la población original. El método Bootstrap utiliza esta muestra representativa de la población, de tamaño n, para obtener un número elevado de remuestras de igual tamaño, o muestras Bootstrap, pero usando muestreo con reposición. Al usar el muestreo con reposición se consiguen remuestras que proviniendo de la muestra original no van a contener los mismo datos. Si la muestra es representativa, la distribución del estadístico que se desee calcular con estas remuestras será semejante a la distribución muestral del mismo. El Bootstrap hace uso del método de simulación Monte Carlo y se pueden fijar el número remuestras que se desean obtener.

El Bootstrap puede ser usado como método de validación de un modelo de aprendizaje estadístico. Se usa para comparar modelos debido a que presenta una menor varianza respecto a otros métodos como podrían ser la los distintos tipos de validación cruzada. Para llevar a cabo este tipo de validación se seguiría el proceso similar al descrito en Schomaker & Heumann (2014)[20].

- 1) Obtener un número elevado B de remuestras a partir de la muestra original usando el proceso descrito anteriormente.
- 2) Ajustar un modelo para cada remuestra.
- 3) Calcular el error del modelo en la remuestra y su error de validación utilizando los datos que se quedan fuera del remuestreo (aproximadamente el 63,2 % en cada remuestreo) para testear el modelo.
- 4) Repetir los pasos 1 a 3 tantas veces como muestras tengamos y obtener el promedio del error de validación que tendrá debido a las propiedades del Bootstrap menor varianza que el que se obtuviese a partir de la muestra original de datos.
- 5) Finalmente se ajusta el modelo con los datos de entrenamiento y se comparan con los obtenidos usando ajustando los datos remuestreados.

La forma en la que se utiliza el Bootstrap es similar aunque con pequeñas variaciones dependiendo del contexto. En el siguiente apartado resumimos cómo opera dentro del Bagging y el porqué de la importancia del método.

3.1.2. Metología del Bagging

El Bootstrap en una metodología de ensamblado que busca el ajuste de un modelo de aprendizaje estadísco en múltiples ocasiones utilizando diferentes muestras. El modelo débil a ajustar debe ser un modelo que al ser ajustado posea persé una gran variabilidad ya que el Bagging es un modelo de ensamblado destinado esencialmente a reducir la variabilidad. Un ejemplo de modelo que sería susceptible de ser tomado como base para el Bagging sería un árbol de decisión con la particularidad de no haber aplicado técnicas de reducción de la varianza previamente como el podado, o *prunning*, en el caso del árbol de decisión.

El Bagging opera de la siguiente forma:

- 1) Se genera a partir del set de entrenamiento S un número L de remuestras Bootstrap S' de tamaño B:

$$S'_1 = \{z_1, \ldots, z_B\}, \ldots, S'_L = \{z_1, \ldots, z_B\}$$

- 2) Se ajusta un modelo $w(*)$ para cada remuestra. Una ventaja que tiene el Bootstrap es que este proceso se puede paralelizar al ser los ajustes independientes unos de otros:

$$w_1(*), \ldots, w_L(*)$$

- 3) Para cada predictor(es) obtener la respuesta evaluando cada modelo y promediando las predicciones obtenidas en el caso de la regresión, o la opción más votada (mayoría simple) en el caso de la clasificación.

$$S_L(*) = \frac{1}{L}\sum_{l=1}^{L} w_l(*), \quad \text{para el caso de la regresión}$$

$$S_L(*) = \arg\max_k [card(l|w(*) = k)], \quad \text{para el caso de la clasificación}$$

donde k es la clase más votada.

El Bagging, como apuntábamos al principio, es un modelo que permite llegar a una reducción de la varianza en el modelo final respecto de los modelos de base. Esto es cierto, siempre y cuando los modelos que se ensamblan no estén altamente correlacionados. De estar muy correlacionados los modelos de base utilizados la reducción de varianza es mínima, y por lo tanto el modelo resultante no aporta nada respecto a los de base. Para superar esta limitación, en el caso de los modelos de Bagging con árboles de decisión como modelos de base, surgen los bosques aleatorios o Random Forest que comentamos en el siguiente apartado.

3.1.3. Random Forest

El Random Forest es un algoritmo que parte de la idea del Bagging, entrenando una serie de árboles de decisión *profundos* (del inglés *deep trees* que significa que tienen un desarrollo en cuanto a ramas y profundidad de las mismas más elevada). Incluye una modificación para reducir la correlación entre los árboles generados en el proceso de ajuste del modelo de ensamblado. El hecho de que entre los predictores del modelo ($p = x_1, \ldots, x_n$) exista uno o dos que tengan un peso muy elevado a la hora de ajustar el modelo, y otros pocos que tengan un peso más moderado, resulta en que a la hora de ajustar los distintos modelos de base los predictores con más peso siempre estarán presentes y con gran influencia en todos los modelos. Esto provoca que exista una alta correlación entre ellos.

Para reducir la correlación entre los modelos de base y evitar que el algoritmo de agregación no suponga ninguna mejora respecto a los modelos de base Random Forest elimina la posibilidad de que las variables predictoras con más peso estén presentes en todos los modelos dominándolos de una forma bastante sencilla. Para cada modelo de base hace una selección aleatoria de m predictores, de esta forma un predictor entre los que se incluyen los que tienen más peso no aparecerán en aproximadamente $\frac{p-m}{p}$ árboles de base. Por consiguiente los árboles generados mediante la selección aleatoria de los predictores serán más heterogéneos y reducirán la correlación entre ellos. Debemos tener en cuenta que el número de predictores seleccionados aleatoriamente debe ser siempre inferior al número total de predictores ($m < p$) por que si $m = p$ el modelo resultante sería un ensamblado Bagging sin ninguna particularidad reseñable.

Los resultados del Random Forest van a variar siempre en función del valor de m tomado.Para calcular el valor óptimo de m se puede recurrir a validación cruzada o a evaluar el error del modelo con las observaciones que no entran en la remuestra Bootstrap en cada modelo ajustado, lo que se denomina **Out of Bag Error**. Una ventaja que debe ser comentada es que debido a que en no todos los árboles de base están presentes los mismos predictores, el Random Forest es muy robusto frente a datos faltantes.

3.2. Boosting

Este método de ensamblado, al igual que el Bagging, trata de ajustar una serie de modelos débiles de base con el objetivo de crear un modelo de aprendizaje fuerte. A diferencia del método de ensamblado anterior, los modelos de base del Boosting no son independientes ni se pueden ajustar de forma paralela, sino que se ajustan de forma iterativa teniendo e cuenta los modelos de base ajustados previamente. Además este método de ensamblado trabaja con modelos de base con varianza reducida y sesgo elevado, siendo su objetivo fundamental el reducir este último en cada iteración. Por ejemplo en el caso de los árboles de decisión, elegiríamos árboles con pocas ramas y poca profundidad, es decir, con un sesgo alto y una varianza reducida.

Los distintos algoritmos que se encuadran en el Boosting se diferencian entre si en la manera de abordar el ensamblado de estos modelos de base en el proceso iterativo. Presentaremos por un lado el Adaptative Boosting (*AdaBoost*) y por otro el Gradient Boosting explicando como operan ambos. Los métodos de ensamblado Boosting manejan dos hiperparámetros cuyo peso en el modelo es de especial relevancia:

- **Número de modelos de base:** Como en el Boosting cada ajuste de un modelo de base representa una nueva iteración, el hiperparámetro que controla el número de modelos de aprendizaje débil controla a su vez el número de iteraciones del ensamblado. Si el número de iteraciones del método es muy alto el ensamblado podría tener problemas serios de sobreajuste.
- **Complejidad de los modelos de base:** Los modelos de base deben modelos de varianza reducida, es decir que producen una predicción un poco mejor que una elección al azar. Este hiperparámetro, si por ejemplo empleamos árboles de decisión como modelos de base podría controlar el número de nodos el árbol.
- **Learning rate o tasa de aprendizaje (λ):** Este hiperparámetro controla el peso que tiene cada modelo de base en el ensamblado. Se suelen utilizar valores pequeños de este hiperparámetro ($\lambda = 0{,}01$ o $\lambda = 0{,}001$) debido a que se ha probado que utilizar valores altos afecta negativamente a la eficacia el ensamblado. Generalmente se utiliza el método de validación cruzada para elegir el valor de λ.

3.2.1. Adaptative Boosting

El algoritmo *AdaBoost* [12] se plantea como una solución mediante ensamblado a un problema de clasificación binaria. Para este algoritmo se necesita establecer un modelo de base cuyo ajuste tenga como resultado un acierto ligeramente superior al azar (elevado sesgo y baja varianza), codificar la variable de respuesta cómo $Y \in \{-1, +1\}$ y establecer un peso inicial igual para todas las observaciones de entrenamiento.

El algoritmo sería definido como:

- 1) Se inicializan los pesos en las observaciones de tal forma que cada observación tenga el mismo peso que el resto:
$$w_i = \frac{1}{n}, i = 1, 2, ..., N$$
- 2) Repetir para $m = 1$ hasta M:

  2.1) Sean el número de iteraciones $M = m_1, \ldots, m_k$ se ajusta el modelos de base G_m utilizando las observaciones de entrenamiento con los pesos w_i.

  2.2) Se calcula el error del modelo G_{m_k} ajustado como:
$$r_m = \frac{\sum_{i=1}^{N} w_i I(y_i \neq G_m(x_i))}{\sum_{i=1}^{N} w_i}$$

2.3) Calcular el peso asignado al modelo de base Gm_k de la siguiente forma:

$$\alpha_m = \log(\frac{1 - r_m}{r_m})$$

2.4) Actualizar los pesos de las observaciones:

$$w_i = w_i \exp[\alpha_m I(y_i \neq G_m(x_i))], \quad i = 1, 2, ..., N$$

donde N es el número de observaciones del set de entrenamiento, M es el número de modelos base y por lo tanto de iteraciones, G_m modelos de base de la iteración m, w_i es el peso de la observación i y α_m es el peso del modelo de base m.

- 3) Se construye el modelo de aprendizaje fuerte $G(x)$ ensamblando los modelos de base anteriormente calculados y se calculan las prediciones como:

$$G(x) = sign[\sum_{m=1}^{M} \alpha_m G_m(x)]$$

donde *sign* es la función que obtiene el si la predicción pertenece a -1 o a $+1$, recordemos que esta variable en AdaBoost debe estar codificada como $Y \in \{-1, +1\}$.

3.2.2. Gradient Boosting

Gradient Boost fue propuesto por Friedman (2001)[13] es un método de ensamblado Boosting que puede ser aplicado en modelos para los que la función de coste, por ejemplo el RSS para la regresión, sea diferenciable. Esto implica que sea un excelente método y ampliamente usado debido a que puede ensamblar distintos modelos de base tanto para regresión como para clasificación.

El algoritmo de Gradient Boosting para arboles de decisión seguiría los siguientes pasos:

- 1) Se inicializa el modelo con un valor constante:

$$F_0(x) = \arg\min y \sum_{i=1}^{n} L(y_i, y)$$

- 2) Repetir para $m = 1$ hasta M:

2.1) Se obtienen los residuos del modelo:

$$r_{im} = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F_x = F_{m-1}(x)} \qquad \text{para} \quad i = 1, \ldots, n$$

2.2) Se ajusta un árbol para los residuos r_{im} dando lugar a las regiones terminales R_{jm} donde $j = 1, \ldots, J_m$.

2.3) Calcular: $y_{jm} = \arg\min y \sum_{x_i \in R_{jm}} L(y - i, Fm - 1(x_i) + y)$ para $j = 1, \ldots, J_m$.

2.4) Actualizar el modelo: $F_m(x) = F_{m-1}(x) + v \sum_{j=1}^{J_m} y_{jm} I$ para $x \in R_{jm}$

Mediante el algoritmo anterior se van construyendo de forma secuencial nuestros M modelos y se van ensamblando usando el descenso por gradiente que nos informa en el paso 2.1 en que signo y valor puede minimizarse la función de coste.

3.3. Stacking

Para finalizar el apartado referido a los métodos de ensamblado definiremos el **Stacking**, cuyo mayor punto de diferencia respecto a los métodos anteriores es el uso de modelos de base de distinto tipo para llegar al modelo final de aprendizaje fuerte. El uso de modelos heterogéneos a la hora de ajustar los modelos de aprendizaje débil, permite aprovechar las distintas ventajas que aportan cada uno de ellos por separado para ajustar finalmente un modelo fuerte, o metamodelo, en base a las predicciones de los anteriores. Por ejemplo en un problema de clasificación como modelos de base se podrían ajustar una máquina de soporte vectorial, una regresión logística u otro clasificador posteriormente que otro clasificador fuerte como una red neuronal tome los resultados de los clasificadores débiles y arroje las predicciones finales.

El algoritmo de Stacking procedería de la siguiente forma:

- 1) Dividimos el data set de entrenamiento en dos mitades $\{A, B\}$.
- 2) Hacemos una selección de los L modelos débiles que formarán los modelos de base.
- 3) Ajustamos los modelos de base con los datos de entrenamiento A.
- 4) Para cada uno de los modelos débiles ajustados en el paso 3 realizamos predicciones con los datos B.
- 5) Ajustamos el meta modelo final usando como datos de entrenamiento las predicciones realizadas por los modelos de base calculadas en el paso 4.

Uno de los problemas que presenta el Stacking es que divide las observaciones de entrenamiento en dos grupos, limitando las observaciones con las que se entrena el modelo final. Para solucionar esta desventaja existen diferentes técnicas que tratan de hacer posible la utilización de todas las observaciones de entrenamiento para ajustar el modelo fuerte. Una de ellas es el *k-fold cross-training* que consiste en ajustar cada modelo débil con los datos de las divisiones $k-1$ (siendo k el número de divisiones totales del conjunto de entrenamiento) y realizar las predicciones para ese modelo de base con las observaciones que no fueron usadas en el ajuste. Esto se repetiría en cada uno de los modelos de base permitiendo así la utilización del conjunto total para ajustar el metamodelo.

3.3.1. Stacking multinivel

El Stacking multinivel es una variante del Stacking en la que tras el ajuste de los L modelos de base, se ajusta otra capa de M metamodelos usando las predicciones de los primeros en lugar de ajustar directamente un metamodelo final. Tras ajustar esta segunda capa de modelos utilizaremos sus predicciones para ajustar el metamodelo final que tomará como predictores las predicciones los modelos de la capa anterior.

El uso de más de un nivel de ajuste de modelos débiles puede traer problemas por la falta de datos, debido a las divisiones en las que se debe cortar el set de entrenamiento. Esto se puede solucionar con técnicas como *k-fold cross-training*, descrito anteriormente, pero al utilizar esta técnica encarecería el coste computacional debido al numeroso número de modelos que habría que ajustar.

Capítulo 4

Aplicación de los modelos a problemas reales

Como final al presente trabajo trataremos de aplicar algunas de las técnicas de aprendizaje estadístico supervisado descritas en el presente trabajo. El objetivo principal, será el ilustrar de forma práctica a gran parte de las técnicas descritas en el presente trabajo. Para ello dispondremos de dos problemas, trataremos de implementar diversos modelos de aprendizaje estadístico centrados en el objetivo de comparar y hallar el mejor modelo que prediga de forma más precisa la variable respuesta. En el primero de ellos la variable a predecir será de tipo categórico, por lo que los modelos implementados serán de clasificación estadística en los que mediremos la precisión de cada uno. En el segundo de ellos realizaremos un procedimiento similar a el anterior con la diferencia de que la variable de respuesta será en este caso continua con lo que utilizaremos modelos centrados en la regresión y las métricas con las que evaluaremos la precisión variarán.

4.1. Librerías de R

Destacaremos la librería **Caret**[4] de R y algunas de las funciones empleadas para el tratamiento de los datos y el ajuste de los modelos en el presente capítulo:

Caret[4]: es una librería de R que se utiliza para el procesamiento de datos de aprendizaje automático y el análisis de clasificación y regresión. La librería proporciona una interfaz unificada para múltiples algoritmos de aprendizaje automático, lo que permite a los usuarios seleccionar y ajustar modelos de forma más eficiente. También incluye funciones para la evaluación de modelos, el preprocesamiento de datos y la generación de gráficos. El grueso de los análisis y construcción de de modelos del presente apartado se realizarán con las funciones que proporciona esta librería. Dentro de esta librería destacaremos algunas de las funciones más relevantes:

- trainControl: Esta función nos permitirá seleccionar el método de validación del modelo, en nuestros problemas elegiremos validación cruzada repetida.

- train: Es la función más importante de la librería ya que con ella ajustaremos los diferentes modelos. El argumento *method* permite seleccionar una amplia gama de modelos de aprendizaje estadístico para construir el algoritmo.

- Predict: Aunque esta función es del paquete *stats* será la que nos permita evaluar los modelos construidos sobre el conjunto de test.

- confusionMatrix: Es la función con la que se pueden obtener las principales medidas de interés para las predicciones de un modelo de clasificación. Crea una matriz de confusión como la descrita en el primer capítulo del trabajo y calcula de forma simultánea las medidas de interés.
- caretEnsemble: Esta función permite apilar varios modelos de base utilizando un modelo fuerte como ensamblador para construir un ensamblado tipo Stacking.

4.2. Aplicación

4.2.1. Problema 1

Para el primer problema que vamos a tratar utilizaremos los datos descargados del repositorio de UCI Machine Learning *"Heart Failure"* que consiste en un conjunto de datos de carácter médico que fueron usados en el artículo Chicco, Davide; Jurman, Giuseppe (2020)[6] para tratar de predecir la eventual muerte de un paciente en seguimiento en función de una serie de predictores. El conjunto consta de 13 variables, tanto cuantitativas como cualitativas, y un total de 299 observaciones. Las variables usadas fueron:

- **age:** Edad del paciente en años
- **anaemia:** disminución de glóbulos rojos o hemoglobina (booleana)
- **high blood pressure:** si el paciente tiene hipertensión (booleano)
- **creatinine phosphokinase (CPK):** nivel de la enzima CPK en la sangre (mcg/L)
- **diabetes:** si el paciente tiene diabetes (booleano)
- **ejection fraction:** porcentaje de sangre que sale del corazón en cada contracción (porcentaje
- **platelets:** plaquetas en la sangre (kiloplaquetas/mL)
- **sex:** mujer u hombre (binario)
- **serum creatinine:** nivel de creatinina sérica en la sangre (mg/dL)
- **serum sodium:** nivel de sodio sérico en la sangre (mEq/L)
- **smoking:** si el paciente fuma o no (booleano)
- **time:** período de seguimiento (días)
- **death event:** si el paciente falleció durante el período de seguimiento (booleano)

Procedimiento utilizado para preprocesar los datos y prepararlos:

- Se filtran los datos buscando valores ausentes y duplicados.
- Se codifican las variables cualitativas como factores añadiendo sus valores como etiqueta.
- Se elimina la variable time debido a que sus valores desiguales introducirían ruido en el modelo por su propia naturaleza.
- Se seleccionan las variables más relevantes para el modelo apoyándonos en el método de selección por filtrado implementado en el paquete Caret.

- **División de los datos para entrenamiento y test:** Tras haber realizado el preprocesado de los datos el data set consta de 298 observaciones y hemos conseguido reducir las variables predictoras a cuatro (age, ejection_fraction, serum_creatinine, serum_sodium). Nuestra variable respuesta será la muerte del paciente (DEATH_EVENT) y para proceder a ajustar los distintos modelos primeramente dividimos el conjunto de datos en dos partes. La primera que recogerá un 70 % de las observaciones se se destinará al set de entrenamiento con el cual se ajustarán los modelos. El 30 % restante de los datos se destinará a al set de test que será utilizado para evaluar la eficacia de los modelos a la hora de realizar predicciones fuera del conjunto de entrenamiento.

Una vez tenemos los datos separados en el conjunto de entrenamiento y de test pasamos a ajustar los distintos modelos, en este caso y con la ayuda de la librería Caret:

- **Regresión Logística**
- **Árbol de clasificación**
- **Red neuronal**
- **SVM con núcleo lineal**
- **Random Forest**
- **Gradient Boosting**
- **Stacking:** conformado por un árbol de clasificación, un SVM de núcleo lineal y una red neuronal como modelos de base ensamblados mediante un Random Forest.

Resultados

En el cuadro 4.1 podemos observar la comparativa para medidas de precisión para los distintos modelos ajustados. Estas medidas de precisión se calcularon usando una matriz de confusión con los datos resultado de evaluar los modelos sobre el set de datos de test. Para ello se usó la función **confusionMatrix** del paquete Caret[4]. Las medidas que se han tenido en cuenta son la **precisión**, la **sensibilidad**, la **especificidad** y la **precisión balanceada**.

Cuadro 4.1: Resultados sobre los datos de test para los modelos entrenados en el problema 1

Modelos	Precisión	Sensibilidad	Especificidad	Precisión balanceada
R. Logística	0.75	0.43	0.9	0.66
Arbol de Clasificación	0.72	0.82	0.5	0.66
NNET	0.76	0.92	0.43	0.67
SVM	0.77	0.92	0.46	0.69
Random Forest	0.86	0.97	0.64	0.8
Gradient Boosting	0.76	0.88	0.5	0.69
Stacking	0.74	0.87	0.46	0.67

Podemos observar los modelos obtienen unas puntuaciones de rendimiento muy parejas, siendo el Random Forest el modelo que destaca entre todos. En el gráfico 4.1. Cabe destacar que en un primer momento, el Random Forest fue el modelo que obtuvo las peores puntuaciones en la predicción. Esto puede explicarse, por que al usar un método de selección de variables por filtrado *sbf* del paquete Caret[4] se redujeron notablemente las variables predictoras. Si volvemos al apartado 3.1.3 veremos que una de las características de los modelos tipo Bagging, y en particular del Random Forest, debido es que tratan de ajustar modelos de base con un alto componente de varianza. Esto se debe a que son modelos de ensamblado que se comportan como reductores de esa varianza, por lo que necesitábamos que los árboles que componen el ensamblado tuviesen numerosos nodos y considerable profundidad, lo cual reduciendo los predictores a del modelo a cuatro era una tarea imposible. Por lo tanto, para el entrenamiento del Random Forest se recurrió al conjunto completo de variables predictoras para conseguir los modelos de base con la máxima varianza posible. Como consecuencia podemos observar que es el modelo que mayor puntuación obtiene en las predicciones.

Podemos observar en la figura 4.1 de forma gráfica y ordenada la puntuación de los diferentes modelos.

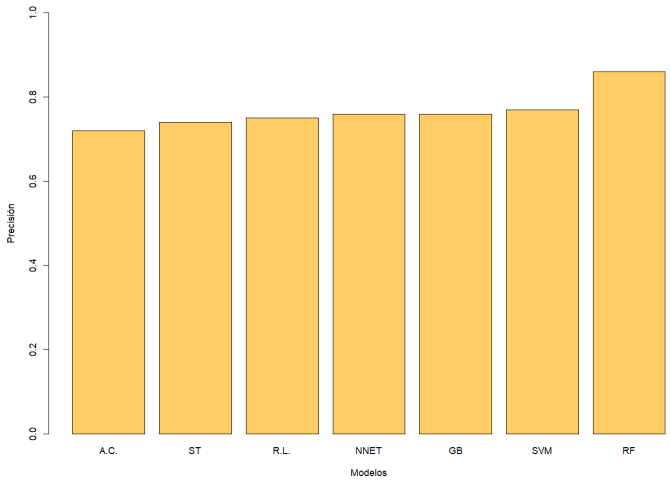

Figura 4.1: **Precisión evaluada con datos de test según el modelo utilizado.** Donde A.C. un es Árbol de Clasificación. ST es un modelo de Stacking conformado por un árbol de clasificación, un SVM de núcleo lineal y una red neuronal como modelos de base ensamblados mediante un Random Forest. RL es una Regresión Logística. NNET es una red neuronal. GB es un modelo de ensamblado de Gradient Boosting de tipo estocástico. RF es un Random Forest entrenado con el número máximo de predictores del modelo.

4.2.2. Problema 2

Para el segundo problema que vamos a tratar volvemos a acudir al repositorio de UCI Machine Learning, esta vez para descargarnos los datos de *Concrete Strength* que consiste en un conjunto de datos relativo a ingeniería de materiales que fueron usados en el artículo Yeh, I.-C. (1998))[23] para tratar de predecir la resistencia del hormigón (*concrete*) en función de una serie de predictores relativos a los componentes utilizados en la mezcla para fabricar hormigón y el tiempo pasado desde su fabricación. El conjunto de datos consta de 9 variables de tipo cuantitativo y un total de 1030 observaciones. Las variables usadas fueron:

- **cement:** componente utilizado para la construcción que se endurece a otros materiales para unirlos. (kg/m^3)
- **blast_f_r,:** Mezcla de óxidos metálicos y dióxido de silicio. (kg/m^3)
- **fly_ash:** producto de la combustión del carbón. Se compone de las partículas que se expulsan de las calderas de carbón junto con los gases de combustión. (kg/m^3)
- **Water:** Augua utiliza para formar la pasta espesa. (kg/m^3)
- **Superplasticizer:** componente utilizado en la fabricación de hormigón de alta resistencia. (kg/m^3)
- **coarse_agg:** trozos de rocas que se obtienen de los depósitos del suelo. (kg/m^3)
- **fine_agg:** trozos de rocas cuyo tamaño es menor de 4,75 mm. (kg/m^3)
- **age:** Momento en días desde su creación en que se evaluó la muestra. ($dias$)
- **concrete strength:** Unidad de medida de la resistencia del hormigón.(MPa)

Procedimiento utilizado para pre procesar los datos y prepararlos:

- Se filtran los datos buscando valores ausentes y duplicados.
- Se codifican las variables cualitativas como factores añadiendo sus valores como etiqueta.
- Se seleccionan las variables más relevantes para el modelo apoyándonos en el método de selección por filtrado implementado en el paquete Caret.
- **División de los datos para entrenamiento y test:** Tras haber realizado el preprocesado de los datos el data set se reduce a 1005 observaciones y hemos conseguido reducir las variables predictoras a seis (age, blast_f_r, coarse_agg, fine_agg, fly_ash, water). Nuestra variable respuesta será la resistencia del hormigón (concrete strength) la cual es continua por lo que deberemos ajustar múltiples modelos de tipo regresivo.Así mismo, como en el problema anterior, dividiremos la muestra en dos partes. La primera que recogerá un 70 % de las observaciones se se destinará al set de entrenamiento con el cual se ajustarán los modelos. El 30 % restante de los datos se destinará a al set de test que será utilizado para evaluar la eficacia de los modelos a la hora de realizar predicciones fuera del conjunto de entrenamiento.

Una vez tenemos los datos separados en el conjunto de entrenamiento y de test pasamos a ajustar los distintos modelos, en este caso y con la ayuda de la librería Caret:

- **Regresión Lineal Múltiple**
- **Árbol de regresión**
- **GAM con Spline**
- **Random Forest**
- **Gradient Boosting**
- **Stacking:** Conformado por los modelos anteriores como modelos de base exceptuando el Gradient Boosting y utilizando como modelo de ensamblado un modelo lineal generalizado.

Resultados

En el cuadro 4.2 podemos observar la comparativa para medidas de precisión para los distintos modelos ajustados. Para este problema hemos recurrido a las medidas de precisión pertinentes para medir el rendimiento de los distintos modelos ajustados en el problema. Una vez más, como en el problema anterior, las medidas se realizan con las predicciones del modelo construido con los datos de entrenamiento sobre los datos de test y se comparan con las observaciones reales. Para ello se han tenido en cuenta como evaluadores del rendimiento el **pseudo coeficiente de determinación** $\tilde{R}^2$, la raíz cuadrada del **error cuadrático medio** ($RMSE$) y el **error medio absoluto** (MAE).

Cuadro 4.2: Resultados sobre los datos de test para los modelos entrenados en el problema 2

Modelos	$\tilde{R}^2$	**RMSE**	**MAE**
R. Lineal Múltiple	0.55	10.83	8.65
Árbol de regresión	0.68	9.05	7.08
GAM con Spline	0.77	7.79	5.98
Random Forest	0.88	5.84	4.47
Stacking	0.89	5.53	4.14
Gradient Boosting	0.93	4.44	2.90

Podemos observar los modelos que obtienen mejores resultados son claramente los modelos de ensamblado estadístico rondando un $\tilde{R}^2$ de 0,90. En el gráfico de la figura 4.2 podemos observar de forma gráfica y ordenada según el coeficiente $\tilde{R}^2$ para cada modelo de menor a mayor precisión el mejor rendimiento señalado para los métodos de ensamblado. En este caso el mejor modelo, evaluado sobre las predicciones en el test sería el Gradient Boosting.

Al respecto de las puntuaciones obtenidas en la evaluación de los modelos sobre los datos de test, llama la atención el bajo rendimiento del modelo lineal múltiple. Por lo que nos disponemos a tratar de explicar el porqué de ello.

Primeramente creamos un gráfico de correlaciones entre las variables que se usan en el modelo (predictores y respuesta) utilizando la función *ggpairs* de la librería GGally [19]. En la figura 4.3 observamos la baja correlación que hay entre las variables predictoras del modelo respecto a la variable respuesta, lo cual puede empezar a darnos pistas sobre el bajo rendimiento del modelo. También podemos concluir, a la vista del gráfico, que no existen altas correlaciones entre los predictores del modelo aunque en ciertas ocasiones sean más altas entre ellos que con la propia variable respuesta, lo cual es reseñable.

Además de analizar la correlación entre las variables del modelo, analizamos la normalidad de los errores del modelo para comprobar si se cumple el supuesto de normalidad. A la vista del gráfico QQ-Plot presentado en 4.4 podemos descartar la ausencia de normalidad en los errores del modelo y el consiguiente cumplimiento del citado supuesto.

Por último y observando el gráfico contenido en la figura 4.5 que representa la dispersión de los residuos respecto a los valores ajustados del modelo podemos detectar uno de los principales problemas

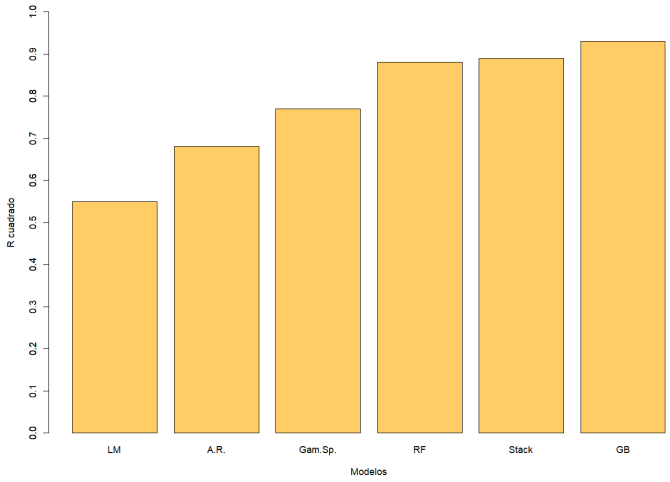

Figura 4.2: **Precisión evaluada con datos de test según el modelo utilizado.** LM es un modelo de regresión lineal múltiple. A.R. es un Árbol de Regresión. Gam Sp. es un modelo aditivo generalizado con Splines. Stack es un modelo de Stacking conformado, una regresión lineal múltiple, un GAM con Splines, un Árbol de Regresión y un Random Forest. GB es un modelo de ensamblado de Gradient Boosting de tipo estocástico. RF es un Random Forest entrenado con el número máximo de predictores del modelo.

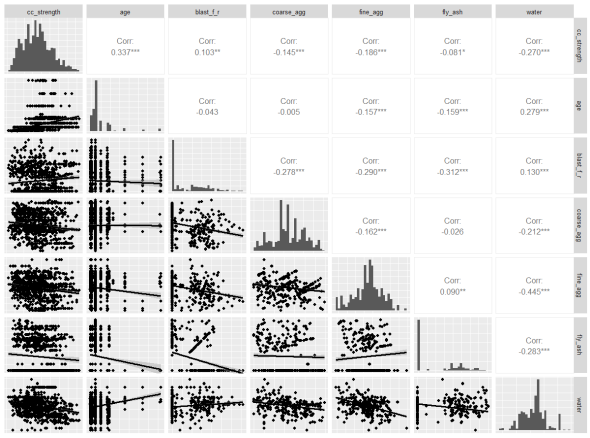

Figura 4.3: Gráfica de correlaciones entre las variables del problema 2.

de este modelo: no se cumple el supuesto de homocedasticidad explicado en el apartado 2.1.1.

Como consecuencia de los puntos explicados anteriormente, el modelo de regresión lineal múltiple ajustado no es válido y produce unas predicciones pésimas al probarlo en la práctica. Por esta razón sus medidas de evaluación en los datos de test han resultado tan malas.

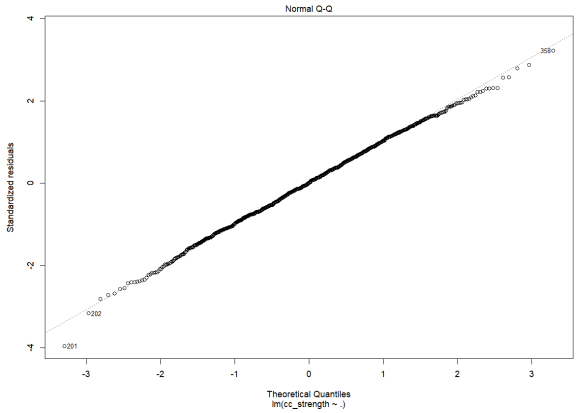

Figura 4.4: QQ-Plot de los errores del modelo lineal múltiple del problema 2.

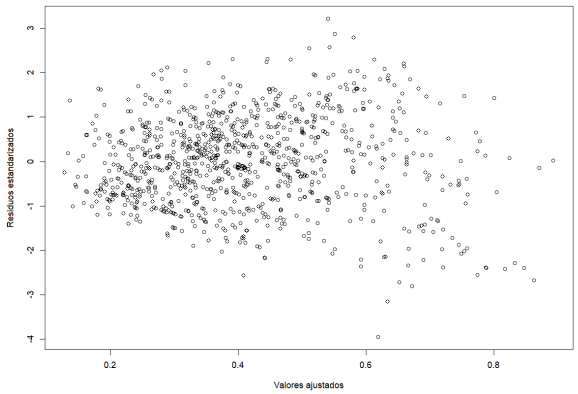

Figura 4.5: Gráfico que representa la dispersión de los errores del modelo lineal múltiple del problema 2 .

Capítulo 5

Conclusiones y reflexiones.

En el presente trabajo hemos revisado diferentes técnicas de aprendizaje estadístico supervisado y sus aplicaciones. Se han revisado tanto técnicas que inicialmente formaron parte de la estadística clásica como técnicas que surgieron bajo el paradigma del aprendizaje estadístico. En este último apartado hemos querido probar las diferentes técnicas estudiadas y observar su comportamiento en dos problemas reales. Aunque el segundo problema quizá se ha podido observar mejor cómo los métodos de ensamblado mejoran la capacidad del aprendizaje estadístico para aumentar su rendimiento, es posible que en el primer problema no se haya podido observar debido al tamaño reducido de datos.

Se ha tratado a la par, de estudiar cómo las técnicas de ensamblado junto con el Bootstrap permiten combinar las fortalezas de distintos modelos de aprendizaje estadístico con el fin de mejorar su rendimiento y obtener el modelo que mejor se adapte a extraer la máxima cantidad de información de valor de los datos que tratamos de medir. Hemos visto cómo a través de los métodos de ensamblado se puede equilibrar el sesgo y varianza de los modelos de aprendizaje estadístico. Cómo partiendo de un conjunto de modelos con un alto sesgo se pueden obtener muy buenas predicciones utilizando el Boosting o cómo partiendo de modelos con una elevada varianza se pueden conformar ensamblados equilibrados, como es el caso del Bagging.

Aunque el presente trabajo ha tenido la contrapartida de no profundizar demasiado en cada modelo tratado y se tuvieron que descartar algunos modelos que hubiera sido interesante tratar, en general ha sido una aproximación muy positiva al aprendizaje estadístico y una forma de explorar nuevas herramientas y reforzar viejas conocidas que me serán de utilidad en mi día a día.

Apéndice A

Código utilizado

```
################################################################################

################################################################################

#                                      Capítulo 4 - Aplicación a datos reales
                                        ###################################
#                                      Heart failure clinical records Data Set
################################################################################

################################################################################

################################################################################

######################################## P1 -- Clasificación
    ########################
#0 - Preanálisis:
getwd()
setwd("G:/Mi unidad/00_TFM_22/TFM/Definitivo_21_07/Análisis")

#Librerías
library(dplyr)
library(ggplot2)
library(readxl)
library(gmodels)
library(Hmisc)
library(ggthemes)
library(caret)
library(caretEnsemble)
library(kernlab)
library(gam)
#library(MASS) #Se invocará cuando sea necesaria, entra en conflicto la
    función select de dplyr.
library(lmtest)
library(GGally)

#1 - Cargamos los datos:

datos_r=read.csv('heart_failure_clinical_records_dataset.csv',header = TRUE,
                 sep = ",",
```

```
                quote = "\"",
                dec = "." )

#Tenemos 299 observaciones de 13 variables.

# Análisis Exploratorio:

#Variables:
#- age: Edad del paciente en años
#- anaemia: disminución de glóbulos rojos o hemoglobina (booleana)
#- high blood pressure: si el paciente tiene hipertensión (booleano)
#- creatinine phosphokinase (CPK): nivel de la enzima CPK en la sangre (mcg/L)
#- diabetes: si el paciente tiene diabetes (booleano)
#- ejection fraction: porcentaje de sangre que sale del corazón en cada
    contracción (porcentaje)
#- platelets: plaquetas en la sangre (kiloplaquetas/mL)
#- sex: mujer u hombre (binario)
#- serum creatinine: nivel de creatinina sérica en la sangre (mg/dL)
#- serum sodium: nivel de sodio sérico en la sangre (mEq/L)
#- smoking: si el paciente fuma o no (booleano)
#- time:   período de seguimiento (días)
#- [target] death event: si el paciente falleció durante el período de
    seguimiento (booleano)

glimpse(datos_r)

#Transformamos la variables categóricas en factores:

to_factor= c('anaemia','high_blood_pressure','diabetes','sex','smoking','
    DEATH_EVENT')

#lo hacemos con la función lapply

datos_r[to_factor] <- lapply(datos_r[to_factor], factor)

#añadimos etiquetas

# variable v1 is coded 1, 2 or 3
# we want to attach value labels 1=red, 2=blue, 3=green

datos_r$anaemia <- factor(datos_r$anaemia,
                    levels = c(0,1),
                    labels = c("no", "si"))

datos_r$diabetes <- factor(datos_r$diabetes,
                        levels = c(0,1),
                        labels = c("no", "si"))

datos_r$high_blood_pressure <- factor(datos_r$high_blood_pressure,
                        levels = c(0,1),
                        labels = c("no", "si"))

datos_r$sex <- factor(datos_r$sex,
                                    levels = c(0,1),
                                    labels = c("mujer ", "hombre"))

```

```r
datos_r$smoking <- factor(datos_r$smoking,
                                    levels = c(0,1),
                                    labels = c("no", "si"))

datos_r$DEATH_EVENT <- factor(datos_r$DEATH_EVENT,
                          levels = c(0,1),
                          labels = c("no", "si"))

#comprobamos
glimpse(datos_r)

summary(datos_r)

#Comprobamos si hay NA y duplicados


anyNA(datos_r)

nrow(datos_r[duplicated(datos_r), ])

#No hay NA en nuestros datos.

#hay datos duplicados asique procedemos a eliminarlos:

datos_r= distinct(datos_r)
#Descartamos los datos relativos al tiempo de seguimiento, ya que el tiempo durante el que se ha observado al paciente no influye en
# el resultado de muerte o supervivencia

#datos_r = datos_r[,-12]



#ver que datos están con varianza proxima a acero
datos_r %>% select(age ,anaemia, creatinine_phosphokinase, diabetes, ejection_fraction, high_blood_pressure,  platelets,serum_creatinine, serum_sodium, sex, smoking, DEATH_EVENT) %>% nearZeroVar(saveMetrics = TRUE)

#############################################################################
#Vemos la proporción de  variable respuesta para ver el porcentaje de acierto a superar (nivel basal)
prop.table(table(datos_r$DEATH_EVENT)) %>% round(digits = 2) #El modelo debe superar el 68%

# FILTRADO DE PREDICTORES MEDIANTE ANOVA, RANDOM FOREST Y CV-REPETIDA
# ============================================================================



#

# Se crea una semilla para cada partición y cada repetición: el vector debe
# tener B+1 semillas donde B = particiones * repeticiones.
particiones = 10
repeticiones = 5
```

```
134 set.seed(123)
135 seeds <- sample.int(1000, particiones * repeticiones + 1)
136
137 # Control por filtrado
138 ctrl_filtrado <- sbfControl(functions = rfSBF, method = "repeatedcv",
139                             number = particiones, repeats = repeticiones,
140                             seeds = seeds, verbose = FALSE,
141                             saveDetails = TRUE, allowParallel = TRUE)
142 set.seed(234)
143 rf_sbf <- sbf( DEATH_EVENT~ ., data = datos_r,
144               sbfControl = ctrl_filtrado,
145               # argumentos para el modelo de evaluación
146               ntree = 500)
147 rf_sbf
148 #Variables predictoras seleccionadas (age,ejection_fraction, serum_creatinine,
      serum_sodium )
149
150 datos = datos_r %>% select(DEATH_EVENT,age,ejection_fraction, serum_creatinine
    ,  serum_sodium)
151
152 ##############################################################################
153 ############ Ajuste de modelos con CARET
154 ##############################################################################
155
156 #Usaremos las variables predictoras proporcionadas para tratar de ajustar un
    modelo capaz de predecir
157 #si el paciente muere o no
158
159 #Regresión Logística
160 #Árbol de clasificación
161 #Red neuronal
162 #SVM
163 #Random Forest
164 #Gradient Boosting
165 #Stacking
166
167 #Creamos el set de entrenamiento y el set de test:
168
169 set.seed(8)
170
171 # Seleccionamos las filas aleatoriamente cogiendo el 80% de las observaciones
172 filas_entrenamiento <- createDataPartition(datos$DEATH_EVENT, p=0.7, list=
    FALSE)
173
174 # Creamos el set de entrenamiento
175 trainData <- datos[filas_entrenamiento,]
176
177 # Creamos el set de test
178 testData <- datos[-filas_entrenamiento,]
179
180
181
182
183 #Creamos la matriz de observaciones de predicciones y el vector de  respuestas
```

```
184
185 x = trainData[,-1 ]
186 y = trainData[,1]
187
188 summary(x)
189 summary(y)
190
191
192
193 # DEFINICIÓN DEL ENTRENAMIENTO
194 #==============================================================================
195 #Logística
196 set.seed(12345)
197
198 control_train <-  trainControl(method = 'repeatedcv',
199                                number = 5,
200                                repeats =  5,
201                                search = 'random')
202
203
204
205 logist <- train(x, y, method = 'glm',
206                 trControl = control_train,
207                 preProcess = "center",
208                 family = 'binomial')
209
210
211 summary(logist)
212
213 #parece que la variable serukm sodim no es significativa en el modelo, lo
    ajustamos sin ella:
214
215 logist<- train(x[,-4], y, method = 'glm',
216               trControl = control_train,
217               preProcess = "center",
218               family = 'binomial')
219
220 summary(logist)
221
222 #Test de las predicciones del modelo:
223
224 test_pred_logist <- predict(logist, newdata = testData)
225 test_pred_logist
226
227 # Matriz de confusión:
228 confusionMatrix(test_pred_logist, testData[,1], positive = "si" )
229
230
231
232 #parece que la variable age se podría retirar del  modelo, lo ajustamos sin
    ella:
233
234 #x2=trainData[,c(3,4)]
235 #y2=trainData[,1]
236 #T2=trainData[,c(1,3,4)]
```

```
#
#summary(T2)
#
#
#logist<- train(x2,y2,  method = 'glm',
#                trControl = control_train,
#                preProcess = "center",
#                family = 'binomial')
#
#summary(logist)


##Test de las predicciones del modelo:
#
#test_pred_logist <- predict(logist, newdata = testData)
#test_pred_logist
#
## Matriz de confusión:
#confusionMatrix(test_pred_logist, testData[,1] )
#
#
#Predice mejor el que incluye la edad

# Árbol de clasificación
set.seed(12345)

control_train <-  trainControl(method = 'repeatedcv',
                               number = 5,
                               repeats =  5,
                               search = 'random')




class_tree <- train(x, y, method="rpart", trControl=control_train, preProcess
    = "center")
test_pred_class_tree <- predict(class_tree, newdata = testData)
test_pred_class_tree

confusionMatrix(test_pred_class_tree, testData[,1], positive = "si")



# NNET
set.seed(12345)
control_train <-  trainControl(method = 'repeatedcv',
                               number = 5,
                               repeats =  5,
                               search = 'random')


nnet <- train(x, y, method = "nnet",trControl=control_train ,preProcess = c('
    center', 'scale'), tuneLength = 10)
nnet$modelInfo
nnet$results
nnet$bestTune
```

```
291
292 test_pred_nnet <- predict(nnet, newdata = testData)
293 test_pred_nnet
294
295 confusionMatrix(test_pred_nnet, testData[,1] )
296
297 # SVM
298 set.seed(12345)
299 control_train <-  trainControl(method = 'repeatedcv',
300                                number = 5,
301                                repeats =  5,
302                                search = 'random')
303
304 SVM <- train(DEATH_EVENT~., data=trainData, method="svmLinear", trControl=
        control_train, preProcess = c("center","scale"))
305
306 test_pred_SVM <- predict(SVM, newdata = testData)
307 test_pred_SVM
308
309 confusionMatrix(test_pred_SVM, testData[,1] )
310
311 # RF
312
313 set.seed(12345)
314 control_train <-  trainControl(method = 'repeatedcv',
315                                number = 5,
316                                repeats =  5,
317                                search = 'random')
318
319 #Para el randomforest utilizamos todos los predictores disponibles
320
321 mtry <- sqrt(ncol(x))
322 tunegrid <- expand.grid(.mtry=mtry)
323
324 #Creamos el set de entrenamiento y el set de test:
325
326 set.seed(8)
327
328 # Seleccionamos las filas aleatoriamente cogiendo el 80% de las observaciones
329 filas_entrenamiento_r <- createDataPartition(datos_r$DEATH_EVENT, p=0.7, list=
        FALSE)
330
331 # Creamos el set de entrenamiento
332 trainData_r <- datos_r[filas_entrenamiento,]
333
334 # Creamos el set de test
335 testData_r <- datos_r[-filas_entrenamiento,]
336
337
338 x_r=trainData_r[,-13]
339 y_r=trainData_r[,13]
340
341 RF <- train(x_r, y_r, method = "rf", trControl=control_train,tuneLength  = 15
        ,metric = 'Accuracy',tuneGrid=tunegrid)
342
343
```

```
344 test_pred_RF <- predict(RF, newdata = testData_r)
345 test_pred_RF
346
347 confusionMatrix(test_pred_RF, testData_r[,13] )
348
349
350
351
352
353 #Boosting
354 set.seed(12345)
355 control_train <-  trainControl(method = 'repeatedcv',
356                                number = 5,
357                                repeats =  5,
358                                search = 'random')
359
360 #Grid para buscar los mejores hiperparámetros
361
362 xgbGrid <- expand.grid(shrinkage = seq(0.1, 1, by = 0.2),
363                        interaction.depth = c(1, 3, 7, 10),
364                        n.minobsinnode = c(2, 5, 10),
365                        n.trees = c(100, 300, 500, 1000))
366
367
368
369 Boosting_Tree <- train(DEATH_EVENT~., data=trainData, tuneGrid = xgbGrid,
370                        method = "gbm", trControl=control_train)
371
372
373 test_pred_boost <- predict(Boosting_Tree, newdata = testData)
374 test_pred_boost
375
376 confusionMatrix(test_pred_boost, testData[,1] )
377
378
379
380
381
382
383
384 #Stacking
385
386 set.seed(12345)
387
388 control <- trainControl(method = "repeatedcv", number = 5, repeats = 5, search
        = "grid",
389                         savePredictions = "final", index = createResample(
                               trainData$DEATH_EVENT, 5),
390                         summaryFunction = twoClassSummary, classProbs = TRUE,
                               verboseIter = TRUE)
391
392 control_train <-  trainControl(method = 'repeatedcv',
393                                number = 5,
394                                repeats =  5,
395                                search = 'random',
396                                savePredictions=TRUE)
```

```
397
398 algorithmList <- c('rpart', 'svmLinear', 'nnet')
399
400
401 model_base <- caretList(DEATH_EVENT ~ ., data = trainData, trControl = control
    , methodList = algorithmList, metric = "ROC")
402
403 res <- resamples(model_base)
404 summary(res)
405
406
407 # Stack
408 stackControl <- trainControl(method = "repeatedcv", number = 5, repeats = 5,
    savePredictions = TRUE, classProbs = TRUE, verboseIter = TRUE)
409
410 stack <- caretStack(model_base, method = "rf", metric = "Accuracy", trControl
    = stackControl)
411
412 # Predict
413
414 stack_val_preds <- predict(stack, newdata = testData)
415 stack_val_preds
416
417 confusionMatrix(stack_val_preds, testData[,1] )
418
419
420 ####################################################################
421 #########
422 ######### Plots  y tablas
423 ####################################################################
424
425 Modelos_Clasif= c('R. Logística', 'Arbol de Clasificación', 'NNET', 'SVM', '
    Random Forest', 'Gradient Boosting', 'Stacking')
426 Accuracy =  round(c(0.75, 0.7159, 0.7614, 0.7727, 0.8636 , 0.7614 , 0.7386 )
    ,2)
427 Sensitivity=round(c(0.4286,0.8167,0.9167,0.9167,0.9667,0.8833,0.8667),2)
428 Specificity=round(c(0.9000,0.5000,0.4286,0.4643,0.6429,0.5000,0.4643),2)
429 Balanced_Accuracy=round(c(0.6643,0.6583,0.6726,0.6905,0.8048,0.6917,0.6655),2)
430
431 Tabla= rbind(Modelos_Clasif,Accuracy,Sensitivity,Specificity,Balanced_Accuracy
    )
432
433 colnames(Tabla)<-Modelos_Clasif
434
435 Tabla=t(Tabla)
436 Tabla=data.frame(Tabla)
437 Tabla$Modelos_Clasif <- factor(Tabla$Modelos_Clasif,
438                                levels = Modelos_Clasif,
439                                labels=c('R.L.','A.C.','NNET','SVM','RF','GB','ST'
                                     ))
440
441
442 Tabla = Tabla[order(Tabla$Accuracy),]
443
444
445
```

```
446
447
448
449 plot(Tabla[,2] , ylim=c(0:1), xaxt = "n", pch=19, col=2, xlab= "Modelos", ylab
        ='Precisión',cex=1.5)
450 axis(1, at=1:7, labels=c('A.C.','ST','R.L.','NNET', 'GB','SVM','RF' ))
451
452
453
454 barplot(as.numeric(Tabla[,2]), ylim=c(0,1),xlab= "Modelos", ylab='Precisión',
        names.arg = c('A.C.','ST','R.L.','NNET', 'GB','SVM','RF' ),
455          col='#FFCC66')
456
457 axis(2, at = seq(from = 0, to = 1, by = 0.1))
458
459 ##############################################################################################
460 ##############################################################################################
461 ##############################################################################################
462 ##################################### P2 -- Regresión
        ##################################
463
464 # I-Cheng Yeh, "Modeling of strength of high performance concrete using
        artificial
465 # neural networks," Cement and Concrete Research, Vol. 28, No. 12, pp.
        1797-1808 (1998).
466
467 ##############################################################################################
468 ##############################################################################################
469 #0 - Preanálisis:
470
471 #1 - Cargamos los datos:
472
473 datos_conc= read.csv('Concrete_Data.csv',header=T)
474
475 #Datos de salida (Posibles Variables respuesta)
476 ##############################################################################
477
478 #Cement (component 1) -- quantitative -- kg in a m3 mixture -- Input Variable
479
480 #Blast Furnace Slag (component 2) -- quantitative -- kg in a m3 mixture --
        Input Variable
481
482 #Fly Ash (component 3) -- quantitative -- kg in a m3 mixture -- Input Variable
483
484 #Water (component 4) -- quantitative -- kg in a m3 mixture -- Input Variable
485
486 #Superplasticizer (component 5) -- quantitative -- kg in a m3 mixture -- Input
         Variable
487
```

```
#Coarse Aggregate (component 6) -- quantitative -- kg in a m3 mixture -- Input Variable

#Fine Aggregate (component 7) -- quantitative -- kg in a m3 mixture -- Input Variable

#Age -- quantitative -- Day (1~365) -- Input Variable

#Concrete compressive strength -- quantitative -- MPa -- Output Variable



#Datos de salida (Posibles Variables predictoras)
################################################################################



glimpse(datos_conc)
summary(datos_conc)


#Comprobamos si hay NA y duplicados


anyNA(datos_conc)

nrow(datos_conc[duplicated(datos_conc), ])


#No hay NA en nuestros datos.

#hay datos duplicados asique procedemos a eliminarlos:

datos_conc= distinct(datos_conc)



#Descartamos los datos relativos al tiempo de seguimiento, ya que el tiempo durante el que se ha observado al paciente no influye en
# el resultado de muerte o supervivencia


#############################################################################


#ver que varibales presentan varianza proxima a acero
datos_conc %>% select(Cement,blast_f_r,fly_ash,water,superplasticizer,coarse_agg,fine_agg,age,cc_strength ) %>% nearZeroVar(saveMetrics = TRUE)

# FILTRADO DE PREDICTORES por CV
#
    ============================================================================


#

```

```
# Se crea una semilla para cada partición y cada repetición: el vector debe
# tener B+1 semillas donde B = particiones * repeticiones.
particiones = 10
repeticiones = 5
set.seed(123)
seeds <- sample.int(1000, particiones * repeticiones + 1)

# Control del filtrado
ctrl_filtrado <- sbfControl(functions = rfSBF, method = "repeatedcv",
                            number = particiones, repeats = repeticiones,
                            seeds = seeds, verbose = FALSE,
                            saveDetails = TRUE, allowParallel = TRUE)
set.seed(234)
rf_sbf <- sbf( cc_strength~ ., data = datos_conc,
               sbfControl = ctrl_filtrado,
               # argumentos para el modelo de evaluación
               ntree = 500)

rf_sbf$optVariables
#Variables predictoras seleccionadas (cc_strength ,age, blast_f_r, coarse_agg,
     fly_ash, water )

datos = datos_conc %>% select(cc_strength ,age, blast_f_r, coarse_agg,
    fine_agg, fly_ash, water)









################################################################################

############ Ajuste de modelos con CARET
################################################################################


#Usaremos las variables predictoras proporcionadas para tratar de ajustar un
    modelo capaz de predecir
#si el paciente muere o no

#Regresión Lineal Múltiple
#Árbol de regresión
#Red neuronal
#SVM
#Random Forest
#Gradient Boosting
#Stacking


# Estandarizamos los datos debido a las diferencias de escala entre los
    diferentes predictores
#maxs <- apply(datos, 2, max)
#mins <- apply(datos, 2, min)
#datos <- as.data.frame(scale(datos, center = mins,
#                             scale = maxs - mins))


```

```
587
588
589 #Create a partition for training and testing
590 set.seed(12345)
591 inTrain <- createDataPartition(y = datos$cc_strength, p = 0.7, list = FALSE)
592
593 #Create training and test sets
594 training <- datos[inTrain,]
595 testing <- datos[-inTrain,]
596
597 #usaremos para evaluar
598
599 #R2
600 #RMSE
601 #MAE
602
603 #LM
604 ########################################################################
605
606 set.seed(12345)
607 control_train <-  trainControl(method = 'repeatedcv',
608                                number = 5,
609                                repeats =  5,
610                                search = 'random')
611
612 LM <- train(cc_strength ~ ., data = training, method = 'lm',
613             preProcess = c('center', 'scale'),trControl=control_train)
614
615 maxs <- apply(datos, 2, max)
616 mins <- apply(datos, 2, min)
617 datos2 <- as.data.frame(scale(datos, center = mins,
618                               scale = maxs - mins))
619
620 LM2 <-lm(cc_strength ~ ., data = datos2)
621
622 summary(LM2)
623
624
625 plot(LM2)
626
627 #Residuos normales pero leverage
628
629
630 #Predict on the test set
631 pred <- predict(LM, testing)
632
633 LM_Stat=data.frame( Model='LM',
634                     R2 = R2(pred, testing$cc_strength),
635                     RMSE = RMSE(pred, testing$cc_strength),
636                     MAE = MAE(pred, testing$cc_strength)
637 )
638
639
640 ##### QUe pasa con el modelo de regresión lineal múltiple? Diagnóstico:
641
642 summary(LM2)
```

```
643
644 x=LM2$residuals
645 ks.test(LM2$residuals)
646 ks.test(x, "pnorm", mean=0, sd=sd(x)) #el error es normal
647
648 plot(LM2)
649
650
651
652
653 # Test Breusch-Pagan:
654
655 plot(LM2$fitted.values,stdres(LM2), xlab='Valores ajustados',ylab='Residuos
    estandarizados') # H0: Los errores no son Homocedásticos
656
657 bptest(LM2) #Se rechaza la homocedasticidad
658
659 ################################################################
660 #Observamos la hipótesis de linealidad
661
662 ggpairs(datos, lower = list(continuous = "smooth"),
663         diag = list(continuous = "barDiag"), axisLabels = "none")
664
665
666
667
668 #No param
    ###########################################################################
669 set.seed(12345)
670 control_train <-  trainControl(method = 'repeatedcv',
671                                number = 5,
672                                repeats =  5,
673                                search = 'random')
674
675 Gamspline <- train(cc_strength ~ ., data = training, method = "gamSpline",
    preProcess = c('center', 'scale'),trControl=control_train, df=6)
676
677 #Predict on the test set
678 pred <- predict(Gamspline, testing)
679
680 #Evaluate the model
681 GamSp_stat= data.frame( Model='Gam.Sp.',
682   R2 = R2(pred, testing$cc_strength),
683   RMSE = RMSE(pred, testing$cc_strength),
684   MAE = MAE(pred, testing$cc_strength)
685 )
686
687 #Regression_tree ##############################################################
688 set.seed(12345)
689 control_train <-  trainControl(method = 'repeatedcv',
690                                number = 5,
691                                repeats =  5,
692                                search = 'random')
693
694 Regre_tree <- train(cc_strength ~ ., data = training, method = "rpart",
    preProcess = c('center', 'scale'),trControl=control_train)
```

```
695
696 #Predict on the test set
697 pred <- predict(Regre_tree, testing)
698
699 #Evaluate the model
700 Arbol_clas_stat=data.frame(Model='A.R.',
701   R2 = R2(pred, testing$cc_strength),
702   RMSE = RMSE(pred, testing$cc_strength),
703   MAE = MAE(pred, testing$cc_strength)
704 )
705
706 #Regression_Randomforest ##################################################
707
708 set.seed(12345)
709 RFTrain <- createDataPartition(y = datos$cc_strength, p = 0.7, list = FALSE)
710
711 #Create training and test sets
712 RFtraining <- datos[RFTrain,]
713 RFtesting <- datos[-RFTrain,]
714
715
716
717
718 set.seed(145)
719 control_train <-  trainControl(method = 'repeatedcv',
720                                number = 5,
721                                repeats =  5,
722                                search = 'random')
723
724
725 Regre_RF <- train(cc_strength ~ ., data = RFtraining, method = "rf",preProcess
      = c('center', 'scale'),trControl=control_train)
726
727 #Predict on the test set
728 pred <- predict(Regre_RF, RFtesting)
729
730 #Evaluate the model
731 RF_stat= data.frame(Model='RF',
732   R2 = R2(pred, RFtesting$cc_strength),
733   RMSE = RMSE(pred, RFtesting$cc_strength),
734   MAE = MAE(pred, RFtesting$cc_strength)
735 )
736
737
738
739
740 #Gradient Stochastic Boosting #####################################
741 set.seed(12345)
742 control_train <-  trainControl(method = 'repeatedcv',
743                                number = 5,
744                                repeats =  5,
745                                search = 'random')
746
747
748 BGLM <- train(cc_strength ~ ., data = training, method = "gbm",preProcess = c
     ('center', 'scale'),trControl=control_train )
```

```
749
750 #Predict on the test set
751 pred <- predict(BGLM, testing)
752
753
754 #Evaluate the model
755 GB_stat=data.frame(Model='GB',
756   R2 = R2(pred, testing$cc_strength),
757   RMSE = RMSE(pred, testing$cc_strength),
758   MAE = MAE(pred, testing$cc_strength)
759 )
760
761
762 #Stacking ##################################################################
763 #Stacking
764
765 set.seed(12345)
766
767 trainControl <- trainControl(method="repeatedcv",
768                              number=10,
769                              repeats=5,
770                              savePredictions=TRUE)
771
772 modelos_base <- c('lm',"gamSpline","rpart","rf")
773
774
775
776
777 # Ensemble models
778
779
780 set.seed(100)
781 modelos_base_s <- caretList(cc_strength~., data = training,
782                     trControl=trainControl,
783                     methodList=modelos_base)
784
785
786
787 # Combine Predictions from multiple models
788 set.seed(12345)
789
790 stackControl <- trainControl(method="repeatedcv",
791                              number=10,
792                              repeats=5,
793                              savePredictions=TRUE)
794
795 # Ensemble the predictions of `models` to form a new combined prediction based
      on glm
796 stack.glm <- caretStack(modelos_base_s, method="glm", trControl=stackControl,
     preProcess = c('center', 'scale'), tuneLength = 10)
797 print(stack.glm)
798
799 stack_pred <- predict(stack.glm, newdata=testing)
800
801
802 #Evaluate the model
```

```
803 Stack_stat=data.frame(Model='Stack',
804   R2 = R2(stack_pred, testing$cc_strength),
805   RMSE = RMSE(stack_pred, testing$cc_strength),
806   MAE = MAE(stack_pred, testing$cc_strength)
807 )
808
809
810 ######################################################################
811 #########
812 ######### Plots  y tablas
813 ######################################################################
814
815 Tabla_P2 = rbind(LM_Stat, GamSp_stat, Arbol_clas_stat, RF_stat, GB_stat,
        Stack_stat)
816
817 cbind(Tabla_P2$Model,round(Tabla_P2[,2:4],2) )->Tabla_P2
818
819 Tabla_P2 = Tabla_P2[order(Tabla_P2$R2),]
820
821 rownames(Tabla_P2)<-Tabla_P2$`Tabla_P2$Model`
822
823 #PLOT
824 plot(Tabla_P2[,2], ylim=c(0,1), xaxt = "n", pch=19, col=2, xlab= "Modelos",
        ylab='R cuadrado', cex=1.5)
825 axis(1, at=1:6, labels=Tabla_P2[,1])
826
827
828
829 barplot(Tabla_P2[,2], ylim=c(0,1),xlab= "Modelos", ylab='R cuadrado',names.arg
        = Tabla_P2[,1],
830           col="#FFCC66")
831 axis(2, at = seq(from = 0, to = 1, by = 0.1))
```

Bibliografía

[1] BREIMAN, Leo. Bagging predictors. Machine learning, 1996, vol. 24, no 2, p. 123-140.

[2] DE BOOR, Carl; DE BOOR, Carl. A practical guide to splines. New York: springer-verlag, 1978.

[3] DRUCKER, Harris, et al. Support vector regression machines. Advances in neural information processing systems, 1996, vol. 9.

[4] KUHN, Max, et al. Package 'caret'. The R Journal, 2020, vol. 223, p. 7.

[5] CASAL, R.; BOUZAS, J.; OVIDEO, M. Aprendizaje Estadístico. 2021.

[6] CHICCO, Davide; JURMAN, Giuseppe. Machine learning can predict survival of patients with heart failure from serum creatinine and ejection fraction alone. BMC medical informatics and decision making, 2020, vol. 20, no 1, p. 1-16.

[7] CLEVELAND, William S. Robust locally weighted regression and smoothing scatterplots. Journal of the American statistical association, 1979, vol. 74, no 368, p. 829-836.

[8] EFRON, Bradley. Computers and the theory of statistics: thinking the unthinkable. SIAM review, 1979, vol. 21, no 4, p. 460-480.

[9] EILERS, Paul HC; MARX, Brian D. Flexible smoothing with B-splines and penalties. Statistical science, 1996, vol. 11, no 2, p. 89-121.

[10] FAHRMEIR, Ludwig, et al. Regression models. En Regression. Springer, Berlin, Heidelberg, 2021. p. 23-84.

[11] FARAWAY, Julian J. Extending the linear model with R: generalized linear, mixed effects and nonparametric regression models. Chapman and Hall/CRC, 2016.

[12] FREUND, Yoav, et al. Experiments with a new boosting algorithm. En icml. 1996. p. 148-156.

[13] FRIEDMAN, Jerome H. Greedy function approximation: a gradient boosting machine. Annals of statistics, 2001, p. 1189-1232.

[14] HASTIE, Trevor, et al. The elements of statistical learning: data mining, inference, and prediction. New York: springer, 2009.

[15] HORNIK, Kurt; STINCHCOMBE, Maxwell; WHITE, Halbert. Multilayer feedforward networks are universal approximators. Neural networks, 1989, vol. 2, no 5, p. 359-366.

[16] KINGMA, Diederik P.; BA, Jimmy. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980, 2014.

[17] JAMES, Gareth, et al. An introduction to statistical learning. New York: springer, 2013.

[18] NOCEDAL, Jorge. Updating quasi-Newton matrices with limited storage. Mathematics of computation, 1980, vol. 35, no 151, p. 773-782.

[19] SCHLOERKE, Barret; CROWLEY, Jason; COOK, Di. Package 'GGally'. Extension to 'ggplot2.'See, 2018, vol. 713.

[20] SCHOMAKER, Michael; HEUMANN, Christian. Model selection and model averaging after multiple imputation. Computational Statistics & Data Analysis, 2014, vol. 71, p. 758-770.

[21] SRIVASTAVA, Nitish, et al. Dropout: a simple way to prevent neural networks from overfitting. The journal of machine learning research, 2014, vol. 15, no 1, p. 1929-1958.

[22] THIELE, Jan; MARKUSSEN, Bo. Potential of GLMM in modelling invasive spread. CABI Reviews, 2012, no 2012, p. 1-10.

[23] YEH, I.-C. Modeling of strength of high-performance concrete using artificial neural networks. Cement and Concrete research, 1998, vol. 28, no 12, p. 1797-1808.