ENSEMBLES, MIXTURAS E INTERPRETABILIDAD DE MODELOS DE MACHINE LEARNING

ÍNDICE:

[**INTRODUCCIÓN: LA NORMATIVA GDPR** 2](#_Toc179027661)

[**1.- BASE DE UN MODELO DE ENSEMBLE: ÁRBOLES DE DECISIÓN Y DE REGRESIÓN** 3](#_Toc179027662)

[1.1.- Árboles de Regresión 4](#_Toc179027663)

[1.2.- Árboles de Decisión 9](#_Toc179027664)

[**2.- PRINCIPIOS DE COMPOSICIÓN DE MODELOS DE ÁRBOLES: BAGGING** 11](#_Toc179027665)

[2.1.- Concepto de ensemble de modelos 11](#_Toc179027666)

[2.2.- El bagging 13](#_Toc179027667)

[Anexo I: La validación cruzada 16](#_Toc179027668)

[**3.- RANDOM FOREST Y BOOSTING MODELS** 18](#_Toc179027669)

[**4.- MEZCLA GENÉRICA DE MODELOS** 22](#_Toc179027670)

[**5.- INTERPRETABILIDAD DE MODELOS: LA LIBRERÍA LIME** 24](#_Toc179027671)

[5.1 La librería LIME 24](#_Toc179027672)

[5.2 La librería SHAP 27](#_Toc179027673)

[5.3 Otras opciones de interpretabilidad de modelos de analítica avanzada 30](#_Toc179027674)

[**6.- UN POCO DE ANALÍTICA AVANZADA CON PYTHON** 32](#_Toc179027675)

[El modelo de Random Forest. Un poco de detalle 32](#_Toc179027676)

[El modelo de Extreme Gradient Boosting o xgboost 33](#_Toc179027677)

[LightGBM un algoritmo de moda 34](#_Toc179027678)

[**CONCLUSIONES FINALES** 35](#_Toc179027679)

[**BIBLIOGRAFÍA** 36](#_Toc179027680)

# **INTRODUCCIÓN: LA NORMATIVA GDPR**

Los modelos que se estudian “no viven en una realidad paralela”, sino que con frecuencia, las empresas pagan por su desarrollo, los aplican y buscan un retorno de la inversión o un beneficio de su aplicación.

La construcción de modelos requiere de datos, en muchas ocasiones personales, que las entidades obtienen de los individuos por la naturaleza del negocio (Ejemplo: Banca, Seguros, Seguridad Social, …) Claramente estos datos pueden ser mal utilizados y sin consentimiento por parte de quien cede la información que lo hace “en teoría” para obtener un servicio y sólo para ese servicio, por tanto debe controlarse su uso una vez cedido el dato / informació y es por ello por lo que existe la Ley de Protección de Datos la cual desde Marzo del 2018 y mediante la normativa europea Regulation (EU) 2016/679, la cual se conoce por su siglas en inglés como GDPR (General Data Protection Regulation), cambió de modo radical el uso de los datos personales dentro de la UE afectando por tanto a empresas como a los distintos interesados que puedan negociar con dicha información.

Lo que destaca de esta ley no es sólo la obligación de la protección del dato a todos los niveles, sino que además en sus artículos 15 y 22 ofrecen al ciudadano lo que se conoce como el: “derecho a la explicación de las acciones del algoritmo”, afirmándose que los ciudadanos tienen el derecho a conocer cuáles son las razones por las que un determinado algoritmo ha tomado una decisión que les afecta, es decir, no es sólo afecta al uso de los datos, sino a lo que se puede hacer con éstos y que en ocasiones llevaran aparejado algoritmos o modelos que como se sabe se crean con información.

Artículo 15:

1. El interesado tendrá derecho a obtener del responsable del tratamiento confirmación de si se están tratando o no datos personales que le conciernen y, en tal caso, derecho de acceso a los datos personales y a la siguiente información:
   1. Los fines del tratamiento
   2. Las categorías de datos personales de que se trate
   3. Los destinatarios o las categorías de destinatarios a los que se comunicaron o serán comunicados los datos personales, en particular destinatarios en terceros u organizaciones internacionales
   4. De ser posible, el plazo previsto de conservación de los datos personales o, de no ser posible, los criterios utilizados para determinar este plazo
   5. La existencia del derecho a solicitar del responsable la rectificación o supresión de datos personales o la limitación del tratamiento de datos personales relativos al interesado, o a oponerse a dicho tratamiento
   6. El derecho a presentar una reclamación ante una autoridad de control
   7. Cuando los datos personales no se hayan obtenido del interesado, cualquier información disponible sobre su origen
   8. La existencia de decisiones automatizadas, incluida la elaboración de perfiles, a que se refiere el artículo 22, apartados 1 y 4, y, al menos en tales casos, información significativa sobre la lógica aplicada, así como la importancia y las consecuencias previstas de dicho tratamiento para el interesado

1. Cuando se transfieran datos personales a un tercer país o a una organización internacional, el interesado tendrá derecho a ser informado de las garantías adecuadas en virtud del artículo 46 relativas a la transferencia.
2. El responsable del tratamiento facilitará una copia de los datos personales objeto de tratamiento. El responsable podrá percibir por cualquier otra copia solicitada por el interesado un canon razonable basado en los costes administrativos. Cuando el interesado presente la solicitud por medios electrónicos, y a menos que este solicite que se facilite de otro modo, la información se facilitará en un formato electrónico de uso común.
3. El derecho a obtener copia mencionado en el apartado 3 no afectará negativamente a los derechos y libertades de otros

Artículo 22:

1. Todo interesado tendrá derecho a no ser objeto de una decisión basada únicamente en el tratamiento automatizado, incluida la elaboración de perfiles, que produzca efectos jurídicos en él o le afecte significativamente de modo similar.
2. El apartado 1 no se aplicará si la decisión:
   1. Es necesaria para la celebración o la ejecución de un contrato entre el interesado y un responsable del tratamiento
   2. Está autorizada por el Derecho de la Unión o de los Estados miembros que se aplique al responsable del tratamiento y que establezca asimismo medidas adecuadas para salvaguardar los derechos y libertadas y los interese legítimos del interesado, o
   3. Se basa en el consentimiento explícito del interesado
3. En los casos a que se refiere el apartado 2, letras a) y c), el responsable del tratamiento adoptará las medidas adecuadas para salvaguardar los derechos y libertades y los intereses legítimos del interesado, como mínimo el derecho a obtener intervención humana por parte del responsable, a expresar su punto de vista y a impugnar la decisión.
4. Las decisiones a que se refiere el apartado 2 no se basarán en las categorías especiales de datos personales contempladas en el artículo 9, apartado 1, salvo que se aplique el artículo 9, apartado 2, letra a) o g), y se hayan tomado medidas adecuadas para salvaguardar los derechos y libertades y los intereses legítimos del interesado

A pesar del paso del tiempo, aún no se tiene claro cómo aplicar estos artículos pero cabe deducir que al menos:

-Debe tenerse en cuenta la historia de los datos, su procedencia y su destino (Art 15)

-Conviene poder explicar lo que hace realmente un algoritmo en la toma de decisiones (Art 22), sobre todo si esta decisión es importante en la vida de un ciudadano (Ejemplo: Denegación de una Hipoteca)

Esta sesión será la primera en la que introduce auténticas “cajas negras” de modelización y se observa además que los métodos que permiten construirlas son muy sencillos, por lo que conviene dedicar un apartado a lo que se conoce como interpretabilidad de modelos con librerías de funciones en R / Python específicas para tal fin.

# **1.- BASE DE UN MODELO DE ENSEMBLE: ÁRBOLES DE DECISIÓN Y DE REGRESIÓN**

Como se verá más adelante un *ensemble* es una agrupación de varios *modelos de aprendizaje*, en principio “débiles”, cuyo uso combinado a través de un *algoritmo* de entrenamiento es capaz de generar un modelo más potente en términos predictivos.

Por tanto se está ante la idea de que la combinación de modelos sencillos puede generar un nuevo modelo más potente que cada uno individualmente interesando en esta parte cómo van a ser los modelos base más utilizado para la generación de dicho modelo “superior”. Nótese que en estos primeros párrafos se está tratando de evitar el uso de la palabra complejidad, porque aunque claramente el modelo generado va a resultar más complejo, sin embargo lo que se busca es siempre la mejora de las predicciones aunque claramente hay que pagar un precio para tal fin.

Enfocando ahora hacia la base de los *modelos de ensemble* hay que comenzar por los *modelos* tipo *building blocks* de dichos *ensembles* y que no son más que aquellos modelos que durante la década del 2002 – 2012 eran el paradigma de lo que se conoce como *minería de datos* y que no son más que los conocidos como *árboles de decisión* (cuando se pretende explicar una variable categórica) o *de regresión* (cuando lo que se pretende explicar es una variable continua). En esta parte se sigue fundamentalmente la línea de exposición de **James G. et al (2013)** la cual se complementa con otras fuentes.

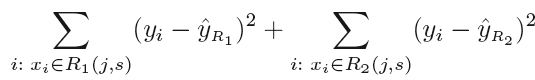
## 1.1.- Árboles de Regresión

Históricamente el primer *algoritmo* para la construcción de este tipo de árboles el denominado *AID* que apareció antes que el *TAID* y el popular *CART*. Tanto el primero como el último se desarrollaron a mediados de los 80 según se detalla en **Breiman et al (1984)** surgiendo después algunos más para corregir algunos defectos, estos nuevos *algoritmos* que sólo se menciona serían entre otros los siguientes: *CRUISE*, *GUIDE*, *QUEST* y el *RPART*. A continuación, se pasa a describir los pasos básicos para el caso de la construcción de un árbol de regresión a través de un *pseudo-algoritmo* sencillo y resumido:

* Se comienza en el *Nodo Raiz* con una variable explicativa que previamente se ha seleccionado (generalmente siguiendo un criterio de *importancia de la variable* que se verá en algunos ejemplos). Suponiendo que dicha variable es continua, cabe seleccionar un número *s* que permite dividir el espacio en 2 regiones tal como sigue (también con algún criterio de optimalidad *discriminante*):



Así pues, el *algoritmo* va a elegir tanto la variable j como el corte, de modo que se minimiza la siguiente cantidad (y que se denomina *Función de Impureza* de un *Árbol de Regresión*):



* Se repite el anterior proceso en cada una de las ramas creadas, siguiendo alguna metodología determinada (que vendría dictada en los anteriormente citados *algoritmos*)

Observación: En el caso de variables discretas, el conjunto de posibilidades se reduce al no analizarse entre un continuo, sino que el modo de buscar el punto de corte es más sencillo porque cabe utilizar directamente las categorías de dichas variables

Por tanto, como puede observarse cualquiera de los *algoritmos* comentados aunque tienen en cuenta los pasos básicos anteriormente descritos, no resultan triviales de presentar a un alto detalle ya que:

* Habría que aclarar cómo ir eligiendo *s* y simultáneamente la variable *p* para tener una convergencia rápida a la solución, el problema se complica si *p* es elevado
* Habría que pensar qué hacer en caso de variables categóricas ordinales
* Habría que pensar cómo tratar un número elevado de categorías

Ejemplo 01: Determinantes del precio de la vivienda

Los *árboles de regresión* tienen un gran elenco de aplicación, en *ejemplos/programa01.R* se observa una de éstas aplicado a la determinación de las causas principales que determinan el precio medio de la vivienda en la ciudad de Boston con el conocido dataset *Boston* **James G. et al (2013)**

Diagrama

Descripción generada automáticamente

Se observa que algunas de las variables escogidas resultan ser:

*lstat*: lower status of the population (percentil)

*rm*: average number of rooms per dwelling

*crim*: per capita crime rate per town

*age:* proportion of owner-occupied units built prior to 1940

Siendo una variable que tiene que ver con la situación (económica) de la persona, otra con el tamaño de la vivienda, una variable que describe el tipo de zona en el que está la vivienda y por último otra que tiene que ver con la antigüedad de viviendas de la zona. Aunque más adelante se analizarán cuestiones como la validez del modelo y demás, unas primeras preguntas a nivel “técnico” podrían ser:

Preguntas: ¿Cuántas *hojas* tiene el *árbol*?

¿Cuál es la *profundidad*?

¿Cuál es el *nodo* con más datos?

¿Cuál es el *nodo* con menos datos?

¿Qué debilidades observa en esta modelización?

\_\_\_\_\_

La idea en los *algoritmos* de construcción de un *árbol de regresión* es la de generar *ramas* cuyas *varianzas* sean mínimas, de modo que se tengan datos muy parecidos dentro de cada *nodo* pero que fuera de éstos, dichos datos sean lo más distintos posibles. El resultado final es un esquema de interpretación sencilla basado en reglas tal y como se ha visto en el ejemplo anterior, pero hay varias cuestiones a tener en cuenta en un algoritmo de este tipo que deben ser consideradas por el *DS*:

-¿Existe un número suficiente de observaciones en los *nodos/hojas* finales del *árbol*?

-¿Cabe imponer algún criterio de parada para que el *árbol* no crezca demasiado?

-¿Puede obtenerse un *árbol* más sencillo que no pierda potencia?

Así pues, unos criterios habituales a imponer en estos algoritmos simples y los mas complicados que se construyan a partir de éstos serían:

- Imponer un número mínimo de observaciones para poder seguir sub-dividiendo

- Imponer un número mínimo de observaciones resultantes en los nodos/hojas resultantes

- Imponer un criterio mínimo de mejora de (porcentual en general) de la función de coste utilizada para la sub-división

- Imponer un nivel de profundidad máximo en el árbol

Ejemplo 02: Restricciones en el crecimiento de un *arbol de regresión*

Profundizando un poco más en el ejemplo anterior, cabe considerar la siguiente extensión que se en cuentra en *ejemplos/programa02.R* (y que se recomienda que no se abra hasta haber leído y trabajado todo este ejemplo) donde se trata de imponer concidiones como las siguientes al anterior ejercicio:

- Un nodo se sub-divide si existen al menos 50 observaciones

- En las hojas finales del árbol deben quedar al menos 25 observaciones

- La mejora debe de ser de al menos del 5% para que se considere la sub-división

En este caso debe analizarse. Usando ls siguiente expresión dentro de la función (averíguesse cómo)

Imagen que contiene Logotipo

Descripción generada automáticamente

Una vez que se obtiene este árbol la cuestión es ¿Es mejor o al menos similar al anterior? Si la respuesta es afirmativa y el árbol es más simple, será preferible, eso sí evaluando sobre un test ¿Cómo lo haría? ¿Qué ocurriría si se construye un árbol con las condiciones anteriores pero sólo se exige que la mejora sea de al menos el 2% para que se considere la subdivisión?

\_\_\_\_\_

Así pues con el anterior ejemplo se muestra el concepto de *poda* que tiene sentido tanto para los *regression* como para los *decision trees* de modo que dado un árbol, puede que tenga sentido de la reunificación de *hojas* hasta el de incluso *ramas*. Estas reunificaciones van a permitir dar más consistencia al *arbol* final, si bien claramente reduce *granularidad* y detalle, sin embargo, puede que no tenga sentido reglas formadas en base a 20 observaciones o menos.

Ejemplo 03: Restricciones en el crecimiento de un *arbol de regresión* en python

En Python el conjunto Boston no está habilitado por contener determinadas variables de tipo racista, para ello se propone otro dataset como el que se puede conseguir en el código *ejemplos/programa03.py* *o* con las líneas:

from sklearn.datasets import fetch\_openml

housing = fetch\_openml(name="house\_prices", as\_frame=True)

En este caso, cuando a python se le pide entrenar un árbol, éste crece sin freno tal y como se muestra en el primer resultado gráfico que se obtiene:

Imagen que contiene Esquemático

Descripción generada automáticamente

Imponiendo algunas condiciones de *poda* como las siguientes se obtiene un modelo más razonable con las variables escogidas:

model2 = DecisionTreeRegressor(random\_state = 155,

max\_depth = 4,

min\_samples\_leaf = 25)

Diagrama

Descripción generada automáticamente

Que desde luego, se deja como ejercicio contrastar sin el test se obtiene una aproximación razonable con este modelo y compararla frente al anterior.

\_\_\_\_\_

## 1.2.- Árboles de Decisión

Los árboles de decisión son totalmente análogos a los anteriores, sin embargo hay 2 elementos clave que cambian:

* Por un lado está el hecho de que la variable respuesta va a ser de tipo categórica, y aunque no hay problema en que haya un conjunto amplio de categorías, aquí se limita el estudio al caso de variables binarias al ser el más sencillo (al final todos los algoritmos de más de 2 categorías pueden reducirse a uno de tipo binario)
* Por otro lado cambia la *función de coste* a utilizar por el algoritmo que no será de tipo cuadrático, sino que se hace uso de alternativas como por ejemplo las que se muestran a continuación:
  + Ganancia de información o reducción de la entropía acorde a la fórmula de *entropía de Shannon* siguiente que dadas 2 clases quedaría como:

A partir de esta *función de entropía*, se define lo que se conoce como *ganancia de entropía* que es la que se usa en *decision trees* para seleccinar la variable y el punto de corte a aplicar en cada iteración:

* + Uso del índice de Impureza de Gini que sigue la siguiente expresión (aplicada en la primera subdivisión):

Este índice tiene que ver con la reducción de la varianza de la cantidad de información ya que cuando hay *J* clases a dividir, la expresión anterior es equivalente a::

Análogamente al caso de los *arboles de regresión*. En este tipo de modelos, también puede ser controlado su crecimiento a través de condiciones tanto en *nodos* como en *hojas*. Adicionalmente será frecuente que dado un primer modelo se desee aplicar técnicas de simplificación del árbol o de *poda*, de hecho bajo el proceso anterior de imponer condiciones limitantes al crecimiento, ya se está de algún modo aplicando un tipo de *poda*, pero bajo este concepto se supone que se parte de un *modelo de decisión* o *regresión* *tree* ya desarrollado y entonces lo que se pretende es:

* Simplificar dicho modelo inicial mediante una inspección visual unificando ramas mediante *criterio experto* o *heurístico* bajo guías de conocimiento de negocio
* Respetando la estructura inicial crear un árbol más pequeño mediante el uso de *algoritmos* que penalicen la complejidad frente a la simplicidad del árbol a general. En este sentido una opción que se observa en el siguiente ejemplo es el uso del parámetro *cp* que altera la *función de costes* a minimizar del siguiente modo:

Con *T* el número de *hojas*, de modo que la *coste corregido por complejidad alpha* es igual a una función que depende aditivamente de la *impureza* o *error de predicción R* y de la cantidad de *hojas* modificada por el *complexity parameter* *cp* o *alpha* (tal y como se escribe habitualmente en las fórmulas)

Ejemplo 04: Podando un árbol mediante el *complexity parameter*

En el código *ejemplos/programa04.R* adaptado de <http://apuntes-r.blogspot.com.es/2014/09/predecir-perdida-de-clientes-con-arbol.html>, se hace uso del parámetro *cp* denominado *complexity parameter* donde a mayor valor, menor será el tamaño del árbol, una forma para determinarlo es observando un gráfico como el siguiente que (se obtiene también en el mencionado código):

Diagrama

Descripción generada automáticamente con confianza media

De este modo un posible valor sería 0.035, dando lugar como se observa a un árbol más simplificado donde no se ha perdido mucho de su potencia predictiva inicial.

\_\_\_\_\_

# **2.- PRINCIPIOS DE COMPOSICIÓN DE MODELOS DE ÁRBOLES: BAGGING**

## 2.1.- Concepto de ensemble de modelos

Definición de *ensemble de clasificadores*: es una agrupación de varios modelos de aprendizaje, que se construyen para resolver el mismo problema de aprendizaje automático y toman decisiones en conjunto para realizar predicciones.

Diagrama

Descripción generada automáticamente

**Figura 1 Esquema básico de un *ensemble de clasificadores***

Fuente: Zhou, Z. (2012) Ensemle Methods. Foundations and Algorithms

\_\_\_\_\_

Por tanto, construir un ensemble implica entrenar una serie de modelos base (*base learners*) y luego combinar sus predicciones de alguna manera. Existen dos grandes categorías de modelos base:

* *Modelos débiles*: Modelos poco potentes que funcionan ligeramente mejor que generar predicciones basadas en la prior de las clases o la media de regresión. En general son rápidos de entrenar. Ejemplo: Árboles de decisión, regresión lineal, …
* *Modelos fuertes*: Modelos potentes que requieren mucho esfuerzo para entrenarlo pero que suelen conseguir buenos resultados. Ejemplo: SVMs, Redes Neuronales, …

Por otro lado, debe tener en cuenta que Los modelos base de un ensemble se construyen de manera que se maximice la combinación de dos métricas, a menudo contrapuestas:

* *Precisión*: la precisión de la predicción combinada de un ensemble depende de la precisión de cada uno de sus modelos base
* *Diversidad*: si todos los modelos son idénticos o muy parecidos combinar sus predicciones no nos aportará nada

Así pues, lo ideal en un *ensemble* será el uso de modelos *precisos*, pero diversos entre sí existiendo dos métodos principales de construcción de este tipo de modelos:

* *Paralelos*: donde cada modelo trabaja en paralelo sobre los datos sin colaborar directamente entre ellos, es decir, *aprenden* de manera independiente. La predicción del ensemble es una combinación de la predicción de cada modelo base. Un esquema ilustrativo sería el siguiente:

Interfaz de usuario gráfica, Gráfico

Descripción generada automáticamente

Estos modelos explotan la *independencia* entre los modelos base y es una técnica que se fomenta cuando los modelos son *diversos*, son además ideales para el uso conjunto de *arquitecturas de entrenamiento* en *paralelo* al poderse entrenar cada modelo aisladamente para después ser comibnados

* *Secuenciales*: donde cada modelo trata de corregir los defectos del modelo anterior, es decir, cada modelo *aprende* de los errores del modelo anterior para hacer predicciones más precisas. La predicción de todos los modelos también se combina según un equema como el siguiente:

Gráfico, Gráfico de barras

Descripción generada automáticamente

En cambio estos modelos explotan la *dependencia* entre los modelos base y es una técnica donde se demuestra que se logra reducir el *error global* si bien su uso bajo *arquitecturas en paralelo* ya no resulta tan posible y por tanto es normal que su *entrenamiento* resulte más lento, aunque como se verá en los ejemplos, es algo sobre lo que se ha trabajado mucho en los últimos años

Uno de los resultados que justifican el funcionamiento de los *ensembles* parte de la idea de suponer que se parte de *N* clasificadores lineales donde se define un *ensemble* del siguiente modo:

En este caso la probabilidad de error de uno de los *clasificadores base* cabe definirla como:

La cual sería *independiente* entre *clasificadores* y por se puede demostrar que *la probabilidad de fallo de un ensemble* es la probabilidad de que la mitad o más de los *modelos base* se equivoquen y que resultaría ser igual (aplicándose el argumento de *independencia* y de *distribución bernoulli*):

Donde para la cota superior se ha usa la *desigualdad de Hoeffding* que muestra como claramente al hacer crecer el número de árboles, el *error de clasificación* puede ir reduciéndose.

Sin embargo lo anterior, que cabría hacer pensar que para lograr un gran modelo, entonces sólo hay que añadir modelos “como si no hubiera mañana”, choca con el supuesto de la *independencia* donde existen problemas tanto debidos a los datos a utilizar como a las variables a considerar sobre éstos, es por ello por lo que resulta recomendable en estos modelos:

* Disponer de un elevado conjunto de datos
* Disponer de un elevado conjunto de variables explicativas que idealmente tengan que ver con la explicada

De modo que se puedan crear suficientes modelos y que estos a su vez resulten lo más *diversos* posibles.

## 2.2.- El bagging

Como ha sido comentado, un modelo construido con un único *árbol* lo hace muy dependiente tanto del conjunto de entrenamiento como de los cortes derivados de éste y del algoritmo correspondiente para deducir sus ramas, así pues atendiendo a las ideas comentadas anteriormente, es cuando la estadística da el primer salto hacia el concepto de *machine learning* planteando reducir la varianza mediante la realización de un promedio de modelos disponibles a partir de un mismo conjunto de datos. En este contexto surge la técnica denominada *Bagging* o *Boostrap aGGregatING* que en realidad es una técnica sencilla introducida por Leo Brieman en 1984.

Un dibujo animado con letras

Descripción generada automáticamente con confianza media

Para ello se plantea el siguiente *algoritmo* de construcción:

* Paso 1: Se parte de un mismo conjunto de datos o *training* y se plantea construir un número *B* de nuevos subconjuntos de datos con el mismo número de observaciones que el original, para lo que se hará uso de un *muestreo aleatorio CON reemplazamiento*. A estos subconjuntos, cuyo número debe ser elevado se les denominará *muestras bootstrap*
* Paso 2: Sobre cada *muestra bootstrap* se entrena un modelo que puede ser un *arbol de regresión* o de *decisión*, aunque cabe considerar otras alternativas (siendo las más comunes las anteriormente citadas)
* Paso 3 o *ensemble*: Es aquí cuando llega el momento de mezclar las predicciones de cada uno de los modelos anteriores y que en el *Bagging* corresponde a realizar un simple promedio de cada uno de los modelos obtenidos

Con la formulación usada aquí, se obtiene un marco totalmente general y sencillo donde los modelos aquí denominados pueden ser “cualquier cosa” como *modelos de regresión logística*, *clasificadores bayesianos*, *funciones discriminante*, … Nótese además que el modo de realizar el *ensemble* no tiene que quedar reducido a un simple promedio, en el caso de decisiones discretas (con targets por ejemplo de tipo binario) cabe plantear predicciones haciendo uso de lo que se conoce como “voto de la mayoría”, que se comentará más adelante.

Sin embargo, tal y como se observa en el siguiente ejemplo, el uso del Bagging, no siempre garantiza mejores predicciones y es que hacer bagging por hacerlo puede resultar hasta cierto punto peligroso ya que aunque la elección de las *muestras bootstrap* garantiza cierta diversidad en los datos, sin embargo al hacer uso de todas las variables, muchos de los modelos creados tendrán un elevado número de variables comunes y por tanto las relaciones de interdependencia es de esperar que aumenten impidiendo que se cumpla la hipótesis de modelos independientes comentada en el anterior apartado.

Además otro hecho que va a enfrentar el *bagging* y a partir de ahora, prácticamente todos los modelos que se van a desarrollar, es la *interpretabilidad* donde el crear y mezclar un elevado número de modelos, sin duda impide conocer el efecto claro e individualizado de una o varias variables en particular, por lo que va ser usual recurrir a concepto como el de *importancia de las variables* en donde no ahora se dará un valor absoluto que sólo tiene que ver si una *variable explicativa* resulta ser más o menos determinante en la *target* pero sin conocer realmente el signo de su efecto particular, por lo que en vez de parámetros o coeficientes, se van a obtener valores de importancia que se estiman de modo distinto según se tenga *regression* o *decision trees*:

* En los árboles de regresión la importancia de cada variable se mide a través de la disminución de energía, univariante de una función objetivo similar a la siguiente (con respecto a la variable objetivo sin segmentar):

Imagen que contiene objeto, reloj

Descripción generada automáticamente

Donde ahora existirán *m* cortes debido a la cantidad de árboles creados para una única variable y tras calcular estas cantidades por cada una de las variables intervinientes, se crea una ordenación relativa en decrecimiento con respecto al resto de las variables

* En los árboles de decisión el razonamiento es análogo sólo que se usará la correspondiente función de pérdida (entropía, índice de gini, …)

Ejemplo 04: Un bagging poco productivo

En el siguiente ejemplo que se encuentra en *ejemplos/ejemplo04.R* se observa con datos simulados, como con una *regresión lineal*  se tiene un error de de 170’764 pero sin embargo, cuando se aplica bagging con distintos número de árboles, el error anterior no se reduce de forma significativa y en algunos casos incluso aumenta

Imagen de la pantalla de un celular con texto e imagen

Descripción generada automáticamente con confianza media

Nota: No conviene aplicar el bagging de 10000 iteraciones porque dura unos 5 – 10 minutos

Si se observa la función, se nota que sólo se consideran modelos de 3 variables explicativas y por tanto la *independencia* no queda totalmente garantizada.

\_\_\_\_\_

En otro orden, cabe comparar ahora con datos reales (pero pocos) cómo sería el efecto de un *bagging* frente a un *decision tree*. Aunque es cierto que es conveniente una mayor cantidad de datos el siguiente ejemplo, a un nivel muy básico permite explicar varios hechos que se van a dar a lo largo de toda esta sesión

Ejemplo 05: Bagging versus decision trees, caso de pocos datos

En el código *ejemplos/ejemplo05.R* se hace uso del conocido dataset *glaucoma* donde existen hasta unas 62 *variables explicativas* y donde se plantea comparar, a nivel puramente de acierto, cómo funciona un *bagging* frente a un *decision tree* aunque se cuente con tan sólo unas 196 observaciones en total.

Lo primero que salta a la vista cuando se ejecuta el código es que se hace uso de la librería *randomForest* y en particular de la función *randomForest()*. En este caso esta función es equivalente a la del *bagging* del anterior ejemplo cuando se consideran todas las variables explicativas del dataset, cosa que se hace indicando 62 en el parámetro *mtry*.

Tras ejecutar esta función y la correspondiente a un *decision tree* se obtiene lo siguiente:

|  |  |
| --- | --- |
|  |  |

Ligeramente parece clasificar mejor el modelo *bagging* teniendo aciertos más ponderados en la diagonal principal siendo de 51/59 = 86’44% frente a 50/59 = 84’75%

Varios comentarios al respecto:

* Se está ante cantidades muy pequeñas de datos y por tanto una mejora de un 2% en la predicción puede deberse por temas puramente muestrales ¿Se mantendría igual ante cambios de composción del *training* y del *test*?
* En el *bagging* se está usando toda la información y el *algoritmo* anteriormente comentado que da lugar a predicciones complejas, mientras que el *decision tree* genera un modelo muy sencillo tal como el siguiente, dondo con 2 variables se obtiene un nivel muy similar al del *bagging* que no tiene ni por asomo, una representación como la siguiente, claramente interpretable:

Una captura de pantalla de un celular

Descripción generada automáticamente

\_\_\_\_\_

## Anexo I: La validación cruzada

La validación cruzada es una técnica, que dadas las capacidades de las computadoras actuales es conveniente utilizarla. Esta técnica permite solucionar alguna de las situaciones que van a tener lugar con el tipo de *algoritmos* que, incluyendo al *bagging*, se van a analizar a continuación:

* Es una técnica robusta que permite asegurar la fiabilidad de los valores obtenidos en las previsiones dado un determinado modelo pudiendo genera intervalos de variación de las principales *medidas de predictividad global* como por ejemplo el *gini*, la *roc*, el *eam*, …
* Ante modelos con arquitecturas complejas como estos, va a contribuir a crear metodologías de selección parámetros
* Permite controlar el *over-fitting* al hacer uso de criterios de optimalidad en la selección paramétrica, complementando la información que darán las *learning curves* que serán comentadas más adelante

Así pues, esta técnica consiste en:

* Dado un conjunto base (que suele ser el conjunto Training), dividirlo en n-partes con igual cantidad de observaciones cada una, de modo que se podrán realizar *n*-entrenamientos distintos
* Se comienza entrenando una estructura de modelo fijada con *n - 1* parte de las anteriores divisiones dejándose una parte que no se entrena para predecir
* Se mide el comportamiento en la parte anteriormente comentada que no se entrena
* Se repite el proceso anterior *n* veces variando la parte no entrenada (y usada como test de predicciones)

Un gráfico que explica bien la validación cruzada es el siguiente extraído directamente de <https://es.wikipedia.org/wiki/Validaci%C3%B3n_cruzada>

Gráfico

Descripción generada automáticamente

**Figura 2 Ilustración esquemática de una validación cruzada**

**Fuente:** [**https://es.wikipedia.org/wiki/Validaci%C3%B3n\_cruzada**](https://es.wikipedia.org/wiki/Validaci%C3%B3n_cruzada)

Como puede intuirse, en función de las partes en que se divida el *training*, del tamaño de éste y de la complejidad del *algoritmo* a utilizar, esta técnica puede requerir tiempo de computación y aunque es conocida desde los 80 – 90, no ha podido abordarse con asiduidad en grandes dataset hasta bien entrada la década del 2010 – 20 con librerías específicas para tal fin tanto en *R* como en *Python*.

Ejemplo 06: Comparación de modelos mediante *validación cruzada*

Una aplicación habitual de esta técnica es la comparación efectiva de modelos tal y como se hace en el siguiente código que se encuentra en *ejemplos/ejemplo06.R*

|  |  |
| --- | --- |
|  |  |

Los patrones en ambos casos resulta similares ya que es el resultado de una poda bajo los mismos datos en un mismo árbol. En este caso el promedio de aciertos y la variabilidad de éstos en el modelo con más variables parece indicar que la elección de elegir una estructura más compleja parece la indicada sin embargo, todavía queda realizar una última prueba que puede arrojar más luz sobre la elección final ¿Sabría indicar cuál es?

\_\_\_\_\_

# **3.- RANDOM FOREST Y BOOSTING MODELS**

Con el *random forest* se da un paso más, aunque se mantiene la misma idea del *bootstrapping*, en cambio no se deja crecer el árbol a todo el conjunto de las variables, si no que se limita su crecimiento a un sub-conjunto de variables explicativas, de modo que si el número de predictores es , entonces el número de variables que se debería escoger debería estar en torno a para evitar problemas de over-fitting. Claramente el número de variables a seleccionar sobre cada muestra bootstrapping va a depender de la composición de dichas muestras.

Por tanto, cabe entender un *random forest* como un *bagging* donde se limita el número de variables a elegir y de nuevo, en la mayoría de las librerías, los *modelos base* a considerar serán o *regression* o *decision trees*. Pero, ¿Por qué se consideran los anteriores tipos de *modelos base*?

* Lo modelos tipo *árbol* son un tipo de *clasificadores inestables* ante perturbaciones en los datos, por tanto en la búsqueda de la *diversidad* que requiere un *bagging* o ahora un *random forest*, se muestran idóneos
* La complejidad de los *árboles* puede ser controlada bien mediante la *poda*, bien mediante limitaciones a su crecimiento, por lo que se pueden obtener *clasificadores débiles* pero no demasiado débiles

Con la anterior idea en mente, lo que se selecciona en cada *muestra bootstrapping* es el conjunto de variables más importantes con lo que se consiguen árboles más distintos entre sí que con el procedimiento anterior, a esto se le denomina *des-correlacionar el bagging*, en el sentido de que se ofrece más grados de libertad en el momento de escoger (o más bien asignar importancias de) las variables que finalmente formarán parte del modelo. Así pues, la superioridad general del *random forest* sobre el *bagging* y otras técnicas va a radicar en:

* Posibilidad de elegir *árboles* diversos donde las variables más importantes serán más elegidas
* Aplicación, para que las variables “más probables” no acaben mermando la *independencia*, del algoritmo denominado *random subspace method* que trata de aplicar criterios de selección aleatoria de variables para permitir trabajar sobre conjuntos distintos de posibles *variables explicativas* a seleccionar sobre cada *muestra boostrap*
* Finalmente es habitual la aplicación de *pre-pruning* o podas consistentes en limitar el crecimiento de los árboles en *profundidad* (generalmente) evitándose la aplicación de técnicas *post-prunning* una vez se construye el *random forest*

La superioridad sobre un amplio elenco de algoritmos de los *random forest* puede verse en el siguiente paper del 2014

|  |  |
| --- | --- |
| Interfaz de usuario gráfica, Texto, Aplicación  Descripción generada automáticamente |  |

Ejemplo 07: *Bagging* vs *random forest*

En este caso se retoma el anterior ejemplo del *glaucoma* y se modifica adecuadamente para hacer una nueva comparativa mediante *ejemplos/ejemplo07.R* Haciendo uso de un *random forest* donde se limita el número de variables a 7, se llega a conclusiones algo más claras que en el caso anterior:

|  |  |
| --- | --- |
|  |  |

\_\_\_\_\_

La otra gran familia para construir *modelos de ensemble* son las denominadas *técnicas boosting* que vienen utilizándose desde principios de los 90 al plantearse Robert Shapire la siguiente cuestión:

“if a set of weak learnear can créate a single strong learner”

Al igual que los *random forest*, estas técnicas no sólo limitan su aplicación a la mejora de los árboles, si no que tienen un marco más general y pueden aplicarse a otros esquemas de modelización tal como se hace en *Benjamin H. et alt (2012)*. No obstante aquí se centra la atención en la aplicación de las técnicas de tipo árbol ya que la mayoría de las librerías de R y *python* están basadas en estos.

Así pues la principal diferencia que se encuentra en los *boosting* es que éstos se construyen secuencialmente necesitándose por tanto en cada nuevo árbol, la información del anterior.

Un *algoritmo* habitual en estos modelos va a seguir un esquema similar al siguiente:

1. Se inicializa el *algoritmo* entrenando un modelo sobre la muestra *training* tratando de minimizar el error a 0 Se igualan los residuos iniciales a los valores de la variable explicada:
2. Se realizan iteraciones del algoritmo del siguiente modo:

-Se ajusta un árbol con cortes al nuevo conjunto *training* derivado del paso anterior

-Se crea una versión del modelo, modificada por un factor de aprendizaje :

-Se actualizan los datos a partir de la actualización de los errores para la próxima iteración sobre el conjunto training:

Una vez iterados todos los árboles y ajustados los residuos el modelo final se construye aplicando la siguiente formulación:

Diagrama, Calendario

Descripción generada automáticamente

**Figura 3 Entrenamiento esquemático de un *boosting***

**Fuente:** [**https://medium.com/swlh/boosting-and-bagging-explained-with-examples-5353a36eb78d?**](https://medium.com/swlh/boosting-and-bagging-explained-with-examples-5353a36eb78d?)

Por tanto la idea del *boosting* es ir progresivamente mejorando el error mediante un ajuste más fino de las predicciones que se aplican en cada iteración. El *coeficiente de aprendizaje* se suele tomar bajo y la convergencia por tanto va a ser lenta. Así pues, los parámetros que van a determinar un modelo boosting van a ser los siguientes:

* El número de árboles : cuyo tamaño debe ser vigilado ya que si es muy grand se puede tener problemas de over-fitting
* El parámetro de aprendizaje cuyos valores típicos estarán entre 0.01 y 0.001
* El número de particiones o de cortes a realizar, donde un valor bajo igual 1 prodría ir bien en ocasiones, aunque se puede llegar hasta 4 o 5

Modelos de tipo *boosting* existen en la actualidad muchos, todos parten de una idea como la anteriormente expresada, pero introduciendo algunas variantes en la forma en la que los *modelos* o errores son actualizados o si se usan *funciones de pérdidas* en el *algoritmo* o no. Una de las variantes más potentes en la actualidad son la familia de *Extreme Gradient Boosting Models* o en sus siglas *XGBoost* donde se analiza a continuación un ejemplo con una cantidad de datos considerable. Para entenderlos mejor es recomendable la siguiente referencia:

<https://www.analyticsvidhya.com/blog/2016/01/xgboost-algorithm-easy-steps/>

Ejemplo 08: aplicación de un *xgboost* sobre datos realistas

En el código *ejemplos/ejemplo08.R* se aplica un *Xgboost* a un conjunto de datos de empresa proporcionados por la plataforma *Kaggle*. Se observa que existe:

* Una importante cantidad de parámetros
* Ciertas particularidades del tratamiento de datos (al ser usada la librería en *R* aquí considerada, siendo más fácil en *python*)
* Variables pre-construidas
* Finalmente se genera un modelo sobre una gran cantidad de datos y se obtiene el siguiente esquema de importancia de las variables:

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Con lo anterior cabe plantearse las siguientes preguntas:

* ¿Qué haría falta para que el ejemplo esté más completo y se pueda hablar de una fiabilidad real del modelo obtenido? ¿Podría defender ahora mismo este modelo?
* ¿Cómo interpretaría el anterior gráfico de importancias de las variables?
* ¿Podría comparar el XGboost obtenido anteriormente, con las mismas variables con un modelo de *regresión logística* y un *random forest*? Se recomienda para el caso del *random forest* con *R* hacer uso de la librería *ranger*.

\_\_\_\_

# **4.- MEZCLA GENÉRICA DE MODELOS**

En general el *Ensembling* es una técnica donde se combinan 2 o más algoritmos de modelización que se denominan *Learners*, aquí se sigue y se resume la línea descrita en <https://www.analyticsvidhya.com/blog/2017/02/introduction-to-ensembling-along-with-implementation-in-r/> que se puede consultar para una información más detallada.

Aunque hasta ahora se han combinado modelos idénticos, cabe considerar mezclar modelos distintos no sólo en variables sino también en *algoritmos* con el objetivo de obtener sistemas más robustos que puedan mejorar unilateralmente, a cada uno de los *algoritmos* / *modelos* por separado.

Hasta ahora, los principales tipos de ensembling son aquellos en las que las predicciones pueden combinarse mediante:

- *Promedio de las predicciones*: obtenidas, que suele aplicarse en modelos donde la *variable explicada* es continua. También puede aplicarse en modelos donde la *target* es categórica, en este caso lo que se promedia van a ser la salidas de las *probabilidades* asociada a cada valor de la categoría

- *Media ponderada*: Es un caso análogo al anterior, sólo que en vez de promediar por igual a todos los modelos, se asignan “*pesos*” a la *predicción* generada por cada *modelo*

- *Voto de la Mayoría*: En este caso, sÍ se está tratando una *variable target* de tipo *categórica*, por loi que se elige la *categoría* que se acuerda por la mayoría de los modelos involucrados

Visto lo anterior, falta el ¿Cómo generar los modelos en sí? Hasta ahora se han estudiado 2 técnicas que han sido descritas en la parte anterior:

* *Bagging*: Con sus 2 variantes y aplicada a *árboles de decisión* y *de regresión,* donde los modelos se agregan por un promedio simple
* *Boosting*: Que tiene múltiples variantes agregándose los modelos por una suma “suavizada”

Sin embargo cabe considerar más técnicas para tal fin y en algunos caso existen librerías adaptadas como la de *skelearn* de *python* que permite la construcción sencilla de los modelos que aquí se denomina como *voting classifiers* y que permite, como se muestra en el siguiente ejemplo mezclar modelos de distinta procedencia y aplicar un criterio final de predicción acorde a lo comentado anteriormente.

Ejemplo 09: *ensemble* y mezcla de modelos en python

En el código correspondiente a *ejemplos/programa09.py* se desarrolla un código que se ha extraído y modificado de:

<https://stackabuse.com/ensemble-voting-classification-in-python-with-scikit-learn/>

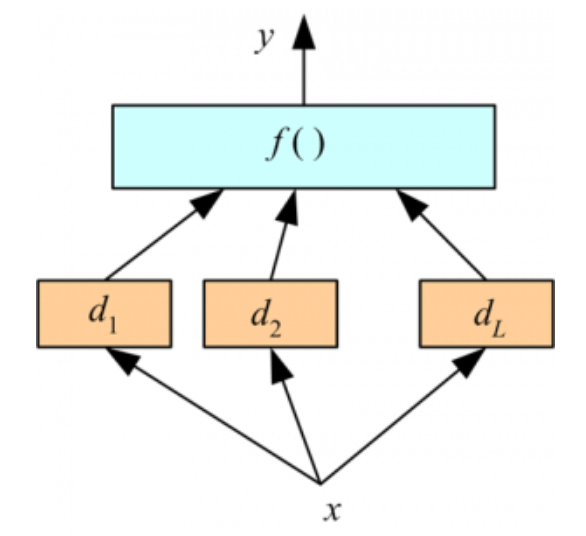
Donde se aplican varias de las técnicas vistas hasta ahora al conocido *dataset* denominado *titanic*. Obteniéndose los siguientes resultados:

|  |  |
| --- | --- |
| **Promedio básico** | **Voting Classifier** |
| Accuracy = 0.77 |  |
| **Bagging** | **Boosting** |
| Bagging 50 log: 0.7964  Bagging 50 tree: 0.8053  Bagging 100 rf: 0.7989  Bagging 100 extree: 0.7977 |  |

¿Qué esquema de modelización cabría ser elegido?

\_\_\_\_\_

Otra técnica que puede ser interesante es la denominada *stacking* de modelos. Es conveniente indicar que en este caso, la técnica suele ser algo más complicada de implementar que las anteriores y tiene determinados peligros que deben conocerse, por tanto es recomendable analizar bien si conviene o no meterse en la complejidad que conlleva. En suma consiste en establecer una estructura en *layers de modelos*. En cada una de éstas capas o *layers* hay modelos de diferentes tipología (en general se considera sólo 1 única capa) y después hay una “capa de mixtura” donde un *modelo* o función realiza la composición de los *modelos* del *layer* tal y como se observa en el siguiente esquema:



Ejemplo 10: hacia el stacking de modelos

En el siguiente código, basado en el que se encuentra en:

<https://medium.com/@brijesh_soni/stacking-to-improve-model-performance-a-comprehensive-guide-on-ensemble-learning-in-python-9ed53c93ce28>

Se dan las pautas para construir un stacking de modelos, atención especial debe prestarse en la construcción de los datasets del stacking, que si hay un *layer*, debe realizarse 2 etapas de construcción de estos datasets si se quiere elaborar un modelo que pueda ser debidamente testeado. El desarrollo se puede en contrar en *ejemplos/programa10.py* y se hace uso del conocido *dataset boston* donde se usa la siguiente familia de modelos base para el *stacking* y se hace la comparación frente a un simple *random forest*:

Texto

Descripción generada automáticamente

¿Cuál sale mejor frente a un *error absoluto medio relativo*?

\_\_\_\_\_

# **5.- INTERPRETABILIDAD DE MODELOS: LA LIBRERÍA LIME**

## 5.1 La librería LIME

Uno de los hándicaps que presentan los *modelos de ML* y *redes neuronales* más avanzados es su *interpretabilidad* de las predicciones que se obtienen bajo modelos avanzados.

Esta *interpretabilidad* resulta hasta cierto punto clave porque si el modelo va a ser decisivo en la aceptación o en el rechazo de un crédito, el usuario, vía *GDPR* va a tener derecho tener una explicación del porqué del rechazo de su crédito, ya que en ocasiones, como pasa por ejemplo en una hipoteca, pueden existir gastos por parte de los clientes como es el caso de las tasaciones, donde quien corre con los gastos al final es el cliente y ya un rechazo de una hipoteca, le incurre en un gasto de más de 200€.

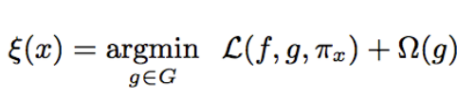
La *interpretabilidad* o la *explicabilidad* de los modelos implica definir una frontera clara de decisión comprensible en términos humanos y en este sentido existen librerías en los softwares open source como R o Python dedicas para tal fin.

Uno de las primeras librerías que surguieron en este sentido es la denominada *LIME* que trata de resolver el problema ofreciendo explicaciones locales a través de modelos subyacentes simples tal y como se detalla a continuación.

Por *interpretabilidad* lo que se desea, al menos para un determinado *elemento de la población* evaluado por un modelo, es dar una explicación clara y comprensible por el cuál se ha tomado una decisión de tipo automático o basada en un *algoritmo* por tanto se trata de responder a la pregunta ¿Por qué un *algoritmo* para un determinado *individuo* ha generado una determinada respuesta? Nótese que la anterior respuesta que podrá ser numérica o no puede se o bien la respuesta final para realizar una acción en el mundo real o una parte en la toma de decisión que finalmente desembocaría en la citada acción real.

Para dar respuesta a la anterior cuestión y tener un mayor insight en las relaciones establecidas, se han desarrollado desde hace algún tiempo librerías como la citada *LIME* cuyas siglas significan “*Local Interpretable Model-agnostic Explanations*”. Y que en sí es un framework que consta de los siguientes elementos:

* Un “explicador” (explainer) agnóstico ante diferentes modelos y consistente
* Una metodología para seleccionar un conjunto representativo con explicaciones consistentes con la lógica humana
* Un model-explainer a con las siguientes propiedades deseables:
  + Interpretabilidad: proveer de comprensión cualitativa de la relación entre las variables explicativas y explicada. En este sentido LIME utiliza representaciones entendidas por el humano como por ejemplo, en el caso de imágenes la presencia/ausencia de super-píxeles, en el caso de datos tabulares la ponderación de la combinación de columnas
  + Fiabilidad local: aunque las interpretaciones no se puedan extender a todos los datos de una población, al menos localmente sí verificar lo anterior. Bajo este contexto *LIME* minimiza lo que se conoce como la *Explanation Model Equation* que se representa como sigue:



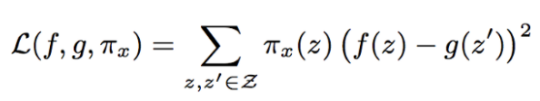
Algunas de los elementos de la anterior ecuación son los siguientes:

*f*: Ecuación del modelo a utilizar

*x*: Variables originales

*g*: Model-Explanation generada por un modelo sencillo de tipo lineal, decisión tree o conjunto de reglas

La ecuación anterior en términos generales, lo que viene a decir es que el modelo complejo, va ser finalmente explicado por uno sencillo que conserve, en términos locales propiedades similares al modelo global, es decir, se hace una “especie de interpolación” de modelos sencillos para explicar el modelo complejo. En el caso de interpretaciones lineales, el primer término de la anterior ecuación se concretaría en la siguiente expresión:



* + Model-Agnostic: no dependencia del modelo utilizado
  + Perspectiva Global: El usuario debe poseer como resultado una intuición global del funcionamiento del modelo

Actualmente aquí hay una clara línea de investigación abierta y aún no cerrada. Como se ha visto el mundo ML ha complicado los algoritmos tanto que al final se requieren de los “algoritmos de siempre” para poder dar explicaciones coherentes de los mismos.

Por tanto, técnicas como LIME y otras más de tipo heurístico son las que se están trabajando actualmente para poder dotar a modelos complejos de la parte de la interpretabilidad. A continuación, en el siguiente Business Case se muestra un caso sencillo de cómo se podría lograr hacer que un modelo, aparentemente complejo, pueda dársele, a nivel de cliente una interpretación suficientemente consistente.

Ejemplo 11: explicando un modelo complejo con *LIME*

En el código *data/programa11.py* se entrena un modelo tipo *boosting* y se pretende dotar de explicación a las decisiones, que toma el modelo para determinados clientes como la dada en el siguiente caso:

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

¿Cómo se interpreta la salida aquí mostrada?

\_\_\_\_\_

## 5.2 La librería SHAP

Otra técnica muy utilizada es la basada en los *valores SHAP* cuyas siglas significan *Shapley Additive exPlanations*.

*SHAP* es una técnica que permite descomponer la predicción de un modelo en contribuciones de las diferentes características, proporcionando una explicación clara y cuantitativa del impacto de cada una en el resultado.

Los valores *SHAP* se basan en la *teoría de juegos* y derivan del concepto de los valores Shapley. Estos son usados para distribuir de manera justa el "mérito" de una predicción entre todas las características de entrada de un modelo. *SHAP* tiene las siguientes propiedades clave:

* *Equidad:* Cada característica recibe el mérito que le corresponde.
* *Consistencia:* Si el modelo cambia para aumentar la importancia de una característica, su valor SHAP no disminuye.
* *Aditividad:* La predicción de un modelo es la suma de los valores SHAP de todas las características.

Estas propiedades aseguran que *SHAP* proporciona explicaciones coherentes y confiables para los resultados del modelo.

Para entender en qué se basan los anteriores valores conviene seguir el siguiente ejemplo:

Supóngase un caso muy sencillo en el que se quiere predecir si un cliente va a comprar o no un producto usando solo dos características: *edad* y *ingreso*:

Para calcular los *valores SHAP* de una característica, se necesita evaluar su impacto en todos los subconjuntos posibles de características.

Los subconjuntos posibles de características son:

1. Ninguna característica
2. Solo la característica "edad"
3. Solo la característica "ingreso"
4. Ambas características: "edad" y "ingreso"

El cálculo de los valores *SHAP* se hace comparando cuánto cambia la predicción cuando se incluye una característica en un subconjunto de características ya existente. Para cada característica, se calcula la *marginal* o la contribución de esa característica al agregarla. Por ejemplo en el caso anterior se podría tener las siguientes combinaciones

:

1. Sin características (valor base): P(∅)=0.5
2. Solo *edad*:
3. Solo *ingreso*:
4. *Edad* e *ingreso*:

Para calcular el valor SHAP de edad, calculamos su contribución en los siguientes dos casos:

1. Frente a ninguna característica:
2. Frente a la característica *ingreso*:
3. Finalmente se tiene que:

Análogamente se obtendría que:

Una propiedad clave de los valores *SHAP* es que la predicción final del modelo es la suma de los valores *SHAP* de todas las características. En este caso:

Nótese que lo anterior se hace para cada observación que tiene un valor en concreto, es decir, un individuo puede que tenga 35 años y unos ingresos de 1.500 y por tanto se tienen determinadas las anteriores cantidades exclusivamente para él, mientras otro podría tener 40 años y unos ingreos de 1.300

Por otro lado, para el caso de modelos sencillos como el anterior (tipor regresión logística), la interpretacion se tiene directamente y a nivel global pero en caso como el visto en el ejemplo 11, si se tiene.

Como se observa, este método resulta bastante más entendible que en el caso del *LIME* aunque si el modelo tiene un elevado número de variables puede resultar lento, aunque este problema también está presente y agravado en el caso del *LIME*, por lo que es preferible este método ya que evita la creación de modelos subyacentes como en el caso anterior y las interpretaciones son bastante más directas por lo general.

Ejemplo 12: explicando un modelo complejo con *SHAP* en variable continua

En el código que se encuetra en *ejemplos/programa12.py* se observa cómo el método es aplicable al conjunto Titanic con un modelo de 5 variables:

Gráfico, Calendario

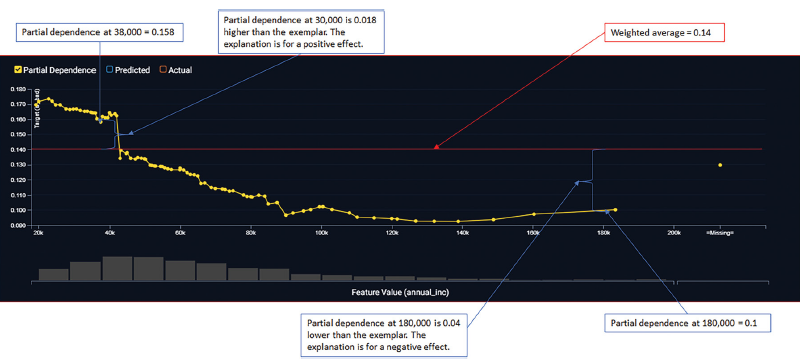
Descripción generada automáticamente

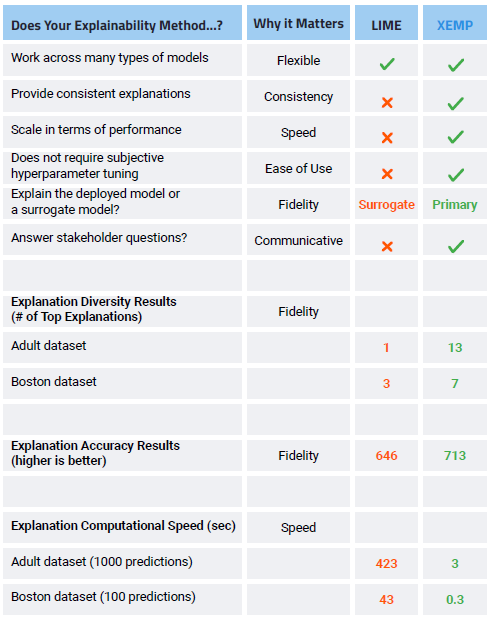
¿Qué variables es la más importante para salvar la vida?

\_\_\_\_\_

## 5.3 Otras opciones de interpretabilidad de modelos de analítica avanzada

Aparte del marco LIME y SHAP en la actualidad existen otros como el denominado XEMP, que tiene la gran ventaja de hacer uso de toda la información existente en los datos, junto con la particularidad de ser especialmente veloces y escalables, especialmente el marco XEMP que está incluso actualmente implementado en soluciones genéricas de construcción de modelos como es la ofrecida por <https://www.datarobot.com/>





Ejemplo 13: emulando XEMP con python

En *ejemplos/programa13.py* hay un ejemplo con la librería *dalex* donde se nota la rapidez en la toma de decisiones, aunque para el tema gráfico se recomienda usar jupyter ya que en *spyder* nose consigue obtener un gráfico como el siguiente basado en la explicabilidad de la primera observación del conjunto *test*.

Gráfico

Descripción generada automáticamente

Esta librería además ofrece interpretaciones globales como la mostrada en el siguiente gráfico tipo importancia de las variables:

Escala de tiempo

Descripción generada automáticamente

¿Qué sorprende entre las variables del ejemplo tomado?

\_\_\_\_\_

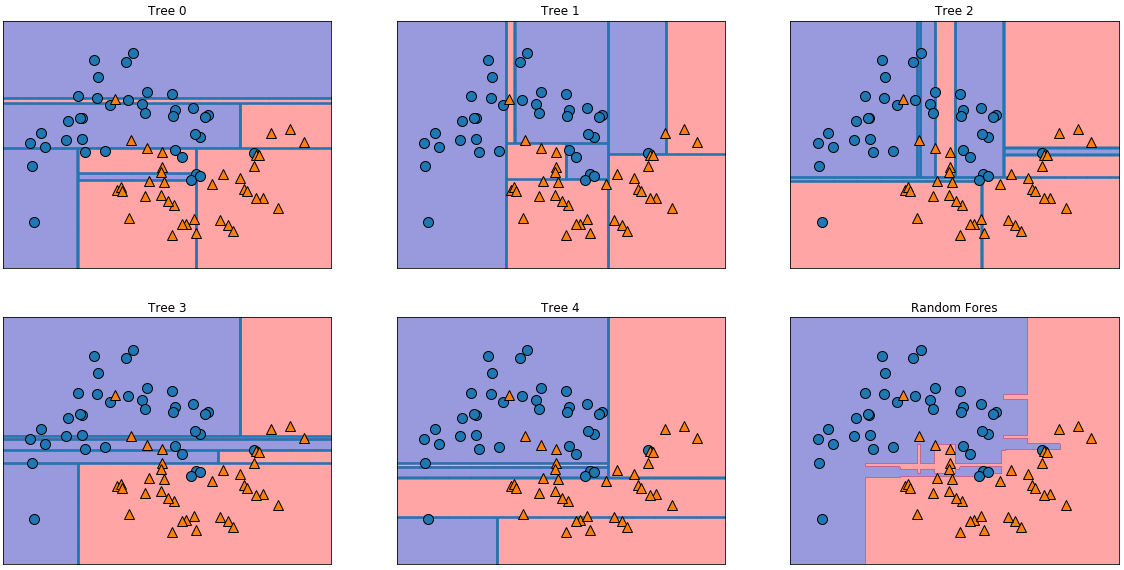
# 

# **6.- UN POCO DE ANALÍTICA AVANZADA CON PYTHON**

## El modelo de Random Forest. Un poco de detalle

Con el Random Forest se da un paso más, aunque se mantiene la misma idea del Bootstrapping, en cambio no se deja crecer el árbol a todo el conjunto de las variables, si no que se limita su crecimiento a un sub-conjunto de variables expliactivas, de modo que si el número de predictores es *p*, entonces el número de variables que se debería escoger debería estar en torno a *p* para evitar problemas de over-fitting. Claramente el número de variables a seleccionar sobre cada muestra bootstrapping va a depender de la composición de dichas muestras.

Con la anterior idea en mente, lo que se selecciona en cada muestra bootstrapping es el conjunto de variables más importantes con lo que se consiguen árboles más distintos entre sí que con el procedimiento anterior; a esto se le denomina *Des-correlacionar el Bagging*, en el sentido de que se ofrece más grados de libertad en el momento de escoger (o más bien asignar importancias de) las variables que finalmente formarán parte del modelo. En el siguiente ejemplo se analiza la construcción y algunos aspectos claves de este tipo de modelos con un conjunto de datos pequeños donde se analiza cómo se interrelacionan los árboles que se crean y la segmentación del espacio de las características que finalmente acaban realizando mostrándose porque cuando se aplican en la realidad, estos modelos llegan a ser una auténtica “caja negra” frente a los sencillos *Decision Trees* construidos al principio de esta sesión y que como se observa a continuación, permite hacer más eficiente, bajo un único modelo, lo que se muestra separadamente por 4 modelos promediando donde se observa que las zonas coloreadas atrapan más figura de su correspondiente “color”

.

**Segmentación espacial de un Random Fores y de sus componentes**

Ejemplo 14: plantando árboles

En el siguiente código que se encuentra en *ejemplos/*ejercicio14.py se extrae de forma independiente los *decision trees* de los que consta un *random forest* y se comprueba que usando estos modelos, se obtiene, promediando lógicamente el *ensemble*

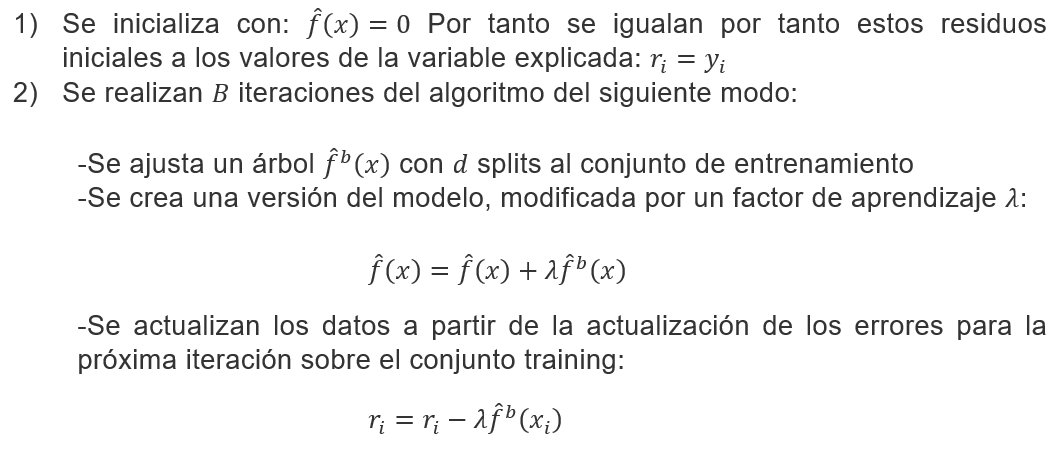
\_\_\_\_\_

Observación: Nótese que en las aplicaciones reales, las reglas a las que da lugar un Random Forest pueden llegar a se muy complejas y la interpretabilidad se reduce sólo a analizar la importancia de las variables, con lo que el grado de interpretabilidad de estos modelos frente a una regresión lineal es muy reducido

## El modelo de Extreme Gradient Boosting o xgboost

La idea de los *xgboost* es distinta, como se ha visto al caso de los *random forest*, en este caso, en cada iteración del algoritmo, es decir, cada vez que se hace una muestra bootstrap (muestra aleatoria con re-emplazamiento), se realiza simultáneamente una corrección de los errores cometidos en la etapa anterior con un algoritmo similar basado en la conocida regla del *gradiente descendente* acorde a la siguiente representación matemática:

-



Nótese que el paso clave es el 2, donde el *árbol* se ajusta sobre los *residuos* obtenidos de la anterior iteración, se des-hacen dichos *residuos* y se llega al árbol por diferencia y esa nueva estimación es la que se añade al paso anterior para ir “caminando sobre el gradiente” hacia unl mínimo.

Actualmente el algoritmo está sólidamente anclado en la librería *xgboost* y como puede observarse en el siguiente ejemplo, su utilización en la práctica resulta bastante sencilla aunque bajo esta implementación sea lenta su ejecución cuando hay muchos datos. Actualmente lo anterior está corregido con implementaciones más eficientes como la que se comenta a continuación.

Ejemplo 15: un *xgboost* en python

Con el código del *ejemplos/programa15.py* se observa cómo se ejecuta, un modelo *xgboost* de un modo sencillo y más directo en *python* que en R

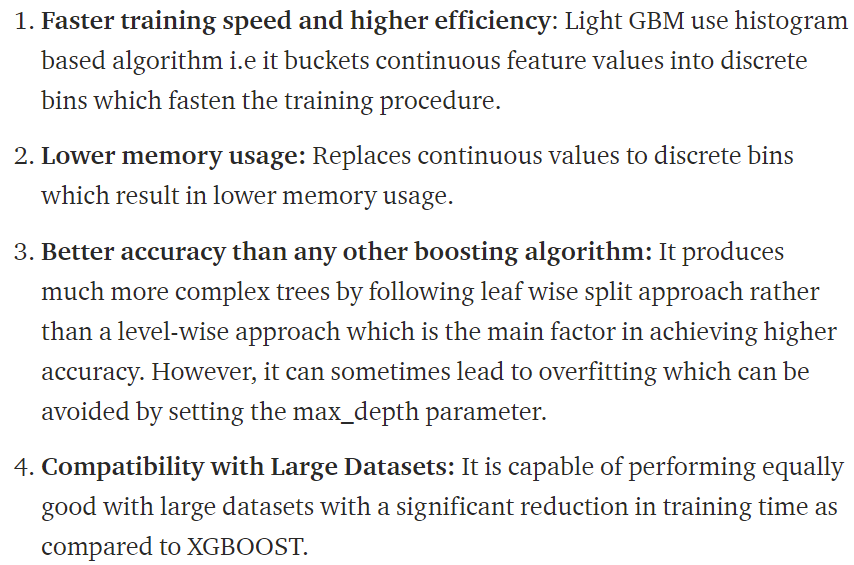
\_\_\_\_\_

## LightGBM un algoritmo de moda

Sin duda un algoritmo que lleva unos pocos años en la palestra es el LightGBM, se trata de un algoritmo liberado por IBM en 2017 donde para un mayor detalle se puede consultar en:

<https://towardsdatascience.com/lightgbm-vs-xgboost-which-algorithm-win-the-race-1ff7dd4917d>

Entre las principales características por las que ha llegado a ser tan popular están las siguientes:



En el siguiente ejemplo se compara un algoritmo *lightgbm* frente al anterior xgboost y observándose que con predictividades similares, con el primero, los tiempos de entrenamiento resultan ser inferiores. En la realidad sucede que en general, incluso a nivel de predictividad, los *lightgbm* pueden a menudo mejorar a los *xgboost*

Ejemplo 16: un *xgboost* vs *lightgbm*

Con el código del *ejemplos/programa16.py* se compara la velocidad y el acierto de estas 2 tipologías de modelos

\_\_\_\_\_

# **CONCLUSIONES FINALES**

Con los elementos vistos hasta ahora el alumno dispone de un amplio elenco de herramientas para modelizar.

En líneas generales se ha seguido una metodología general de modelización bastante estándar que puede ser generalizada con facilidad con los siguientes pasos resumen:

-Análisis exploratorio de datos

-Feature Engineering con o sin tratamiento de missing anterior o posterior

-Sampling

-Modelización con o sin Cross Validation

-Comparación de alternativas

-Medición del poder predictivo del modelo creado

Desde la primera sesión del módulo de Machine Learning hasta ahora la metodología no ha cambiado, por tanto a partir de ahora se puede “paquetizar” todo lo anteriormente comentado bajo un entorno de aplicación que puede hacerse con shiny (en el entorno R) o con flask (en el entorno python) donde se tenga en cuenta todas las pautas anteriores.

Como se describe en la actividad final del módulo, el alumno (o grupo de alumnos) tendrá que sistematizar todos los pasos anteriores y hacerlos accesibles a un tercero que tenga una idea media o baja, para ello es recomendable combinar tanto un conocimiento estadístico matemático como técnico para hacer no sólo una aplicación atractivo y de fácil uso, sino con cierta potencia, implementar una metodología similar a las que se han visto e incluso mejorarla será algo muy valorable que tiene a día de hoy una gran aplicabilidad en el mundo de la empresa.

El módulo cerrará con una breve introducción a las *Redes Neuronales* cuya evolución se conoce actualmente como *Deep Learning* y era prácticamente la última milla hace unos 2 o 3 años hata que aparecieron los *LLM*s que claramente están significando un claro salto del conocimiento actual que se tiene tanto de los modelos en sí como de la forma de trabajarlos.

# **BIBLIOGRAFÍA**

**Benjamin H.; Mayr A.; Robinzonov N. Schmidt M. (2012)** *Model-based Boosting in R A Hands-on Tutorial Using the R Package mboost* Technical Report Number 120, 2012 Department of Statistics University of Munich

**James G.; Witten D.; Hastie T. Tibshirani R. (2013)** *An Introduction to Statistical Learning* Springer ISBN 978-1-4614-7137-02

Ejemplo introductorio de una aplicación para la construcción de modelos: <https://github.com/FJROAR/RiskDevelopment/tree/master/CREDITSCORING>

Modelos LIME: <https://shiring.github.io/machine_learning/2017/04/23/lime>