El juego de explicación: Explicando los resultados de los modelos de Machine Learning mediante juegos cooperativos

TRABAJO FIN DE GRADO
Curso 2019/2020

UNIVERSIDAD
COMPLUTENSE
MADRID

FACULTAD DE CIENCIAS MATEMÁTICAS

GRADO EN INGENIERÍA MATEMÁTICA

Andrés Romero Herrera

Tutor: Juan Antonio Tejada Cazorla

Madrid, 17 de Julio de 2020

Resumen

Con la cada vez más rápida adopción de las técnicas de aprendizaje automático (Machine Learning) en importantes ámbitos de la sociedad, la falta de transparencia de los modelos se ha convertido en una gran preocupación dentro del mundo científico y de los potenciales usuarios.

Es por eso, que la interpretabilidad de los modelos es una línea de investigación muy popular en la actualidad. Entender el por qué de una predicción puede ser útil a la hora de confiar en un modelo. Habitualmente, los modelos más precisos son los más complejos de entender. En los últimos años, se han desarrollado nuevas técnicas para explicar las predicciones de estos modelos complejos, atribuyendo la predicción del modelo a las características de entrada, conocidos como métodos de atribuciones aditivas de las características. Entre ellos destacan los métodos basados en el valor de Shapley, que atribuye a cada característica, de forma justa y única, un valor acorde a su contribución en la predicción.

En este trabajo, veremos una serie de métodos para interpretar los modelos de machine learning. Empezamos estudiando algunos modelos básicos dentro del campo del machine learning interpretable, que entran dentro de la categoría de métodos de atribuciones aditivas de las características. Después, introduciremos el valor de Shapley y estudiaremos sus propiedades para dar paso al estudio de métodos basados en el valor de Shapley, y las formulaciones del juego cooperativo resultantes de cada método.

Palabras clave: *Machine Learning, Interpretabilidad, Modelos de caja negra, Valor de Shapley*

Abstract

With the increasingly rapid adoption of Machine Learning techniques in significant areas of society, the lack of transparency in Machine Learning models has become a major concern in the scientific world and possible users.

For that reason, interpretability in machine learning models is a popular line of research at present. Understanding the reasons behind predictions could be useful in order to trust a model. Most accurate models are usually the most complex to understand. Recently, new techniques have been proposed to explain a machine learning model's prediction by attributing it to the corresponding input features, known as additive feature attributions methods. Popular among them are methods based on Shapley value, which assigns each feature, in a fair and unique way, a value according to its contribution in the prediction.

In this work, we will see several methods to explain machine learning models. We begin by studying some basic methods within the field of interpretable machine learning, which fall into the category of additive feature attributions methods. We then introduce the Shapley value and its properties, which lead us to studying different methods based on Shapey value, and the resulting game formulations for each method.

Keywords: *Machine Learning, Interpretability, Black-box models, Shapley Value*

Índice general

Índice de figuras

Capítulo 1

Introducción

Coches que se conducen solos, asistentes que traducen instantáneamente de un idioma a otro o sugerencias de compra personalizadas. Complejas tareas que antes eran una quimera son hoy posibles gracias a las técnicas de 'Machine Learning'.

El término 'Machine Learning', fue definido por primera vez por Arthur Samuel en 1959 como un "campo de estudio que da a los computadores la capacidad de aprender sin estar programados explícitamente." Formalmente, se considera una subdisciplina de la Inteligencia Artificial, donde una máquina es capaz de extraer patrones a través de los datos gracias a modelos matemáticos, computacionales, y estadísticos. Esta rama ha tomado bastante protagonismo en los últimos años debido al aumento de la capacidad de computación y al boom de los datos. Las técnicas de Machine Learning son fundamentales para tratar los problemas de Big Data. Es por ello, que ha surgido un notable afán por la mejora y creación de nuevos algoritmos con menor error en su respuesta, sin importar la complejidad del mismo, y dando lugar, la mayoría de las veces, a un algoritmo del que no entendemos las razones por las que los resultados que proporciona son unos y no otros.

Estos modelos complejos, se están extendiendo rápidamente a diversas situaciones en el mundo real, como diagnósticos médicos o detección de crimen. Dado que estamos hablando de decisiones importantes, impera la necesidad de encontrar formas de interpretar, aunque sea de manera aproximada, lo que está haciendo un modelo. Consecuentemente, la interpretación y explicación de predicciones individuales se está volviendo cada vez más importante.

Recientemente, han aparecido un conjunto de técnicas que permiten, con mayor o menor detalle, entender qué está haciendo un modelo de machine learning. Una forma común de interpretar las predicciones son métodos basados en atribuciones de características, donde a cada característica se le asigna un valor en proporción a la contribución de esa característica a la predicción.

Por otro lado, la teoría de juegos se encarga de la modelización y análisis matemático de situaciones de conflicto, total o parcial, en las que intervienen dos o más decisores racionales (jugadores), de cuya interacción depende el resultado final de la situación (juego). Lo que cada jugador obtiene, no sólo depende de su propia decisión, sino que también depende de las decisiones tomadas por el resto de participantes.

Se distingue, fundamentalmente, entre dos tipos de juegos: Juegos cooperativos y Juegos no cooperativos, dependiendo de si los jugadores pueden o no comunicarse entre sí y tomar acuerdos vinculantes. En particular, en los juegos cooperativos se contempla la posibilidad de que los jugadores formen coaliciones. A su vez, los juegos cooperativos se clasifican en Juegos con utilidad transferible (juegos TU) y Juegos sin utilidad transferible (juegos NTU), dependiendo de que la utilidad pueda o no ser representada por algún medio de cambio que sea perfectamente transferible de un jugador a otro.

El problema central de la teoría de juegos cooperativos es el reparto de beneficio entre

los jugadores que forman la coalición. En este sentido, la Teoría de Juegos ofrece distintas "soluciones" en forma de conceptos de solución, entendiéndose por concepto de solución para juegos TU un mecanismo de selección (regla) que asigne a cada posible juego un conjunto de propuestas de reparto.

Llegados a este punto, cabe preguntarse la posibilidad de explicar las predicciones de los modelos de machine learning mediante juegos cooperativos, donde los jugadores sean las características del modelo de machine learning y el beneficio que se tiene que repartir entre jugadores se corresponda con la predicción del modelo.

En los últimos años ha habido un aumento del uso de métodos de atribuciones de características, destacando entre ellos, los métodos basados en el valor de Shapley.

El valor de Shapley es un método de distribución de riquezas en la teoría de juegos cooperativos. Para cada juego cooperativo se asigna un único reparto (entre los jugadores) del beneficio total generado por la coalición de todos los jugadores. El valor de Shapley se caracteriza por una colección de propiedades deseables o axiomas, que han hecho, que los métodos basados en el valor de Shapley estén emergiendo como la mejor aproximación de atribuciones de características.

El objetivo del trabajo es entender las distintas formas de lograr interpretabilidad en machine learning para predicciones individuales y analizar los métodos basados en el valor de Shapley para entender las distintas formulaciones del juego de estos métodos.

El trabajo se organiza de la siguiente manera. En el subcapítulo 1.1, se recogen las nociones básicas de machine learning y se introduce el problema de interpretabilidad, así como la importancia de interpretar predicciones individuales. En el capítulo 2, se introducen distintas técnicas de machine learning para lograr interpretabilidad (Cristoph Molnar, 2020), que entran en la categoría de métodos de atribuciones de características, que se usarán de una forma directa o indirecta en el siguiente capítulo. El capítulo 3, introduce conceptos básicos de la teoría de juegos cooperativos y el valor de Shapley (Shapley 1953), y estudia varios métodos de atribuciones de características basados en el valor de Shapley (Merrick and Taly 2019): IME (Strumbelj and Kononoenko 2010), KernelSHAP (Lundberg and Lee 2017) y TreeSHAP (Lundberg, Erion and Lee 2018). Por último, en el capítulo 4, se presentará todas las conclusiones extraídas de este trabajo junto a las posibles líneas futuras. Para ilustrar estas ideas, presentaremos dos casos de estudio para explicar las predicciones de los modelos de datos extraídos de la UCI de Machine learning, descritas en el capítulo 5.

1.1. Problema de la interpretabilidad en Machine learning

En este capítulo se hace una revisión de los conceptos básicos de Machine learning; además, introduciremos el problema de la interpretabilidad. Por último, veremos como interpretar las predicciones individuales, que dará paso a los siguientes capítulos donde presentaremos métodos para explicar estas predicciones.

Fundamentos básicos de Machine Learning

El aprendizaje automático (Machine Learning) es una disciplina científica del ámbito de la Inteligencia Artificial que crea sistemas que aprenden automáticamente. Aprender en este contexto quiere decir identificar patrones complejos en millones de datos. La máquina que realmente aprende es un algoritmo que revisa los datos y es capaz de predecir comportamientos futuros. Automáticamente, también en este contexto, implica que estos sistemas se mejoran de forma autónoma con el tiempo, sin intervención humana. Por ejemplo, para predecir el precio de una casa, la máquina aprenderá patrones de ventas pasadas.

Un algoritmo es un conjunto de instrucciones que sigue la maquina para realizar una tarea.

Los algoritmos de machine learning se dividen en 3 categorías:

- *Aprendizaje supervisado*, donde los algoritmos cuentan con un aprendizaje previo basado en un sistema de etiquetas asociadas a unos datos que les permiten tomar decisiones o hacer predicciones.

- *Aprendizaje no supervisado*, los algoritmos no cuentan con un conocimiento previo, se enfrentan al caos de datos con el objetivo de encontrar patrones que permitan organizarlos de alguna manera.

- *Aprendizaje por refuerzo*, en el que el objetivo es que un algoritmo aprenda a partir de la propia experiencia. Esto es, que sea capaz de tomar la mejor decisión ante diferentes situaciones de acuerdo a un proceso de prueba y error en el que se recompensan las decisiones correctas.

En este trabajo nos centraremos principalmente en el aprendizaje supervisado. El objetivo, es construir un modelo predictivo que transforme las características en una estimación de la variable objetivo. Este modelo vendrá definido por f. Cuando realizamos modelos de machine learning nos podemos encontrar con datos tabulares, de tipo texto o imágenes, nos centraremos en los datos tabulares.

Usaremos un lenguaje propio de machine learning. El conjunto de datos viene dado por una matriz de dimensiones $n \times (m+1)$, siendo n, las filas, el número de instancias o casos, y $m+1$ el número de columnas de la matriz, donde m es el número de características, y la última columna se corresponde con la variable objetivo.

Una instancia es una fila del conjunto de datos. Los términos registro, observación, caso y ejemplo también serán utilizados como alternativa a instancia. Una instancia es un vector que contiene los valores de las características para la i-ésima fila, representado por x^i, y por y^i, si conocemos el objetivo, para $i = 1, ..., n$.

Las características (features) o variables son los datos de entrada que se utilizan para la predicción. Una característica es una columna de la matriz de datos X, de dimensiones $n \times m$, representada por x_j para $j = 1, .., m$. Cada x_j representa el valor de todas las instancias para la j-ésima característica. El valor de la característica j en la instancia i se representará por x_j^i.

El tipo de característica depende de su naturaleza. Distinguimos entre característica cuantitativa, que toma valores numéricos, y característica cualitativa, donde la característica toma valores de un conjunto finito de clases.

Dentro de la característica cualitativa, también llamada categórica, encontramos dos tipos: de tipo nominal, por ejemplo, el estado civil de una persona, la característica toma los valores: soltero/a, casado/a, viudo/a; y de tipo binaria, donde solo toma dos valores, como 'True' y 'False', o '0' y '1'.

La variable objetivo (target), o variable dependiente, es el resultado asociado a las características observadas, la información que el modelo aprende para poder predecir, y vendrá definifa por y, o por y^i para una instancia particular.

La variable objetivo también varía en función de su tipo. Depende del tipo de target que tengamos nos encontramos con dos tipos de problemas:

- *Problemas de Regresión*: El objetivo es predecir un valor real. El ejemplo de la predicción del precio de una casa es un ejemplo de problema de regresión.

- *Problemas de Clasificación*: El objetivo es predecir a qué clase pertenece cierta muestra entre un grupo de clases previamente establecidas. En este tipo de problemas suele ser habitual diferenciar entre la clasificación binaria (donde solo existe dos clases) y la clasificación multiclase (donde existe más de dos clases).

En este trabajo nos encontraremos con un problema de regresión y otro de clasificación binaria, y con características de tipo numérico y categórico.

La predicción es lo que el modelo pronostica que vale la variable objetivo, que se denotará por $\hat{y}$ o $\hat{f}(x)$.

Para medir la precisión del algoritmo, utilizamos una puntuación de acierto o una función de pérdida que tratamos de minimizar. La función de pérdida es un método para evaluar como el algoritmo modela los datos dados. Si las predicciones se desvían demasiado de los objetivos observados, la función de pérdida será alta y no tendremos un algoritmo preciso.

Para los problemas de regresión utilizaremos normalmente el coeficiente de determinación, también llamado R^2. Este coeficiente mide la proporción de varianza del objetivo que es explicada por el modelo.

Para los problemas de clasificación utilizaremos una medida de precisión, que medirá el porcentaje de casos que el modelo ha acertado.

Para desarrollar un modelo de machine learning, partimos de una base de datos de entrenamiento (cuanto más datos, mejor), que debe contener el resultado observado que queremos predecir. El algoritmo de machine learning detectará patrones para predecir los resultados. Una vez creado el modelo, podemos validarlo con datos nuevos (datos de validación).

Las técnicas de machine learning traen consigo grandes ventajas. Un modelo de machine learning puede realizar una tarea de forma más rápida y fiable que un ser humano. Replicar un modelo de machine learning puede ser simple y rápido. Para una persona, afrontar esa tarea con una gran cantidad de datos, puede ser un trabajo costoso y largo. Sin embargo, el principal problema de machine learning, es que, lo que pasa dentro de los algoritmos con los datos está escondido en modelos que son cada vez más complejos. Estos modelos complejos en los que no entendemos qué ocurre dentro de sus algoritmos son los denominados modelos de caja negra (black-box models). Se caracterizan por no revelar sus mecanismos internos.

Se necesita una gran cantidad de datos para crear una red neuronal, y no hay forma de entender el modelo entero. Los modelos con mejor rendimiento predictivo son normalmente conjuntos (ensembles) de varios modelos que no pueden ser interpretados, aunque cada modelo pudiera ser interpretado individualmente.

Interpretabilidad

Entendemos el término interpretabilidad, en machine learning, como el grado en el que una persona puede entender la causa de una decisión. Cuanto más interpretable sea un modelo, más fácil será comprender por qué ha hecho una determinada predicción.

Si un modelo de machine learning tiene un buen rendimiento predictivo, ¿por qué no confiamos en el modelo e ignoramos por qué se hizo esa predicción?

Esto depende del impacto del modelo en la vida real. Para modelos que se usan en ambientes de bajo riesgo, donde un error no tiene consecuencias serias, por ejemplo, un sistema de recomendaciones de películas, no se le da importancia al por qué de una decisión, basta con saber que tiene una buena capacidad de predicción. Sin embargo, para situaciones con impacto en la vida real, como en detección de fraude bancario, donde las personas tienen que tomar decisiones basándose en los resultados, no vale con saber cuál es el resultado, el modelo debe explicar también como llega a hacer esa predicción. Entender el por qué de una decisión puede ayudar más a saber sobre los datos y sobre cómo el modelo puede fallar.

El problema es que una sola medida de evaluación del algoritmo, cómo el porcentaje de acierto a la hora de clasificar, es una descripción incompleta para un modelo aplicado en la vida real. Las personas necesitan razonar las cosas antes de tomar decisiones, es por eso, que saber entender e interpretar un modelo es de gran importancia.

Además de la necesidad de razonar las cosas y la curiosidad humana, la interpretabilidad es

también requerida a nivel legal, debido a la ley de protección de datos europea, donde se recoge el derecho a la explicación, en la que, un interesado tiene derecho a la información sobre la lógica involucrada que determina el resultado obtenido.

Los métodos para la interpretabilidad de machine learning pueden ser clasificados siguiendo varios criterios. En función de cuándo se consigue la interpretabilidad en el modelo, encontramos métodos intrinsecos y métodos post-hoc. Los métodos intrínsecos son aquellos modelos de machine learning considerados interpretables debido a su simple estructura, como los árboles de decisión. Los métodos post-hoc son métodos interpretables que se aplican a un modelo después de haberlo entrenado. Los métodos post-hoc son los utilizados normalmente para explicar las predicciones de los modelos de caja negra.

Los métodos post-hoc pueden dividirse también en modelos específicos y modelos agnósticos. Los modelos específicos pueden aplicarse solamente a un conjunto en particular de modelos, ya que consideran su estructura para explicar el comportamiento del modelo. Los modelos agnósticos pueden aplicarse a cualquier tipo de modelo, ignoran el modelo f concreto, y explican el comportamiento basándose en las características de entrada y el resultado.

Los métodos de interpretación pueden explicar el comportamiento del modelo entero o una predicción individual. Esto da lugar a dos tipos de interpretabilidad: interpretabilidad global e interpretabilidad local.

¿Cómo realiza las predicciones un modelo? Para explicar un resultado global se necesita entender cómo funciona el modelo, cómo toma decisiones basándose en todas las características. Una interpretabilidad global nos ayuda a entender la distribución del objetivo en relación a las características.

En el trabajo, veremos técnicas para obtener predicciones individuales interpretables. Llamaremos explicaciones a estas predicciones interpretables.

Interpretación de predicciones individuales

Una explicación relaciona los valores de las características de un registro con su predicción. Las explicaciones deben tener una serie de propiedades.

Tienen que ser interpretables, esto es, la relación entre las características y el objetivo tiene que ser comprensible. La interpretabilidad tiene que tener en cuenta las limitaciones que puede tener una persona a la hora de interpretarlo. Por ejemplo, si cien características contribuyen de forma significativa a una predicción, no tiene sentido esperar que alguien entienda por qué se ha hecho esa predicción. Las explicaciones deben ser fáciles de entender, por lo que tiene sentido presentar una explicación con un número pequeño de características.

Las explicaciones han de ser consistentes, no pueden diferir mucho entre distintos modelos entrenados en el mismo conjunto de datos y que producen predicciones similares. También tienen que ser estables, las explicaciones tienen que ser similares para registros parecidos.

Las explicaciones deben ser fiables localmente, es decir, deben aproximar bien la predicción de un modelo. Tiene que explicar cómo actúa el modelo en la proximidad del registro de interés. Que una explicación sea fiable localmente, no implica que lo sea globalmente, pues características que sean importantes globalmente pueden no serlo en un contexto local, o viceversa.

Además de explicar las predicciones individuales, añadir una perspectiva global del modelo puede ayudar a confiar en el modelo.

Muchos de los métodos para interpretar predicciones individuales entran dentro de la categoría de métodos de atribuciones aditivas de características. Esta clase de métodos explica el resultado del modelo como una suma de valores reales atribuidos a cada característica.

Afrontaremos el problema de la interpretación de predicciones individuales por medio de atribuciones aditivas a sus características.

Definition 1. Métodos de atribuciones aditivas de características: *Sea* $f : X \to \mathbb{R}$ *un*

modelo que transforma un m-espacio vectorial X de características en predicciones de valores reales. Las atribuciones aditivas de características para $f(x)$ con $x = (x_1, ..., x_m) \in X$ están compuestas por un valor de referencia ϕ_0 y atribuciones de características $\phi = (\phi_1, ..., \phi_m)$ correspondientes a las m características, tal que $f(x) = \phi_0 + \sum_{j=1}^{m} \phi_j$.

Las atribuciones aditivas de características son atribuciones cuya suma es la diferencia entre el valor predicho por el modelo $f(x)$ y un valor de referencia ϕ_0. Este valor de referencia ϕ_0, normalmente se corresponde con el resultado medio.

En este trabajo dividiremos estos métodos en dos tipos: métodos de atribuciones aditivas basadas o no en el valor de Shapley, concepto de la teoría de juegos cooperativos que explicaremos más adelante.

Capítulo 2

Interpretabilidad aditiva de algunos modelos particulares

En este capítulo estudiaremos algunos modelos concretos en los que explicaremos sus predicciones por mediode atribuciones aditivas de las características. Por un lado, presentaremos los modelos interpretables necesarios para comprender algunas de las metodologías de interpretación. Por otro, introduciremos LIME, como ejemplo de modelo agnóstico para explicar las predicciones de un modelo de caja negra.

2.1. Modelos interpretables

La mejor forma de conseguir interpretabilidad es usar solo un conjunto de algoritmos que crean modelos interpretables.

Estos modelos son interpretables intrínsicamente, métodos que no requieren de aproximaciones complicadas para entender qué nos dicen sobre un problema. La lista de modelos interpretables de machine learning crece continuamente, por lo que hay un gran número de modelos de este tipo. En este trabajo explicaremos dos de ellos: los modelos de regresión lineal y de árboles de decisión con el fin de ayudar a comprender el objetivo de la interpretabilidad aditiva.

2.1.1. Regresión lineal

La regresión lineal es un modelo estadístico que predice el resultado como una suma ponderada de las características de entrada.

Se utiliza para modelar la relación entre una variable escalar dependiente y y una o más variables explicativas denotadas por $x = (x_1, ..., x_m)$. El modelo de regresión lineal ajustado para un registro x se puede escribir de la siguiente forma:

$$y = \beta_0 + \beta_1 x_1 + ... + \beta_m x_m + \epsilon \tag{2.1}$$

donde x_j es el valor de la característica, con $j = 1, ..., m$. β_0 es conocido como el intercepto. Cada β_j representa el peso correspondiente a la característica j. El ϵ es el error que cometemos, es decir, la diferencia entre la predicción y el valor real del resultado.

Existen varios métodos para estimar los pesos óptimos. El método de mínimos cuadrados es el que se utiliza normalmente para encontrar los pesos que minimicen las sumas de cuadrados de las diferencias entre los datos reales y las predicciones:

$$\hat{\beta} = arg \min_{\beta_0, ...\beta_m} \sum_{i=1}^{n} (y^i - \beta_0 - \sum_{j=1}^{m} \beta_j x_j^i)^2.$$

La predicción para una instancia x^i es una suma ponderada de las m características:

$$\hat{y}^i = \hat{\beta}_0 + \hat{\beta}_1 x_1^i + ... + \hat{\beta_m} x_m^i \tag{2.2}$$

2.1.1.1 Interpretación de los pesos del modelo

La interpretación de los pesos en el modelo de regresión lineal, donde los datos están estandarizados, es la siguiente:

- Característica númerica: Un incremento de la característica x_k en una unidad estándar incrementa la predicción $\hat{y}$ en β_k unidades cuando el resto de las características permanecen igual.
- Característica categórica: Cambiar la variable x_k de la categoría de referencia a otro tipo incrementa la predicción $\hat{y}$ en β_k unidades cuando el resto de las características permanecen igual.
- Intercepto: Es el peso para la 'característica constante', que siempre es 1. Cuando los datos están estandarizados, el intercepto representa el valor medio cuando todas las características toman su valor medio, o en el caso de categóricas, su característica de referencia.

2.1.1.1.1 Ejemplo

Para aplicar el modelo de regresión lineal a nuestros datos de alquiler de bicis, hay que estandarizar las características numéricas y codificar las categóricas, de forma que el intercepto represente la predicción media de los datos. Además, discretizamos la característica 'hr', que se relaciona de forma no lineal con la variable objetivo, en 4 intervalos.

Una vez realizadas estas transformaciones, ajustamos el modelo de regresión lineal a nuestros datos. La figura 2.1 muestra el peso estimado para cada característica, el valor del intercepto, la suma de los errores cuadrados (SSE), la suma de los cuadrados totales (SST) y el valor de R^2.

feature	weight	feature	weight
temp	40,847	mnth_NOV	-1,560
hum	-20,371	mnth_OKT	11,904
windspeed	-0,825	hr_0-5	-105,703
season_FALL	6,490	hr_12-17	55,367
season_SPRING	-25,217	hr_18-23	28,159
season_SUMMER	-5,285	holiday_HOLIDAY	-12,315
yr_2011	0,000	weekday_FRI	2,422
mnth_APR	2,608	weekday_MON	-1,230
mnth_AUG	-12,252	weekday_SAT	6,821
mnth_DEC	3,133	weekday_SUN	3,450
mnth_FEB	-11,262	weekday_THU	-6,304
mnth_JAN	-11,759	weekday_TUE	-1,759
mnth_JUL	-25,405	workingday_NO WORKING DAY	-3,911
mnth_JUN	7,355	weathersit_GOOD	4,247
mnth_MAR	-11,798	weathersit_MISTY	-0,063
mnth_MAY	33,893	weathersit_RAIN	-33,882

name	value
Intercept	146,802
SSE	32550,985
SST	43638,784
R-Cuadrado	0,254

Figura 2.1: Resultados de Regresión Lineal

Como ejemplo de interpretación de los pesos en las características categóricas, nos fijamos en la característica 'weathersit'. La característica de referencia es la que no aparece, ya que al tomar las demás el valor 0, es esta la que toma el valor 1, en este caso 'weathersit=STORM/SNOW'. Cuando el tiempo es 'good', y todas las demás características permanecen igual, se predicen 4,2 bicis más.

2.1.1.2 Interpretación de una predicción individual

¿Cuánto ha contribuido una característica de un registro en su predicción? Queremos expresar una predicción individual como un método de atribuciones aditivas de características, tal que:

$$\hat{f}(x^i) = \phi_0 + \sum_{j=1}^{m} \phi_j^i$$

Para responder esto, tendremos que identificar el valor de referencia y las contribuciones de las características. En este caso, ϕ_0 se corresponde con el intercepto, que es la predicción media de los datos, $\phi_0 = \beta_0$. El valor de las contribuciones ϕ_j^i se calculan multiplicando el peso de la característica j por el valor de la carcaterística j en x^i, es decir, $\phi_j^i = \beta_j x_j^i$.

2.1.1.2.1 Ejemplo

Para hallar las contribuciones ϕ_j^i, hay que multiplicar el pesos de cada característica por su valor en el registro x^i. Primero, calculamos la predicción, $\hat{f}(x^i)$, que realiza el modelo para la instancia 11728. Esta instancia consta de las características mostradas en la figura 2.2. El valor de la característica 'hr' será 18-23.

feature	value
season	SUMMER
yr	2012
mnth	MAY
hr	21
holiday	NO HOLIDAY
weekday	TUE
workingday	WORKING DAY
weathersit	MISTY
temp	22,08
hum	65
windspeed	19,001

Figura 2.2: Características de la instancia 11728

Multiplicando los valores de las características por sus respectivos pesos, que vemos en la figura 2.1, obtenemos las contribuciones. La suma del intercepto, β_0, predicción media de los datos de entrenamiento, más las contribuciones, $\beta_j x_j^i$, de cada característica es igual a la predicción del modelo $\hat{f}(x^i)$.

$$\hat{f}(x^i) = \beta_0 + \sum_{j=1}^{m} \beta_j x_j^i,$$

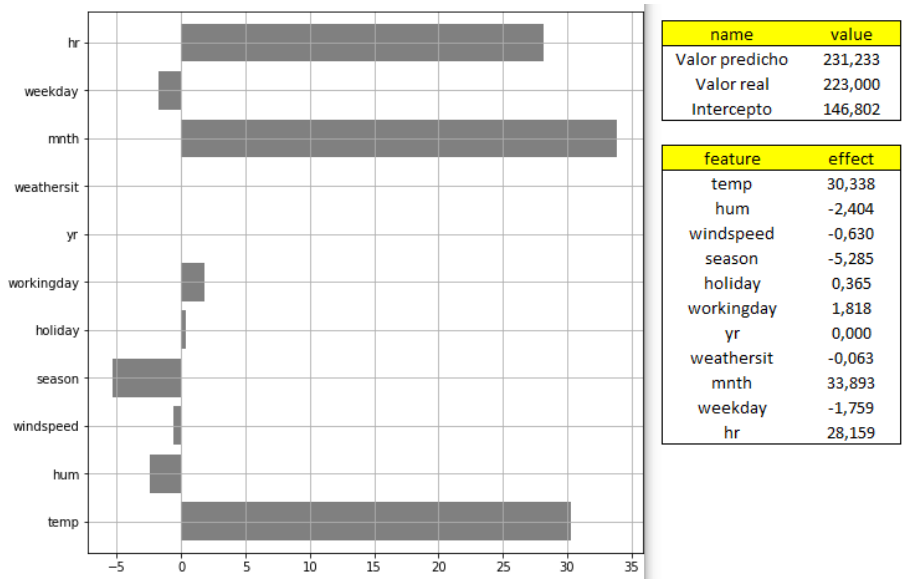
Vemos los resultados obtenidos para la instancia 11728 en la figura 2.3.

Podemos comparar la distribución de las contribuciones de las características para todo el modelo, multiplicando los pesos de las características por todos sus respectivos valores en la matriz de datos, $\beta_j x_j$. Para ilustrar esta idea, representamos en la figura 2.4, la distribución de las contribuciones de las características en un diagrama de cajas. Las cruces rojas, representan las contribuciones de la instancia 11728, calculadas anteriormente.

2.1.1.3 Ventajas y desventajas

Ventajas:

- La linealidad del modelo hace que la estimación sea simple, y que los resultados sean fáciles de interpretar. Esta es una de las razones por las que este método es tan utilizado en diferentes campos cuantitativos.



name	value
Valor predicho	231,233
Valor real	223,000
Intercepto	146,802

feature	effect
temp	30,338
hum	-2,404
windspeed	-0,630
season	-5,285
holiday	0,365
workingday	1,818
yr	0,000
weathersit	-0,063
mnth	33,893
weekday	-1,759
hr	28,159

Figura 2.3: Regresión Lineal: Contribuciones instancia 11728

- La regresión lineal tiene buen rendimiento cuando los datos son separables linealmente.
- Existen modelos lineales dispersos(con pocas características) que dependen de pocas características, con el fin de obtener mejor interpretabilidad, como el método Lasso o Step Forward.

Desventajas:

- Cuando dos características están muy correladas, este modelo tiene problemas al estimar los pesos.
- Sólo puede representar relaciones lineales por lo que cada relación no lineal tiene que ser transformada manualmente.
- El algoritmo sobresimplifica la realidad por lo que la capacidad predictiva no es siempre buena.
- Sufre con valores atípicos y es propenso al overfitting.

2.1.1.4 Software

Para obtener los pesos de la regresión lineal se ha utilizado la función 'LinearRegression' de la librería de scikit learn en python.

2.1.2. Árboles de decisión

Un árbol de decisión en Machine Learning es una estructura de árbol similar a un diagrama de flujo. El nodo superior en un árbol de decisión se conoce como el nodo raíz. Representa el total de los datos que se están analizando y los divide en dos o más subconjuntos. La división se realiza mediante ramas que indican si se cumple una condición o no sobre una determinada variable.

Cada nodo interno, llamado nodo de decisión, representa un subconjunto de datos, que se vuelve a dividir, formando nuevos nodos de decisión. Este proceso se repite iterativamente hasta llegar al nodo final, llamado nodo hoja, que representa el resultado.

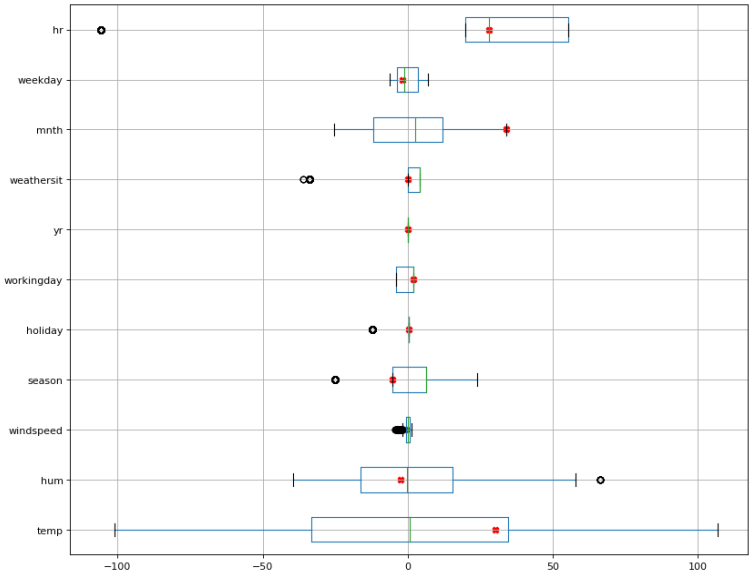


Figura 2.4: Distribución de las contribuciones de las características y contribuciones instancia 11728

Existen varios algoritmos para construir árboles de decisión, pueden diferir en estructura, criterios para buscar particiones, cuándo parar las divisiones y cómo estimar los resultados. Nos centraremos en el algoritmo CART, caracterizado por realizar particiones binarias.

Sea S el conjunto de nodos terminales/hoja, R_s el conjunto de instancias pertenecientes al nodo hoja s, $I\{x \in R_s\}$, la función indicador que nos devuelve 1 si el registro x está en R_s, y 0 en otro caso, y sea $\bar{y}_s$, el valor medio de la variable objetivo, del conjunto de instancias que caen en el nodo s. Podemos describir la predicción para un registro x como:

$$\hat{y} = \hat{f}(x) = \sum_{s=1}^{S} \bar{y}_s I\{x \in R_s\} \tag{2.3}$$

Cada registro cae exactamente en un nodo hoja (=subconjunto R_s), donde la predicción es el valor medio de la variable objetivo de registros clasificados en ese nodo, $\bar{y}_s$.

Para elegir los subconjuntos, el algoritmo distingue entre regresión y clasificación.

En el caso de regresión, el algoritmo toma la característica y encuentra la división de los datos que minimice la varianza de la predicción $\hat{y}$. En el caso de clasificación, el algoritmo toma la característica y encuentra la división que minimice el índice Gini, valor que mide la impureza de los nodos.

Después de calcular el mejor corte para cada característica. el algoritmo selecciona la característica que dé una mejor partición en términos de la varianza o del índice Gini, y añade esa partición al árbol. El algoritmo continua con este proceso de búsqueda y división recursivamente en los nuevos nodos creados hasta que alcanza un critero de parada.

2.1.2.1 Interpretación de una predicción individual

Sea $c^i = (n_{j_1}, n_{j_2}, ..., n_s)$ el camino que sigue la predicción de una instancia, desde el nodo raíz, n_{j_1}, hasta el nodo hoja, n_s. Podemos explicar una predicción individual descomponiendo el camino que sigue el registro de interés.

Para un registro que pasa por un nodo n_{j_k}, y que es dividido por la característica j, siguiendo su camino por $n_{j_{k+1}}$, podemos definir la contribución de la característica j como la diferencia

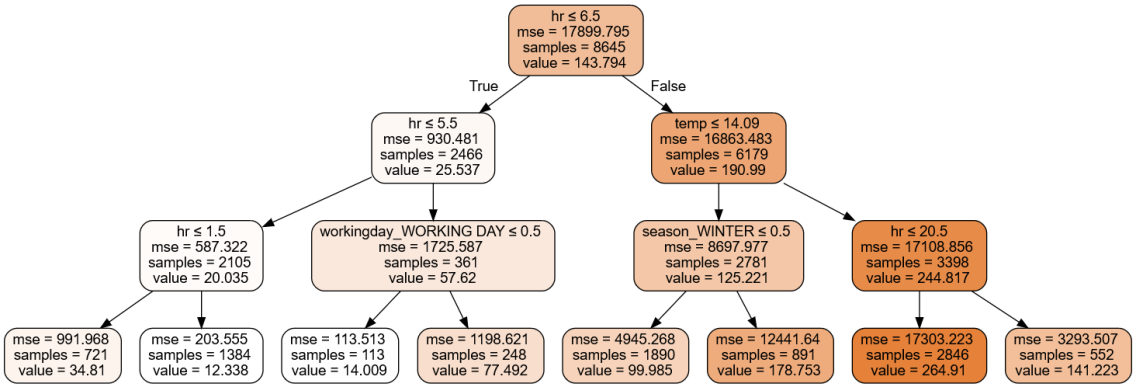


Figura 2.5: Árbol de decisión para el problema de regresión con profundidad máxima=3

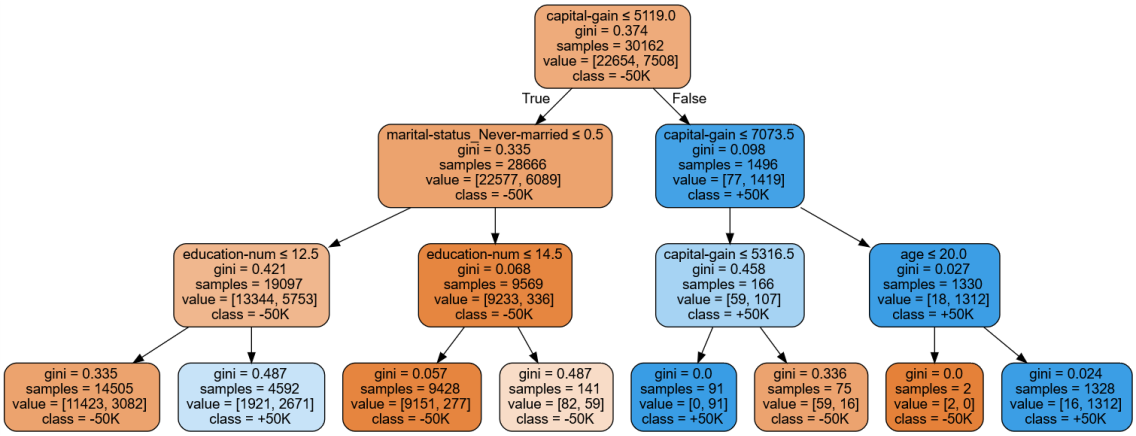


Figura 2.6: Árbol de decisión para el problema de clasificación con profundidad máxima=3

de valor medio del nodo hijo $\bar{y}(n_{j_{k+1}})$ y el nodo padre $\bar{y}(n_{j_k})$. Teniendo en cuenta que una característica puede aparecer más de una vez en el camino, definimos las contribuciones de características como:

$$\phi_j^i = \sum_{\substack{k=1 \\ Si\ j_k=j}}^{s-1} (\bar{y}(n_{j_{k+1}}) - \bar{y}(n_{j_k}))$$

donde la contribución de una característica en una predicción es la suma de sus contribuciones a lo largo del camino recorrido por el registro hasta alcanzar el nodo hoja.

Podemos expresar la predicción individual, $\hat{f}(x^i)$, de nuestro modelo como un modelo aditivo:

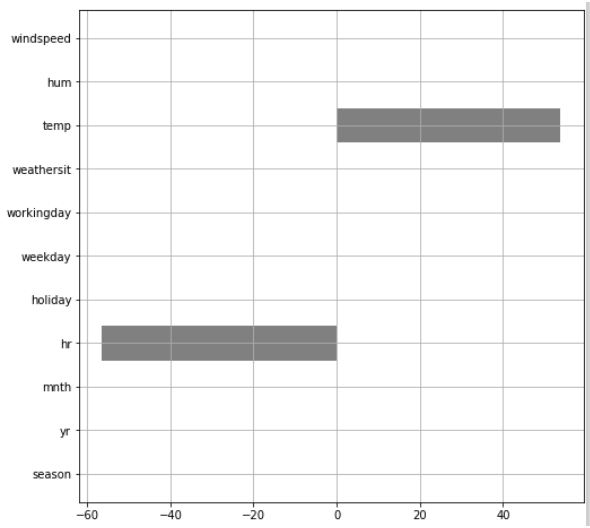
$$\hat{f}(x^i) = \phi_0 + \sum_{j=1}^{m} \phi_j^i$$

donde $\phi_0 = \bar{y} = \bar{y}(n_{j_1})$, es el valor medio de la variable objetivo para todos los datos (nodo raíz), y ϕ_j^i son las contribuciones de las características a la predicción.

2.1.2.1.1 Ejemplo

En el caso de regresión, la contribución de una característica es la diferencia entre el valor del nodo padre (conjunto de datos que esa característica divide) y del nodo hijo (siguiendo el camino de la predicción).

Tomando de nuevo como ejemplo la instancia 11728 con las características descritas en la figura 2.2, calculamos la predicción del árbol de decisión y obtenemos los resultados que aparecen en la figura 2.7.



name	value
Valor medio de referencia	143,794
Valor Predicho	141,223
Valor Real	223,000

feature	contribution
season	0
yr	0
mnth	0
hr	-56,398
holiday	0
weekday	0
workingday	0
weathersit	0
temp	53,827
hum	0
windspeed	0

Figura 2.7: Árbol de Decisión: Contribuciones para la instancia 11728

Observamos, teniendo en cuenta el árbol de la figura 2.5, que para esta instancia las únicas características que contribuyen son 'hr' y 'temp'. Para la característica 'hr' la suma de sus 2 contribuciones: 190,99-143,794+141,223-244,817=-56,398, nos da su contribución final. Mientras que la contribución de la característica 'temp': 244,817-190,99 = 53,827. La suma del valor medio $\bar{y} = 143,79$, valor del nodo raíz, más las contribuciones de las características es igual a la predicción del modelo:

$$\hat{f}(x^i) = \bar{y} + \sum_{j=1}^{m} \phi_j^i$$

2.1.2.1.2 Ejemplo

Para el caso de clasificación trabajamos con probabilidades. Esto es, si tenemos un nodo con 5 pacientes con cancer y 10 sanos, la probabilidad de que sea sano, es 2/3. Por lo que las contribuciones de las características se entiende como la variación de probabilidad del nodo padre al nodo hijo, siguiendo el camino de la predicción.

Si queremos predecir cuanto han contribuido las características en la predicción del registro número 9118 de los datos de test, con las características descritas en la figura 2.8.

La predicción y las contribuciones para esta instancia aparecen en la figura 2.9.

Para explicar esta predicción, replicamos el camino seguido por el árbol hasta llegar al nodo hoja calculando las contribuciones de las características de la siguiente manera: El nodo raíz tiene una probabilidad de 0,249 (7508/30162) de que una persona gane más de 50K al año, este nodo, es dividido por la característica Capital-gain. En nuestro ejemplo, la persona que estudiamos tiene una ganancia de capital nula por lo que vamos al nodo hijo de la izquierda donde la probabilidad de que se gane más de 50K al año es de 0,212 (6089/28666). Luego, la contribución de la característica capital-gain es: $0,212 - 0,249 = -0,037$.

Para la siguiente característica que encontramos en el camino de la predicción, 'Marital-status', obtenemos una contribución: $5753/19097 - 6089/28666 = 0,089$. Y para la última característica que aparece en el camino, 'education-num', obtenemos una contribución: 2671/4592−

feature	value
age	54
workclass	Private
fnlwgt	312631
education-num	15
marital-status	Married
relationship	Wife
race	White
sex	Female
capital-gain	0
capital-loss	1887
hours-per-week	50

Figura 2.8: Características para la instancia 9118

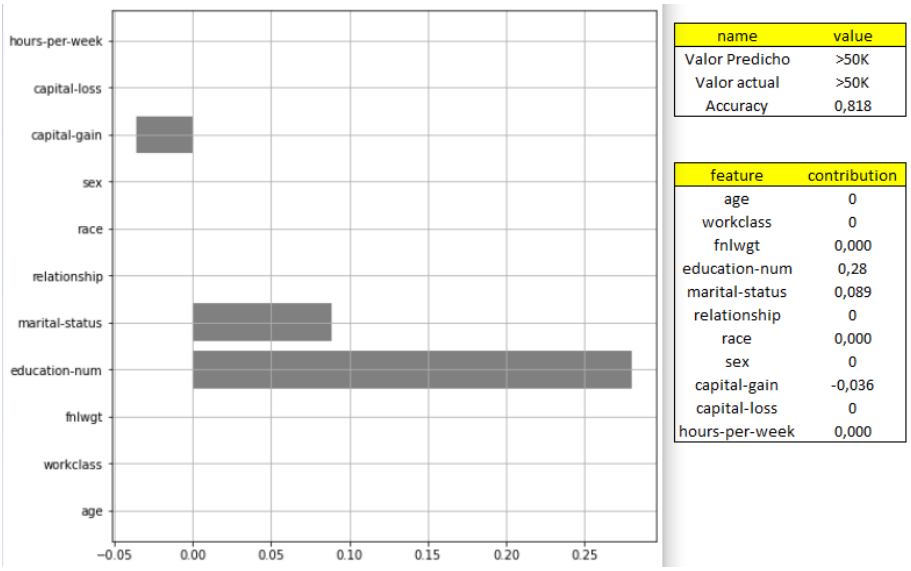


Figura 2.9: Árbol de Decisión: Contribuciones para la instancia 9118

$5753/19097 = 0,28$.

La probabilidad de que una persona gane más de 50K al año en el nodo raíz representa la probabilidad media en los datos de entrenamiento. La suma de la probabilidad media más la suma de las contribuciones de las características tiene que ser igual a la probabilidad de que una persona gane más de 50K al año en el nodo hoja, que en nuestro ejemplo es de 0,582.

$$\hat{f}(x^i) = \bar{y} + \sum_{j=1}^{m} \phi_j^i$$

Hemos trabajado en los dos problemas con árboles de profundidad máxima igual a 3, para facilitar la visualización de los árboles en este trabajo, sin embargo, árboles más profundos tendrían mejor precisión.

2.1.2.2 Random Forest

Random forest consiste en un conjunto de arboles de decisión combinados con bagging, esto es, cada modelo se entrena con subconjuntos del conjunto de entrenamiento. Estos subconjuntos se forman eligiendo muestras aleatoriamente (con repetición) del conjunto de entrenamiento.

La interpretación de este modelo no es sencilla, pues al ser un conjunto de árboles, calcular las contribuciones de las características para una predicción individual sería un trabajo muy costoso. Este modelo es un ejemplo de modelo de caja negra (black-box) con una gran capacidad predictiva. En lo que sigue de trabajo, trataremos de explicar las predicciones individuales de este modelo, mediante distintas técnicas post-hoc.

2.1.2.3 Ventajas y desventajas

Ventajas:

- A diferencia de los modelos lineales, localizan y representan bastante bien las relaciones no lineales entre características.
- Los árboles de decisión son fáciles de entender e interpretar, ya que imitan el pensamiento a nivel humano.

Desventajas:

- Falla cuando se encuentra con relaciones lineales.
- Tiende al overfitting.
- Pequeños cambios en los datos de entrada pueden causar un gran cambio en la estructura del árbol, son bastante inestables..

2.1.2.4 Software

Para obtener los árboles de decisión se han usado las funciones 'DecisionTreeRegressor' y 'DecisionTreeClassifier' de la librería scikit learn de Python.

2.2. Modelos agnósticos

La mejor explicación de un modelo simple es él mismo; se representa a él mismo y es fácil de entender. Sin embargo, para modelos complejos (como las redes neuronales o random forest), no podemos usar el modelo original como su propia explicación, ya que no son fáciles de entender.

Los métodos agnósticos al modelo son aquellos que separan la explicación del modelo de machine learning.

2.2.1. LIME

El método Local Interpretable Model-agnostic Explanation o LIME, es un método local aplicable para explicar predicciones individuales, obteniendo la contribución de cada característica sobre la predicción de una observación individual.

LIME permite interpretar qué ocurre con las predicciones de un modelo black-box cuando los datos varían, esto es, a partir de una observación que se quiere interpretar, se generan nuevas observaciones vecinas permutadas modificadas a partir de la original, tomando valores de una distribución normal con media y varianza respectivas de cada característica.

Sobre estas nuevas observaciones, se entrena un modelo interpretable (modelo local sustituto, como regresión lineal o árboles de decisión) que debe aportar una buena aproximación de la predicción de manera local (no tiene por qué ser una buena aproximación global).

En notación matemática, la explicación de una observación x viene dada por:

$$\text{explicación}(x) = arg\min_{g \in G} \mathcal{L}(f, g, \pi_x) + \Omega(g) \tag{2.4}$$

La explicación del modelo para un registro x es el modelo g (por ejemplo, regresión lineal) que minimiza la pérdida de fidelidad (error) $\mathcal{L}$, que mide cómo de cerca está la explicación de la predicción del modelo original f (por ejemplo, random forest), mientras que la complejidad del modelo $\Omega(g)$, se mantiene baja (pocas características). G es la clase de modelos interpretables (por ejemplo, todos las posibles regresiones lineales). La medida de proximidad π_x define cómo de grande es la vecindad alrededor del registro x que se quiere explicar.

Debido a la versión actual de LIME en Python, nos centraremos en modelos lineales dispersos como explicaciones, por lo que limitamos G, familia de modelos interpretables, a la clase de modelos lineales.

En la práctica, este método solo optimiza la parte de la pérdida:

$$\mathcal{L}(f, g, \pi_x) = \sum_{x \in X} [f(x) - g(x)]^2 \pi_x(x)$$

El algoritmo usado por LIME para calcular la explicación lineal de una predicción sigue el proceso descrito en el algoritmo 1.

Algorithm 1 Explicaciones lineales dispersas del modelo f, usando LIME, dada la instancia x, y número N de muestras a crear

```
Definimos el núcleo π_x, y la longitud K de la explicación
Z ← {}
for i = 1, ..., N: do
    #Se generan nuevas observaciones a partir de la original
    z^i ← remuestreo alrededor(x)
    #Se añade al conjunto de datos nuevos y se calcula la predicción y el
  peso del nuevo registro
    Z ← Z ∪ ⟨z^i, f(z^i), π_x(z^i)⟩
end for
#Se aplica el modelo lineal que proporciona explicaciones interpretables
w ← K-Lasso(Z, K) con instancias z^i, y f(z^i), como variable objetivo
```

Nosotros tenemos que definir la compejidad del modelo, por ejemplo, seleccionando un número máximo de características que un modelo linear debe usar, mediante Lasso, forward selection o 'highest weights'.

La anchura del núcleo (kernel width) determina lo grande que es la vecindad. Una anchura pequeña significa que una instancia debe estar muy cerca para influenciar al modelo local, mientras que una anchura grande significa que registros que estén lejos también tienen influencia en el modelo.

2.2.1.1 Interpretación de una predicción individual

Expresar el método LIME como un método de atribución aditiva de características es sencillo, pues el algoritmo nos proporciona directamente el intercepto de la regresión lineal más las contribuciones de las características. Es decir:

$$\hat{f}(x^i) \approx \hat{g}(x^i) = \beta_0 + \sum_{j=1}^{M} \phi_j^i$$

Esto quiere decir que la predicción local no tiene porque ser igual a la predicción del modelo black box. Además, la suma de las contribuciones más el intercepto será igual a la predicción local Veamos unos ejemplos donde aplicamos LIME a nuestros datos.

2.2.1.1.1 Ejemplo

En el primer caso, de regresión, aplicando previamente RandomForest a los datos, podríamos saber a que 5 características se deben la prediccion de la instancia número 11728 ya que LIME aporta explicaciones dispersas con pocas características. Sin embargo, como queremos comparar las atribuciones que cada método da a las distintas características, veremos la predicción explicada con todas las características. En la figura 2.10, podemos ver los resultados obtenidos con el método LIME para la instancia descrita en la figura 2.2.

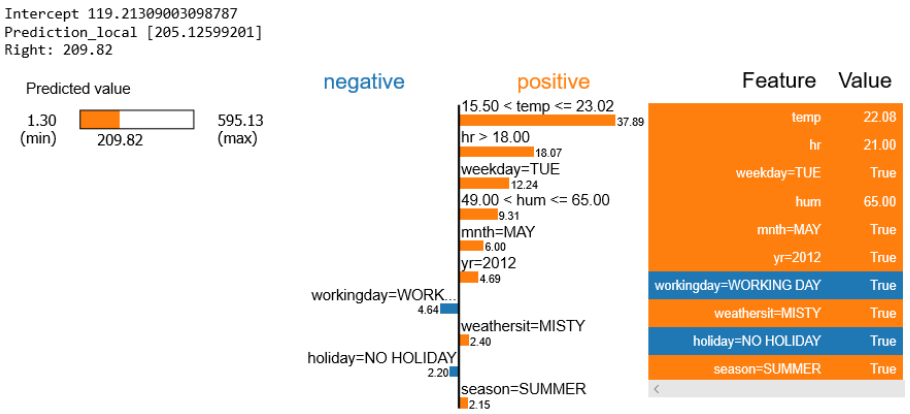


Figura 2.10: Resultados LIME instancia 11728

Arriba a la izquierda de la figura 2.10, vemos primero el intercepto, calculado a partir de la regresión lineal creada, el valor de predicción local que representa el valor predicho por la regresión lineal. 'Right' representa el valor que predice nuestro modelo de caja negra, RandomForest.

A la derecha, vemos la tabla con el listado de las características y loa valores que toma en la instancia de interés.

En el centro, aparece la contribución de las características al modelo, valores que se suman o restan al intercepto para obtener la predicción del modelo local.

Las explicaciones de LIME son contrastables. La forma de interpretar una característica es la siguiente: para una variable numérica, como la temperatura, cuando su valor no se encuentra entre 15.50 y 23.02, la predicción de alquier de bicis sería de 37,9 bicis menos de media.

La suma del intercepto, β_0, más las contribuciones de las características ϕ_j es igual a la predicción del modelo local g:

$$\hat{g}(x^i) = \beta_0 + \sum_{j=1}^{m} \phi_j^i$$

2.2.1.1.2 Ejemplo

Para el problema de clasificación, volvemos a aplicar el modelo de RandomForest los datos de adulto. Queremos saber cómo contribuyen las características a la predicción de la instancia descrita en la figura 2.8, y en que medida. Mediante LIME, obtenemos los resultados mostrados en la figura 2.11

La interpretación es similar a la anterior. A la izquierda nos encontramos con la predicción del modelo original y con la del modelo local. Se ve que el modelo lineal no se aproxima muy bien a la predicción del modelo.

A la derecha, los valores de las características seleccionadas por el modelo local para explicar la predicción. Y en la gráfica del centro, las contribuciones de las características al modelo.

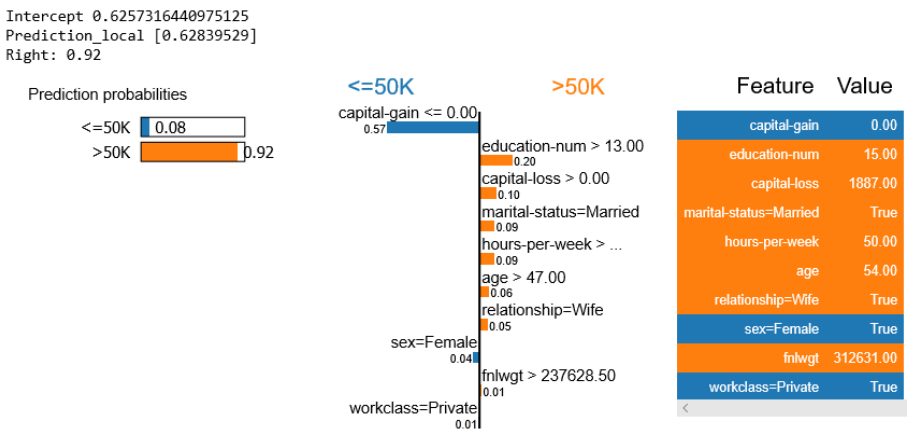


Figura 2.11: Resultados LIME instancia 9118

En este caso las contribuciones de las características son variaciones de probabilidad. La suma de las contribuciones, ϕ_j, más el intercepto, β_0, es igual a la predicción del modelo interpretable, $\hat{g}(x^i)$, que se aproxima localmente a la predicción del modelo f:

$$\hat{g}(x^i) = \beta_0 + \sum_{j=1}^{m} \phi_j^i$$

2.2.1.2 Ventajas y desventajas

Ventajas:

- LIME ofrece explicaciones dispersas, con pocas características, para una mejor interpretación de la predicción.
- Aunque en este trabajo nos centramos en datos tabulares, lime funciona también para texto e imágenes.
- Al igual que la regresión lineal, las explicaciones son contrastables.

Desventajas:

- La correcta definición del kernel es un problema en LIME, pues hay que ir probando distintas medidas y ser tu mismo el que vea si la explicación tiene sentido.
- Las explicaciones son inestables. Dos puntos similares pueden variar bastante.
- Las explicaciones no suelen llegar a alcanzar la predicción del modelo de caja negra, si no que se aproximan.

2.2.1.3 Software

Para la realización de los ejemplos se ha utilizado el paquete de LIME en Python.

Capítulo 3

Métodos de atribuciones aditivas basados en el valor de Shapley

En este capítulo se estudiarán una serie de métodos de atribuciones aditivas de las características basadas en el valor de Shapley. Para ello, primero introduciremos unas nociones básicas de teoría de juegos cooperativos y explicaremos el valor de Shapley.

3.1. Juegos Cooperativos

Los juegos cooperativos son aquellos que permiten cooperar a los jugadores formando coaliciones. Los jugadores de esa coalición actuarán buscando el máximo beneficio para la coalición, pudiendo establecerse o no restricciones a la hora de construir coaliciones y repartir los beneficios obtenidos al cooperar. Así, los juegos cooperativos en los que cualquier reparto de la recompensa entre los jugadores es posible, se conocen como juegos de utilidad transferible (UT). Mientras que aquellos en los que existen restricciones, se conocen como juegos de utilidad no transferible (NTU).

Sea $M = \{1, ..., m\}$ un conjunto finito y no vacío de jugadores. Llamamos coalición a cualquier subconjunto $S \subseteq M$. Denotaremos por 2^M al conjunto potencia del conjunto de jugadores, es decir, al conjunto de todos los subconjuntos de M, que representa a todas las posibles coaliciones de jugadores que se pueden generar.

Definition 2. Un juego cooperativo de utilidad transferible es un par (M, v), en el cual M representa el conjunto de jugadores participantes y, v la función característica que asigna un número real $v(S)$ a cada uno de los subconjuntos S de M,

$$v : 2^M \to \mathbb{R}.$$

Además, asumimos que $v(\emptyset) = 0$.

El valor asignado por la función característica determina el máximo beneficio que se pueden garantizar obtener, por medio de la cooperación, los jugadores que forman la coalición S.

A continuación se describen diversas propiedades que pueden cumplir los juegos cooperativos.

Un juego es **monótono** si $v(S) \leq v(T)$ para todo T y S subconjuntos de M tal que $S \subseteq T$. Esto es, cuantos más jugadores se unen a una coalición, mayor es el beneficio que la coalición genera.

Un juego se dice **superaditivo** si dadas dos coaliciones cualesquiera S, T disjuntas ($S \cap T = \emptyset$), se cumple que $v(S+T) \geq v(S)+v(T)$. Es decir, el beneficio de formar una coalición es mayor que la suma de los beneficios de trabajar por separado.

Un juego es **convexo** si y sólo si para todo $j \in M$

$$v(S \cup \{j\}) - v(S) \leq v(T \cup \{x_j\}) - v(T), \quad \forall S \subseteq T \subseteq M \setminus \{j\}.$$

Esto es, la contribución de un jugador a una coalición es creciente al aumentar el tamaño de la coalición. Cabe resaltar que la superaditividad es una condición necesaria para la convexidad.

Se dice que un juego es **simétrico** si y sólo si, para cualesquiera dos coaliciones S y T de M con la misma cardinalidad, se tiene que $v(S) = v(T)$. Es decir, los juegos simétricos modelan situaciones en las que la valía obtenida por una coalición depende de la cantidad de jugadores que posee, sin importar quiénes son.

Una de las tareas fundamentales de la teoría de juegos cooperativos es estudiar cómo dividir el valor total, $v(M)$, conseguido por la coalición entre los jugadores involucrados en el juego v.

Definition 3. Un vector de pagos ϕ es un elemento de $\mathbb{R}^m$ cuya j-ésima coordenada representa el pago correspondiente en el juego cooperativo del j-ésimo jugador. Se denotará por $\phi(v)$ a

$$\phi(S) = \sum_{j \in S} \phi_j, \qquad \forall S \subseteq M.$$

Un vector de pagos tiene racionalidad individual si y sólo si $\phi_j \geq v(\{j\})$ para todo $j \in M$. Además, se dice que tiene racionalidad de grupo si y sólo si $\phi(v) \geq v(S)$ para todo $S \in 2^M$. Se dirá también, que un vector de pagos es eficiente si y sólo si $\phi(v) = v(M)$.

Con un vector de pagos racional individualmente, cada jugador garantiza que obtiene al menos lo que obtendría si juega por si solo, mientras que un vector con racionalidad de grupo, garantiza que cada una de las coaliciones consiga al menos lo mismo que si todos sus integrantes jugaran por separado.

Sea $G = \{v | v : 2^M \to \mathbb{R}, v(\emptyset) = 0\}$ el conjunto de juegos cooperativos con conjunto de jugadores M. Definimos una solución para los juegos operativos en G como una función $\phi : G \to \mathbb{R}^M$ que asigna a cada juego $v \in G$ un vector de pagos $\phi(v)$, donde la coordenada $\phi_j(v)$ es el pago del jugador j.

3.1.1. El valor de Shapley

Utilizando una aproximación axiomática, Shapley (1953) definió el valor que es la solución más extendida en juegos cooperativos.

El método utilizado por Shapley consiste en proponer unas determinadas propiedades que debería verificar esa solución única y demostrar que estas mismas propiedades la caracterizan de forma unívoca, de manera que pueden ser tomadas como axiomas.

Definimos dos conceptos previos antes de analizar estas propiedades.

Diremos que $j \in M$ es un **jugador nulo** en el juego (M, v) si para cada $S \subset M \setminus \{j\}$, $v(S \cup \{j\}) = v(S)$. Un jugador nulo, j, no cambia el valor de ninguna coalición a la cual se une.

Diremos que $i, j \in M$ son **jugadores simétricos** en el juego (M, v) si para cada $S \subset M \setminus \{i, j\}$, $v(S \cup \{i\}) = v(S \cup \{j\})$.

Entonces, los axiomas que se exige que satisfaga esa solución única de un juego (M, v), que se designa por $\phi(v)$, son:

- **Eficiencia** si para cada $(M, v) \in G$, $\sum_{j=1}^{m} \phi_j(v) = v(M)$.
- **Jugador nulo** si para cada $(M, v) \in G$ y j jugador nulo, $\phi_j(v) = 0$.
- **Simetría** si para cada $(M, v) \in G$ y para cada par de jugadores simétricos $j, i \in M$, $\phi_i(v) = \phi_j(v)$.

- **Aditividad** si dados (M, v), $(M, w) \in G$,$\phi(v + w) = \phi(v) + \phi(w)$.

El significado de estas propiedades es claro; la propiedad de eficiencia, nos indica que la suma de las asignaciones que reciben todos los jugadores ha de coincidir con el valor de la coalición total; la propiedad de jugador nulo, establece que si un jugador no realiza aportación a ninguna coalición entonces no debe recibir asignación alguna; la propiedad de simetría, nos dice que si dos jugadores son intercambiables en el juego deben recibir el mismo pago; y la propiedad de aditividad, indica que el pago que reciben los jugadores en un juego es igual a la suma de los pagos que recibirían si el juego se dividiera en dos juegos.

Shapley probó que existe un único valor, denotado por Sh, en G, satisfaciendo las propiedades anteriores, y se puede expresar de la siguiente forma:

$$Sh_j(v) = \sum_{S \subset M \setminus \{j\}} \frac{s!(m-s-1)!}{m!}(v(S \cup \{j\}) - v(S)) \qquad \forall j \in M, \tag{3.1}$$

donde $s = |S|$, representa el número de jugadores participantes en la coalición S, y $m = |M|$, el número de participantes en el juego.

El valor de Shapley se puede interpretar como el valor esperado por un jugador obtenido como un promedio ponderado de sus contribuciones marginales a cada una de las coaliciones a las que podría incorporarse.

3.1.1.1 Ejemplo

Una empresa tecnológica utiliza un sistema que recomienda si la empresa debería contratar a un solicitante al puesto. Este sistema toma dos características binarias de entrada 'es joven' y 'tiene ambición' ($x = (x_1, x_2)$). El sistema devuelve un resultado de recomendación entre 0(no contratar) y 1(contratar). Definimos $f_{both}(x) ::= x_1 \wedge x_2$ (solo se contrata a jovenes con ambición). En la siguiente tabla se recoge la distribución de probabilidad de las dos características de entrada con su respectiva predicción.

x_1	x_2	$\Pr[X = x]$	$f_{both}(x)$
0	0	0.1	0.0
0	1	0.0	0.0
1	0	0.4	0.0
1	1	0.5	1.0

Consideremos la entrada (1,1), en la que se presenta un joven con ambición, para la que el modelo devuelve un resultado de 1. Queremos calcular lo que ha contribuido cada característica a esta predicción.

Esta situación puede modelarse mediante un juego cooperativo, con la siguiente función característica:

$$v(\emptyset) = 0; \quad v(\{1\}) = 0; \quad v(\{2\}) = 0; \quad v(\{1,2\}) = 1$$

donde $v(\{1\})$ significa que sólo la primera característica está presente ($x = (1, 0)$). El conjunto 2^M de coaliciones

$$2^M = \big\{\emptyset, \{1\}, \{2\}, \{1,2\}\big\}$$

Calculemos el pago esperado para cada jugador de acuerdo a la ecuación (5):

$$Sh_1(v) = \frac{1!}{2!}(v(\{1\}) - v(\emptyset)) + \frac{1!}{2!}(v(\{1,2\}) - v(\{2\})) = \frac{1}{2}$$

$$Sh_2(v) = \frac{1!}{2!}(v(\{2\}) - v(\emptyset)) + \frac{1!}{2!}(v(\{1,2\}) - v(\{1\})) = \frac{1}{2}$$

Por lo que el valor de Shapley sería: $Sh(v) = (\frac{1}{2}, \frac{1}{2})$. Como vemos, el valor de Shapley cumple la propiedad de eficiencia.

3.2. Juegos Explicativos

Para explicar la predicción de una instancia x con el valor de Shapley, es necesario formular un juego cooperativo en el que los jugadores se corresponden con las m características y una función característica v^x que se corresponda con el valor de la predicción. De forma que el juego cooperativo viene definido por el par (M, v^x).

La función característica v^x del juego explicativo debe estar definida para cada subconjunto $S \subseteq M$, tal que $v^x(S)$ represente la contribución de un registro $x_S \in S$. Esto nos permitirá calcular la contribución marginal de cada característica presente en S a la predicción y obtener su valor de Shapley.

Siguiendo la definición de las atribuciones aditivas de características y el axioma de eficiencia que cumplen los valores de Shapley, definimos $v^x(M) ::= f(x) - \phi_0$, esto es los pagos de la coalición total deben ser la diferencia entre la predicción del modelo a explicar y una predicción media de referencia.

Una vez definida la eficiencia del vector de pagos para los juegos explicativos, nos encontramos con el problema de obtener la función característica cuando algunas características no contribuyan, cuando están ausentes.

Para obtener los valores de Shapley se necesita evaluar todas las posibles coaliciones con y sin la j-ésima característica. ¿Pero cómo calcula un modelo de machine learning una predicción con un subconjunto de características? ¿Qué pasa con las que están ausentes?

Encontramos que los métodos existentes tratan este problema muestreando aleatoriamente características ausentes de acuerdo a una distribución particular y calculando el valor esperado para esa predicción. Las formulaciones del juego resultantes difieren sólo en las distribuciones usadas en cada método.

La notación que usaremos para formular el juego será la siguiente. D^{inp} es la distribución empírica de entrada, es decir, de los datos de entrenamiento. Denotaremos por x como una entrada de la que conocemos sus valores, y r para denotar otros valores de entrada que no conocemos. Usaremos $x_S = \{x_j, j \in S\}$ para representar el subconjunto de características de x presentes en S. Utilizaremos R para representar el conjunto de entradas aleatorias. Por último, introduciremos la entrada compuesta $z(x, r, S)$, que concuerda con los valores de x en S y con los valores de r para las características que no están en S.

$$z(x, r, S) = (z_1, z_2, ..., z_m), \quad \text{donde} \quad z_j = \begin{cases} x_j & j \in S \\ r_j & j \notin S \end{cases}$$

Explicaremos algunos de estos métodos a continuación.

3.2.1. IME

Este método fue el primero en hacer uso de los valores de Shapley en machine learning. Surge para tener en cuenta las interacciones entre las características.

En un modelo lineal en el que sabemos que las características no interactuan entre ellas, tiene sentido pensar que la contribución de una característica podría calcularse como la diferencia entre lo que contribuye una característica cuando su valor es x_j y lo que es esperado que contribuya.

$$\phi_j(x) = \beta_j x_j - \beta_j \mathbb{E}[x_j]$$

Sin embargo, en la práctica las características a menudo interactúan entre ellas, por lo que para detectar las interacciones, deberíamos observar el cambio de predicción para cada subconjunto de características y combinarlos para obtener la contribución de cada característica, obteniendo así el valor Shapley de cada característica.

Todas las posibles coaliciones (conjuntos) de características tienen que ser evaluadas con y sin la j-ésima característica para calcular el valor Shapley. Sin embargo, calcular la solución exacta, tiene una complejidad exponencial, lo que hace el método infactible para el uso práctico. El siguiente método, IME (Strumblej and Kononoenko 2010),Interactions-based Method for Explanation,, es una aproximación de los valores Shapley.

Definimos la predicción del modelo condicionada por un subconjunto de características $S \subseteq M$ del que se conocen sus valores por:

$$f_S(x) = \mathbb{E}[f|X_j = x_j, \forall j \in S]$$

Para un conjunto vacío la ecuación se reduce a $f_{\{\}}(x) = \mathbb{E}[f]$, lo que nos permite definir la contribución de los distintos subconjuntos de características:

$$v^x(S) = f_S(x) - f_{\{\}}(x)$$

Se define la diferencia de predicción de un subconjunto como el cambio esperado de predicción al observar esos valores para una instancia x.

Para definir las interacciones, los autores de IME, definen la contribución de un subconjunto de características como la suma de todas las interacciones I, de todos los subconjuntos W, del subconjunto de características conocidas S:

$$v^x(S) = \sum_{W \subseteq S} I_W(x)$$

lo que determina únicamente las interacciones:

$$I_S(x) = v^x(S) - \sum_{W \subset S} I_W(x).$$

Finalmente, cada interacción se divide entre las características participantes, lo que define la contribución:

$$\phi_j(x) = \sum_{W \subseteq S \setminus \{j\}} \frac{I_{W \cup \{j\}}(x)}{|W| + 1}$$

Esto lleva a la siguiente expresión (demostración en Strumbelj and Kononenko, 2010):

$$\phi_j(x) = Sh_j(v^x) = \sum_{S \subset M \setminus \{j\}} \frac{s!(m-s-1)!}{m!}(v^x(S \cup \{j\}) - v^x(S)) \qquad \forall j \in M, \tag{3.2}$$

que es equivalente al valor de Shapley, es decir, la contribución de las características es el valor de Shapley de esas características. El objetivo, es distribuir el valor de la coalición formada por todas las características de una manera justa, y el valor de Shapley es la única solución que satisface las propiedades de eficiencia, jugador nulo, simetría y aditividad.

Podemos escribir de una forma distinta pero equivalente la expresión de valor de Shapley:

$$Sh_j(v^x) = \frac{1}{m!} \sum_{\sigma \in \pi(M)} (v^x(Pre^j(\sigma) \cup \{x_j\}) - v^x(Pre^j(\sigma))) \quad j = 1, ..., m$$

donde $\pi(m)$ es el conjunto de las permutaciones del conjunto de características M y $Pre^j(\sigma)$ el conjunto de características predecesoras a la característica x_j en la permutación $\sigma \in \pi(m)$.

Para evitar el problema de la complejidad exponencial, Strumbelj y Kononenko(2010) desarrollaron un proceso de remuestreo eficiente para aproximar las contribuciones de las características perturbando las características del registro de entrada:

$$\hat{Sh}_j = \frac{1}{K}\sum_{k=1}^{K}(f(z_{[z_s=x_s, s\in Pre^j(\sigma)\cup\{x_j\}]}) - f(z_{[z_s=x_s, s\in Pre^j(\sigma)]})) \tag{3.3}$$

donde la notación $z_{[z_j=x_j, j\in S]}$ representa una instancia z, formada por una combinación aleatoria de características, con el valor de la característica j reemplazado por el valor de esa característica en la instancia x. El valor s va de 1 a m, comprobando si cada característica pertenece o no al subconjunto de predecesores en esa permutación de la característica j. El valor K representa el número de muestras que se crean con reemplazamiento para definir el valor de Shapley para cada característica.

Para aproximar el valor esperado condicionado por x_S, $\mathbb{E}[f(x)|x_S]$, se reemplazan valores de características aleatoriamente, provocando que el método asuma que las características son independientes entre ellas, rompiendo así, la correlación entre las características presentes y las ausentes.

Podemos formular el juego explicativo de la siguiente manera:

$$v^x_{unif}(S) = \mathbb{E}_{R\sim U}[f(z(x, R, S))] - \mathbb{E}_{R\sim U}[f(R)]$$

donde los valores de las características del subconjunto R se remuestrean acorde a una distribución uniforme ($R \sim U$) respecto al espacio de características, donde cualquier combinación de características tiene la misma probabilidad de ser escogida, ignorando de esta manera la distribución empírica de entrada.

La instancia z está formada, por los valores de las características de x en S, y por unos valores extraídos del espacio de características de forma aleatoria para el subconjunto de características R.

El algoritmo usado por este método para estimar el valor Shapley para cada característica sigue el proceso descrito en el algoritmo 2.

Algorithm 2 Aproximando la contribución de la j-ésima característica para el modelo f, de la instancia x, y número K de muestras

$\hat{Sh}_j \leftarrow 0$
for $k = 1, ..., K$ **do**
    `Se elige una permutación aleatoria de los valores de las características` $\sigma \in \pi(m)$.
    `Se escoge un registro aleatorio` z `perteneciente al espacio de características.`
    for $s = 1, ..., m$`:` **do**
        `Si` $x_S \in Pre^j(\sigma) \cup \{x_j\}$ `entonces:` $x'_s = x_s$`, en otro caso` $x'_s = z_s$
        `Si` $x_s \in Pre^j(\sigma)$ `entonces:` $x''_s = x_s$`, en otro caso` $x''_s = z_s$.
    end for
    $\hat{Sh}_j = \hat{Sh}_j + (f(x') - f(x''))$
end for
`Calculamos la media del valor Shapley para esa característica:` $\hat{Sh_j}(x) = \frac{\hat{Sh_j}(x)}{K}$

Vemos en el proceso que $f(x')$ y $f(x'')$ son dos predicciones del modelo de machine learning f, para dos observaciones, que solo se diferencian en el valor de la j-ésima característica. Estas observaciones se construyen tomando una instancia aleatoria z, y cambiando el valor de aquellas características que aparezcan antes de la j-ésima característica en la permutación σ (para x', la j-ésima característica también se cambia) por el valor de las características de la instancia que queremos explicar x.

El procedimiento se tiene que repetir para todas las características $j = 1, ..., m$, para poder obtener todos los valores Shapley.

3.2.1.1 Ejemplo

Volviendo al ejemplo descrito anterior en el apartado del valor de Shapley, queremos averiguar las contribuciones de las características para $x = (1, 1)$, de acuerdo a este método. En este caso, el objetivo del juego es dividir la diferencia entre la predicción para un registro en particular y el valor esperado. Calcularemos primero el valor esperado para nuestro modelo:

$$\mathbb{E}[f(Z_1, Z_2)] = \frac{1}{4}(0 + 0 + 0 + 1) = \frac{1}{4}$$

. La función característica viene descrita de la siguiente manera:

$$\begin{aligned} v^x(\emptyset) &= 0 \\ v^x(\{1\}) &= \mathbb{E}[f(Z_1, Z_2)|Z_1 = 1] - \mathbb{E}[f(Z_1, Z_2)] = \frac{1}{2} - \frac{1}{4} = \frac{1}{4} \\ v^x(\{2\}) &= \mathbb{E}[f(Z_1, Z_2)|Z_2 = 1] - \mathbb{E}[f(Z_1, Z_2)] = \frac{1}{2} - \frac{1}{4} = \frac{1}{4} \\ v^x(\{1, 2\}) &= \mathbb{E}[f(Z_1, Z_2)|Z_1 = 1, Z_2 = 1] - \mathbb{E}[f(Z_1, Z_2)] = 1 - \frac{1}{4} = \frac{3}{4} \end{aligned}$$

Calculamos el pago esperado para cada jugador de acuerdo a la ecuación (3.3):

$$\hat{Sh}_1 = \frac{1}{2}[v^x(Pre^1(\{1,2\}) \cup \{1\}) - v^x(Pre^1(\{1,2\})) + v^x(Pre^1(\{2,1\}) \cup \{1\}) - v^x(Pre^1(\{2,1\}))] =$$
$$\frac{1}{2}[v^x(\emptyset \cup \{1\}) - v^x(\emptyset) + v^x(\{2\} \cup \{1\}) - v^x(\{2\})] = \frac{1}{2}[v^x(\{1\}) - v^x(\emptyset) + v^x(\{1,2\}) - v^x(\{2\})] =$$
$$\frac{1}{2}[0,25 - 0 + 0,75 - 0,25] = 0,375$$

$$\hat{Sh}_2 = \frac{1}{2}[v^x(Pre^2(\{1,2\}) \cup \{2\}) - v^x(Pre^2(\{1,2\})) + v^x(Pre^2(\{2,1\}) \cup \{2\}) - v^x(Pre^2(\{2,1\}))] =$$
$$\frac{1}{2}[v^x(\{1\} \cup \{2\}) - v^x(\{1\}) + v^x(\emptyset \cup \{2\}) - v^x(\emptyset)] = \frac{1}{2}[v^x(\{1,2\}) - v^x(\{1\}) + v^x(\{2\}) - v^x(\emptyset)] =$$
$$\frac{1}{2}[0,75 - 0,25 + 0,25 - 0] = 0,375$$

Por lo que los valores de Shapley calculados mediante este método son $\hat{Sh} = (0,375, 0,375)$. Vemos que estos valores cumplen la propiedad de la eficiencia pues $f(x) = \phi_0 + \sum_{j=1}^{M} \hat{Sh}_j = 0,25 + 0,375 + 0,375 = 1$.

3.2.1.2 Interpretación de una predicción individual

El valor $\hat{Sh}_j$ para una característica j indica lo que contribuye esa característica en una predicción individual en comparación a la predicción media de los datos de entrenamiento.

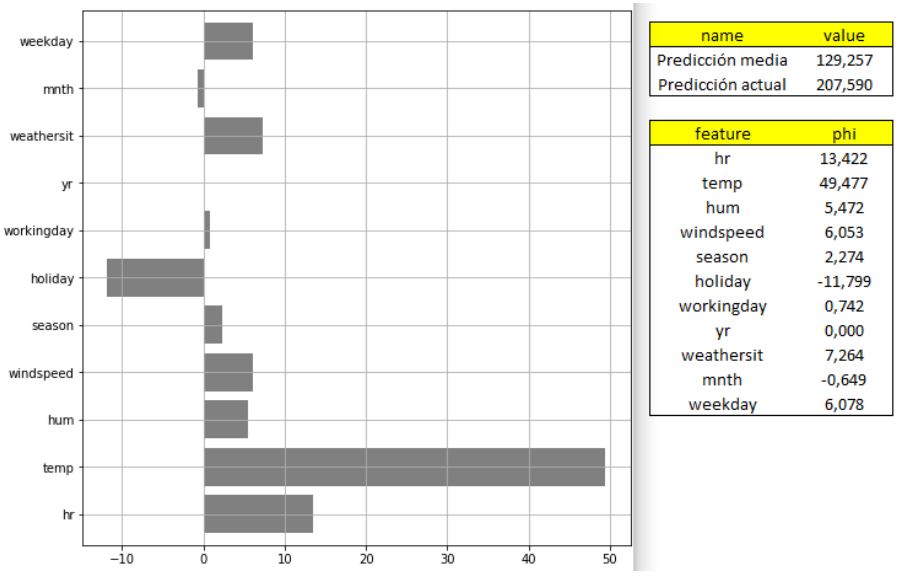
Podemos expresarlo como un método aditivo de atribuciones de características de la siguiente forma:

$$\hat{f}(x^i) = \phi_0 + \sum_{j=1}^{M} \hat{Sh}_j^i$$

donde $\hat{f}(x^i)$ es la predicción del modelo original para la instancia x^i, ϕ_0, la predicción media del modelo f para los datos de entrenamiento, y $\hat{Sh}_j^i$, la contribución de la característica j a la predicción.

3.2.1.2.1 Ejemplo

En la figura 3.1, vemos un ejemplo donde hemos aplicado el algoritmo para aproximar los valores de Shapley para explicar la predicción dada por el modelo de Random Forest para la instancia 11728 descrita en la figura 2.2 .



name	value
Predicción media	129,257
Predicción actual	207,590

feature	phi
hr	13,422
temp	49,477
hum	5,472
windspeed	6,053
season	2,274
holiday	-11,799
workingday	0,742
yr	0,000
weathersit	7,264
mnth	-0,649
weekday	6,078

Figura 3.1: IME: Aproximación de los valores de Shapley para la instancia 11728

La suma de las contribuciones, ϕ_j^i, más la predicción media (average prediction), ϕ_0, es igual a la predicción actual, $\hat{f}(x^i)$:

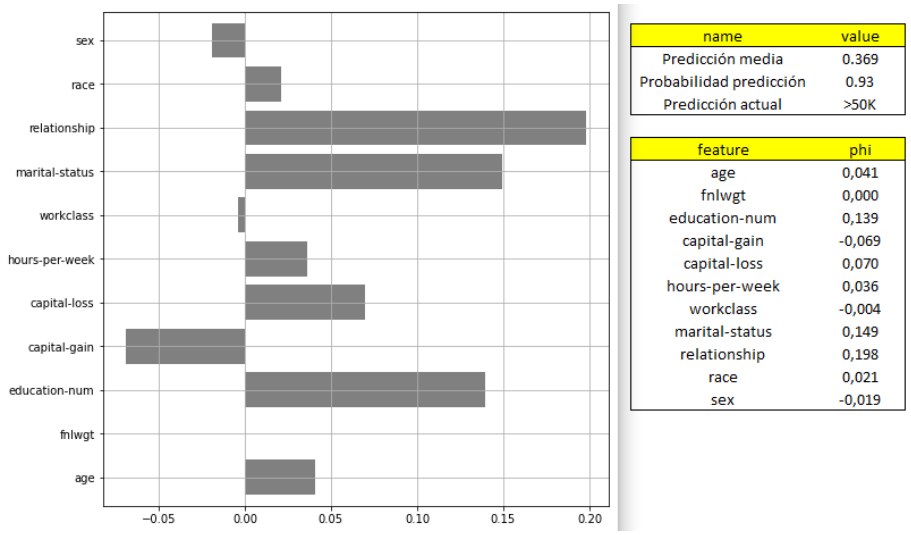
$$\hat{f}(x^i) = \phi_0 + \sum_{j=1}^{m} \phi_j^i$$

3.2.1.2.2 Ejemplo

Para el ejemplo de clasificación vemos los resultados obtenidos en la figura 3.2 para la predicción de la instancia a explicar descrita anteriormente en la figura 2.8.

La suma de las contribuciones, ϕ_j^i, más la predicción media (average prediction), ϕ_0, es igual a la predicción actual, $\hat{f}(x^i)$:

$$\hat{f}(x^i) = \phi_0 + \sum_{j=1}^{m} \phi_j^i$$



name	value
Predicción media	0.369
Probabilidad predicción	0.93
Predicción actual	>50K

feature	phi
age	0,041
fnlwgt	0,000
education-num	0,139
capital-gain	-0,069
capital-loss	0,070
hours-per-week	0,036
workclass	-0,004
marital-status	0,149
relationship	0,198
race	0,021
sex	-0,019

Figura 3.2: IME: Aproximación de los valores de Shapley para la instancia 9188

3.2.1.3 Ventajas y desventajas

Ventajas:

- La diferencia entre la predicción y la predicción media está distribuida de manera justa entre los valores de las características para esa instancia.
- Los valores Shapley dan explicaciones contrastables. Es decir, podemos comparar una predicción con la de otro registro o con un suconjunto de ellos.
- Es un método de explicación con una teoría sólida. Los axiomas cumplidos por la solución del valor Shapley le dan una fundación razonable a la explicación.

Desventajas:

- No hay un número apropiado definido de iteraciones K. Uno bajo disminuye el tiempo y aumenta la varianza del valor de Shapley.
- No es un buen método si estás buscando explicaciones con pocas características. Las explicaciones de los valores Shapley se crean usando todas las características.
- Necesitas acceso a los datos si quieres calcular los valores Shapley para una nueva instancia.
- Al romper la correlación entre las características, provoca que instancias poco probables puedan tener mucho peso en el modelo.

3.2.2. SHAP

SHAP (SHapley Additive exPlanations) es un método basado en los valores Shapley de la teoría de juegos para explicar predicciones individuales. En esta sección, nos encontraremos con dos métodos distintos; KernelSHAP, un método basado en un kernel e inspirado en LIME, utilizado para estimar los valores de Shapley; y TreeSHAP, un método de estimación para los modelos basados en árboles. Además, SHAP viene con varios métodos de interpretación global del modelo basados también en los valores de Shapley.

Los valores de Shapley son la única solución que satisface los axiomas de eficiencia, jugador nulo simetría y aditividad. Además, en SHAP, destacan la presencia de una única solución para los métodos adtivos con tres propiedades deseables. Mientras que estas propiedades no son ajenas a IME, eran desconocidas para los métodos aditivos anteriores.

- *Precisión local*

$$f(x) = g(x') = \phi_0 + \sum_{j=1}^{M} \phi_j x'_j$$

  El modelo explicativo $g(x')$ se iguala a la predicción original $f(x)$, donde $x' \in \{0,1\}^M$, indica la presencia o no de una característica. Cuando todas las características están presentes, se corresponde a la propiedad de eficiencia de los valores de Shapley.

- *Valores faltantes*

$$x'_j = 0 \implies \phi_j = 0$$

  Con esta propiedad, nos aseguramos que una característica que no esté presente reciba una atribución nula.

- *Consistencia*

  Definimos una función $h(x') = x$ donde $h : \{0,1\}^M \to \mathbb{R}^M$. Si $x'_j = 1$, significa que la característica está presente en la coalición y la función h devuelve el valor de esa característica que queremos explicar. Cuando el valor $x'_j = 0$ indica que la característica j no está presente.

  Sea $f_x(x') = f(h(x'))$ y $x'_{\setminus j}$ un registro indicando $x'_j = 0$. Para dos modelos cualesquiera f y f' que satisfacen:

$$f'_x(x') - f'_x(x'_{\setminus j}) \geq f_x(x') - f_x(x'_{\setminus j})$$

  para todas las entradas $x' \in \{0,1\}^M$, entonces:

$$\phi_j(f', x) \geq \phi_j(f, x)$$

3.2.2.1. KernelSHAP

El algoritmo de KernelSHAP (Lundberg and Lee 2017), nos provee de explicaciones interpretables de modelos de caja negra. Para lidiar con el problema de la complejidad exponencial del calculo de los valores de Shapley, este método aproxima los valores de Shapley mediante una regresión lineal ponderada.

Se describen los valores SHAP, como una medida unficada de importancia de las características. Estos valores atribuyen a cada característica el cambio en la predicción esperada del modelo cuando se condiciona por esa característica. Explican como se alcanza desde un valor base $E[f(z)]$, valor esperado si no se conociera ninguna característica, a la predicción del modelo $f(x)$.

Se define la predicción de un registro como $f_x(z') = f(h(z')) = \mathbb{E}[f(z)|z_S]$, siendo S el conjunto de índices distintos de cero en z'. La función h es la descrita anteriormente en las propiedades, que devuelve el valor de la característica cuando está presenta en la coalición, y en caso de no estar presente la característica en la coalición, devuelve el valor de esa característica para otra instancia, elegida aleatoriamente dentro de los datos de entrenamiento.

Al igual que IME, nos encontramos con el objetivo de repartir la diferencia de predicción esperada cuando se conoce un subconjunto de características de una manera justa.

La diferencia entre los métodos, es la distinta forma en la que se comportan cuando se encuentran con las características ausentes, es decir, con las características no presentes en S.

Este método, conecta los valores de Shapley con la regresión lineal. Busca un núcleo ponderado, de forma que obtenga los valores de Shapley.

Para ello, parte de LIME, método explicado en el capítulo anterior que aproxima localmente el modelo f mediante un modelo lineal. Ya que LIME es un método de atribuciones aditivas de características y los valores de Shapley son la única solución posible a un método aditivo que satisfaga las propiedades descritas al principio de SHAP, se busca la manera de que los valores de Shapley sean la solución a la ecuación (2.4).

Dependiendo de las elecciones de la función de perdida $\mathcal{L}$, el núcleo ponderado π_x, y el término de regularización Ω, podemos obtener los valores de Shapley. En LIME, la elección de estos parámetros se realiza de forma heurística.

En KernelSHAP, las formas específicas de $\mathcal{L}$, π_x y Ω que hacen que la solución para la ecuación (4) cumpla las propiedades de los valores SHAP son:

$$\Omega(g) = 0$$

$$\pi_x(z') = \frac{M-1}{\binom{M}{|z'|}|z'|(M-|z'|)}$$

$$\mathcal{L}(f, g, \pi_x) = \sum_{z' \in Z} [f(h(z')) - g(z')]^2 \pi_x(z')$$

donde $|z'|$ es el número de valores distintos de 0 en z'. Z representa los datos de entrenamiento y g es nuestro modelo explicativo que se entrena mediante la optimización de la ecuación de perdida $\mathcal{L}$. El modelo g se podrá expresar mediante un método aditivo:

$$g(z') = \phi_0 + \sum_{j=1}^{M} \phi_j z'_j$$

donde los coeficientes estimados del modelo, ϕ_j, son los valores de Shapley. El término ϕ_0 será la predicción media del modelo f para el conjunto de datos de entrenamiento.

Podemos formular el juego explicativo de la siguiente manera:

$$v^x_{inp} = \mathbb{E}_{R \sim D^{inp}}[f(z(x, R, S))] - \mathbb{E}_{R \sim D^{inp}}[f(R)]$$

pues el subconjunto de características desconocidas R se remuestrean de acuerdo a la distribución empírica ($R \sim D^{inp}$).

De esta forma, la instancia z está formada, por los valores de las características de x en S, y por los valores de una instancia aleatoria z perteneciente a los datos de entrenamiento.

El algoritmo usado por este método para calcular los valores de SHAP para una instancia x sigue el siguiente proceso:

- Datos iniciales: Número de muestras K, ejemplo de interés x, matriz de datos X y modelo de machine learning f.
- Se muestrean coaliciones $z'_{(k)} \in \{0,1\}^M, k \in \{1, ..., K\}$(1=característica presente en la coalición, 0 = característica ausente).
- Se calcula la predicción para $z'_{(k)}$ transformando previamente el vector $z'_{(k)}$ mediante la función h tal que la predicción final es: $f(h(z'_{(k)}))$.
- Se calcula el peso para cada $z'_{(k)}$ con el núcleo de SHAP, π_x.
- Se ajusta el modelo lineal ponderado y se obtienen los valores de Shapley $\phi_{(k)}$, que son los coeficientes de la regresión lineal.

3.2.2.1.1 Ejemplo

Volvemos a retomar el ejemplo descrito en los valores de Shapley para calcular las contribuciones de las características para la entrada $x = (1, 1)$. Al igual que IME este método define la función característica del juego como el cambio esperado al observar unos valores. Calculamos primero el valor esperado para nuestro modelo de acuerdo a la distribución de probabilidad de entrada:

$$\mathbb{E}[f(Z_1, Z_2)] = 0,1 \times 0 + 0,4 \times 0 + 0,5 \times 1 = 0,5$$

Las distintas coaliciones que podemos obtener son $z'_{(0)} = (0, 0); z'_{(1)} = (1, 0); z'_{(2)} = (0, 1); z'_{(3)} = (1, 1)$. En kernelSHAP se define la predicción de un registro como el valor esperado condicionado por el subconjunto S. Omitimos el calculo de $z'_{(0)}$ y $z'_{(3)}$ pues el peso de estas coaliciones será infinito. Calculamos la predicción para las otras dos coaliciones:

$$\mathbb{E}[f(Z_1, Z_2)|Z_1 = 1] = \frac{0,4}{0,9} \times 0 + \frac{0,5}{0,9} \times 1 = 0,55$$

$$\mathbb{E}[f(Z_1, Z_2)|Z_2 = 1] = \frac{0}{0,5} \times 0 + \frac{0,5}{0,5} \times 1 = 1$$

y calculamos los pesos para estas coaliciones que será el mismo al tener la misma cardinalidad z'.

$$\pi(z'_{(1)}) = \pi(z'_{(2)}) = \frac{2-1}{\binom{2}{1} \times 1 \times (2-1)} = \frac{1}{2}$$

El siguiente paso es ajustar el modelo lineal ponderando minimizando la función de pérdida.

$$\mathcal{L}(f, g, \pi) = \Big[\mathbb{E}[f(Z_1, Z_2)|Z_1 = 1] - (\phi_0 + \phi_1)\Big]^2 \frac{1}{2} + \Big[\mathbb{E}[f(Z_1, Z_2)|Z_2 = 1] - (\phi_0 + \phi_2)\Big]^2 \frac{1}{2}$$

donde $\phi_0 = \mathbb{E}[f(Z_1, Z_2)]$ y $f(x) = \sum \phi_j$ por lo que $\phi_2 = f(x) - \phi_0 - \phi_1$. Obteniendo así:

$$\mathcal{L}(f, g, \pi) = \Big[\mathbb{E}[f(Z_1, Z_2)|Z_1 = 1] - \phi_0 - \phi_1\Big]^2 \frac{1}{2} + \Big[\mathbb{E}[f(Z_1, Z_2)|Z_2 = 1] - f(x) + \phi_1\Big]^2 \frac{1}{2}$$

$$\mathcal{L}(f, g, \pi) = \Big[0,55 - 0,5 - \phi_1\Big]^2 \frac{1}{2} + \Big[1 - 1 + \phi_1\Big]^2 \frac{1}{2}$$

Derivando e igualando a cero obtenemos el punto crítico $\phi_1 = 0,016$ que es el mínimo de esa función. Entonces, $\phi_2 = 0,484$. Vemos que los valores obtenidos, son los valores de Shapley, que cumplen también la propiedad de eficiencia.

3.2.2.1.2 Interpretación de una predicción individual

Mediante este método la interpretación de una predicción individual puede expresarse de manera sencilla como un método aditivo pues los valores SHAP explican como se alcanza la predicción desde el valor base $\mathbb{E}[f(x)]$, que es la predicción media para los datos de entrenamiento:

$$\hat{f}(x^i) = \phi_0 + \sum_{j=1}^{M} \phi_j^i = \mathbb{E}[f(x)] + \sum_{j=1}^{M} \phi_j^i$$

donde $\hat{f}(x^i)$ es la predicción del modelo y los valores $\phi_j^i = shap_j^i$, los valores obtenidos al minimizar la función $\mathcal{L}$.

Al igual que LIME, podemos obtener explicaciones dispersas, es decir, con pocas características, que son las más influyentes en la predicción del modelo. Sin embargo, presentaremos los resultados con todas las características para comparar con el resto de modelos.

El paquete SHAP, ofrece una visualización propia de los resultados en la que las contribuciones se ven como fuerzas que van desplazando el valor base inicial hasta el resultado predicho.

3.2.2.1.3 Ejemplo

Para el problema de regresión, obtenemos los resultados de la figura 3.3, como ejemplo de la visualización propia del paquete SHAP, y los resultados de la figura 3.4 en la que observamos los valores para todas las características, para la instancia 11728 descrita en la figura 2.2.

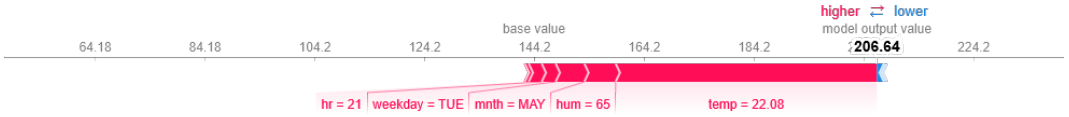


Figura 3.3: KernelSHAP: Valores SHAP para la instancia 11728, visualización propia de SHAP

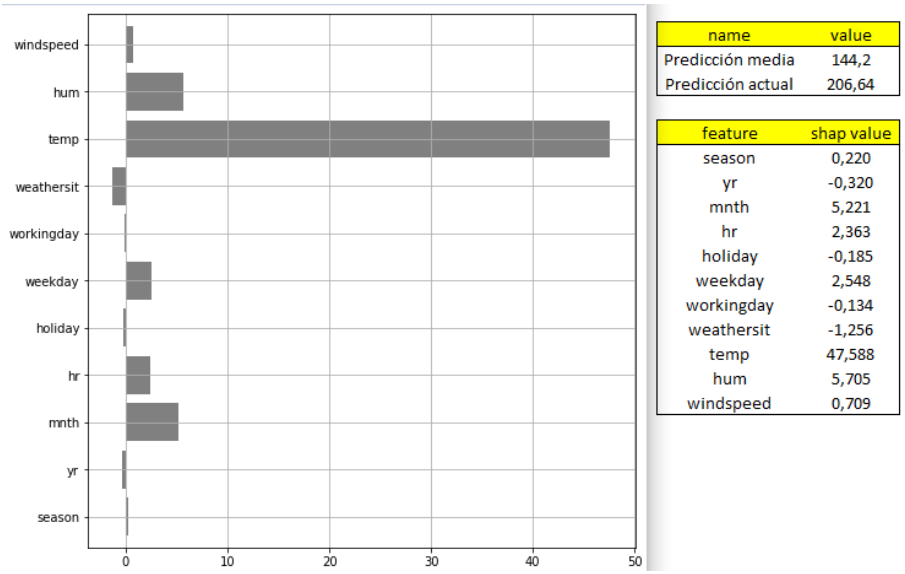


Figura 3.4: KernelSHAP: Valores SHAP para la instancia 11728

La suma de los valores shap, $shap_j^i$, obtenidos en la figura 3.4, más la predicción media, ϕ_0, es igual a la predicción del modelo de random forest $\hat{f}(x^i)$:

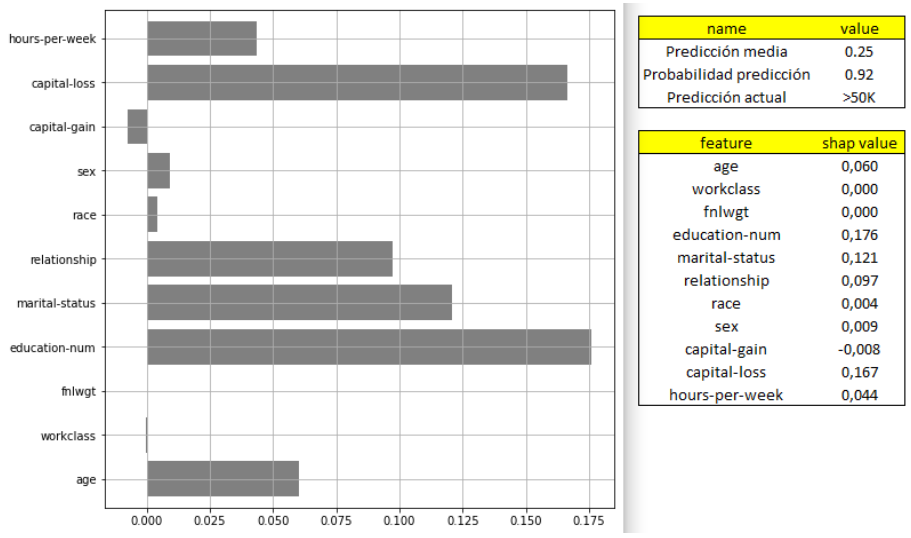
$$\hat{f}(x^i) = \phi_0 + \sum_{j=1}^{m} shap_j^i$$

3.2.2.1.4 Ejemplo

Para el problema de clasificación, queremos explicar la predicción para la instancia 9118, descrita en la figura 2.8, mediante KernelSHAP, obteniendo los resultados que vemos en la figura 3.5.

La suma de los valores shap, $shap_j^i$, obtenidos por este método, en la figura 3.5, más la predicción media, ϕ_0, es igual a la predicción del modelo de random forest $\hat{f}(x^i)$:

$$\hat{f}(x^i) = \phi_0 + \sum_{j=1}^{m} shap_j^i$$



name	value
Predicción media	0.25
Probabilidad predicción	0.92
Predicción actual	>50K

feature	shap value
age	0,060
workclass	0,000
fnlwgt	0,000
education-num	0,176
marital-status	0,121
relationship	0,097
race	0,004
sex	0,009
capital-gain	-0,008
capital-loss	0,167
hours-per-week	0,044

Figura 3.5: KernelSHAP: Valores SHAP para la instancia 9118

3.2.2.1.5 Ventajas y desventajas

Ventajas:

- Al igual que IME, ya que SHAP calcula los valores de Shapley, es un método de explicación con un fundamento teórico. La predicción se distribuye de manera justa entre las características.
- SHAP conecta LIME con los valores de Shapley. Unifica el campo de machine learning interpretable.
- Las explicaciones son contrastables.

Desventajas:

- KernelSHAP ignora las dependencias entre las características. Reemplazar los valores de las características con valores de otras instancias aleatorias, cuando las características no son independientes, hace que el modelo ponga mucho peso a instancias poco probables.
- KernelSHAP es lento cuando quieres calcular los valores de Shapley para muchas instancias.
- Al igual que IME se necesita acceso a los datos de entrenamiento para poder reemplzar las características ausentes de la instancia de interés.

3.2.2.1.6 Software

Los autores implementaron el paquete SHAP en Python, se ha utilizado la función KernelExplainer para la obtención de los ejemplos de este métdodo.

3.2.2.2. TreeSHAP

TreeSHAP (Lundberg, Erion and Lee 2017), es una variante de SHAP para modelos de machine learning basados en árboles, cómo los árboles de decisión, random forests o gradient boosted trees. Este método fue introducido como un modelo específico como alternativa más

rápida a KernelSHAP.

Este algoritmo se propuso para calcular los valores SHAP, pero en tiempo polinomial en vez de exponencial. Especificamente, reduce el tiempo computacional de $O(TL2^M)$ a $O(TLD^2)$, donde T es el número de árboles, L es el número máximo de nodos hoja en cualquier árbol, y D es la profundidad máxima de cualquier árbol.

Se define la predicción del modelo condicionada por un subconjunto de características $S \subseteq M$ del que se conocen sus valores por $f_S(x) = \mathbb{E}[f(x)|x_S]$. Para un modelo basado en árboles esta predicción se puede calcular de forma recursiva.

El algoritmo calcula la predicción para los nodos que son alcanzables por el subconjunto de características S. Este algoritmo calcula el valor de Shapley exacto, pues calcula la predicción para cada subconjunto.

Expliquemos esto mediante un ejemplo sencillo. Si conocieramos todas las características, es decir, si S fuera el conjunto de todas las características, la predicción del nodo hoja para la instancia x es la predicción esperada. Si no conocieramos ninguna característica, $S = \emptyset$, la predicción de x puede terminar en cualquiera de los nodos hoja, y por lo tanto la predicción esperada será la media ponderada de las predicciones de todos los nodos hoja.

Imaginemos que conocemos ahora solo un conjunto de características $S \subset M$, tomamos como ejemplo el árbol de decisión descrito en la figura 2.6. Recordemos que la predicción para nuestra instancia descrita en la figura 2.8 caía en el segundo nodo hoja empezando por la izquierda. Imaginemos ahora, que conocemos todas las características a excepción de la característica 'education-num', entonces, el algoritmo tomaría los dos primeros nodos (son los que divide esa característica y están en el camino de la predicción) y calcularía una media ponderada de ellos. Si por ejemplo, la característica faltante sería 'Age', no habría ningun cambio en la predicción esperada porque esa característica no está en el camino tomado por la instancia.

Si la característica faltante se encuentra en el nodo raíz del árbol, se crearán dos subárboles y la instancia seguirá el camino que le corresponda en los dos árboles, y la predicción esperada se calculará como una media ponderada (por el número de registros en el nodo) de todos los nodos hoja posibles.

Una vez calculadas las predicciones esperadas para todos los subconjuntos en un árbol de decisión, se repite el proceso para todos los árboles que forman el modelo, y por la propiedad de aditividad del valor de Shapley, podemos calcular la media de los valores Shapley de todos los árboles.

El problema es que se tienen que calcular los valores esperados para todos los subconjuntos posibles de características en todos los árboles. Los autores de TreeSHAP propusieron un algoritmo, donde se calculan todos los posibles subconjuntos de un árbol a la vez.

Por lo que podemos formular el juego de la siguiente manera:

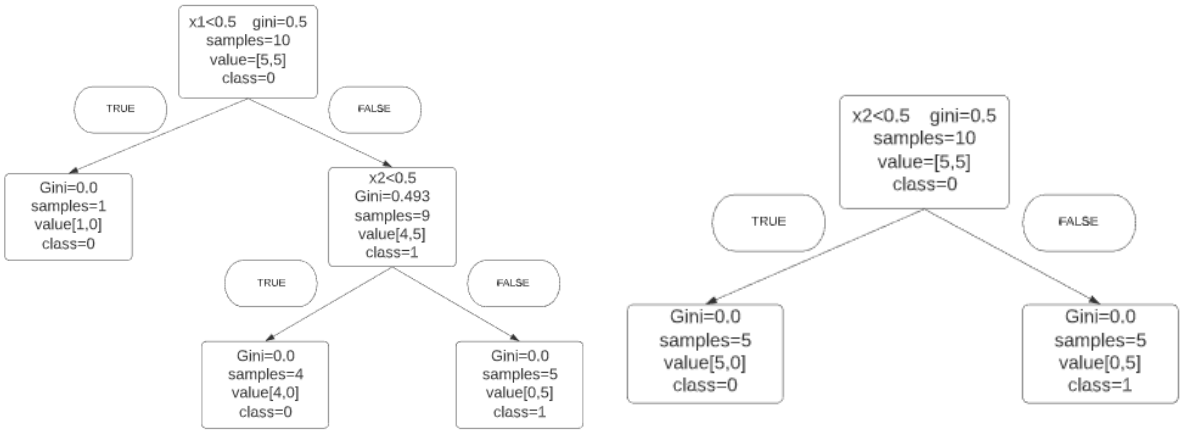
$$v^x_{inp} = \mathbb{E}_{R\sim D^{inp}}[f(z(x, R, S))|R_S = x_S] - \mathbb{E}_{R\sim D^{inp}}[f(R)]$$

ya que el algoritmo no tiene en cuenta las características desconocidas, y calcula la predicción sólo teniendo en cuenta las características presentes en S aprovechándose de la estructura de árbol de sus modelos.

3.2.2.2.1 Ejemplo

Retomamos de nuevo el ejemplo del apartado de Shapley para calcular las contribuciones de las características para la entrada $x = (1, 1)$. Primero calcularemos los dos árboles de decisión distintos que se pueden formar con dos características. Vemos estos árboles en la figura 3.6.

Calculemos las predicciones para todos los subconjuntos de acuerdo a este método para los



(a) Modelo A (b) Modelo B

Figura 3.6: Árboles de decisión

dos modelos:

$$f^A_{\{\}} = 0{,}5; \qquad f^A_{\{x_1\}} = \frac{5}{9} = 0{,}55; \qquad f^A_{\{x_2\}} = \frac{5}{6} = 0{,}83; \qquad f^A_{\{x_1,x_2\}} = 1$$

$$f^B_{\{\}} = 0{,}5; \qquad f^B_{\{x_1\}} = 0{,}0; \qquad f^B_{\{x_2\}} = 1; \qquad f^B_{\{x_1,x_2\}} = 1$$

Calculamos los valores SHAP para las dos características conociendo el valor de cada subconjunto:

$$\phi^A_1 = \frac{0!1!}{2!}[f^A_{\{x_1\}} - f^A_{\{\}}] + \frac{1!0!}{2!}[f^A_{\{x_1,x_2\}} - f^A_{\{x_2\}}] = \frac{1}{2}[0{,}05] + \frac{1}{2}[1 - 0{,}83] = 0{,}11$$

$$\phi^A_2 = \frac{0!1!}{2!}[f^A_{\{x_2\}} - f^A_{\{\}}] + \frac{1!0!}{2!}[f^A_{\{x_1,x_2\}} - f^A_{\{x_1\}}] = \frac{1}{2}[0{,}83 - 0{,}55] + \frac{1}{2}[1 - 0{,}5] = 0{,}39$$

$$\phi^B_1 = \frac{0!1!}{2!}[f^B_{\{x_1\}} - f^B_{\{\}}] + \frac{1!0!}{2!}[f^B_{\{x_1,x_2\}} - f^B_{\{x_2\}}] = \frac{1}{2}[0] + \frac{1}{2}[0] = 0{,}0$$

$$\phi^B_2 = \frac{0!1!}{2!}[f^B_{\{x_2\}} - f^B_{\{\}}] + \frac{1!0!}{2!}[f^B_{\{x_1,x_2\}} - f^B_{\{x_1\}}] = \frac{1}{2}[1 - 0{,}5] + \frac{1}{2}[1 - 0{,}5] = 0{,}5$$

Vemos que los valores obtenidos por cada modelo son sus respectivos valores de Shapley. Gracias a la aditividad de los valores de Shapley, podemos calcular la media de los valores Shapley para estos árboles obteniendo los valores finales $\phi_1 = 0{,}055$ y $\phi_2 = 0{,}445$, que cumplen también con la propiedad de eficiencia.

3.2.2.2.2 Interpretación de una predicción individual

Este método se interpreta de forma similar al método de KernelSHAP, ya que los valores obtenidos $shap^i_j$, explican como se alcanza la predicción de una instancia, $\hat{f}(x^i)$, desde el valor de predicción media, $\mathbb{E}[f(x)]$, que sería la predicción para un subconjunto de características vacío:

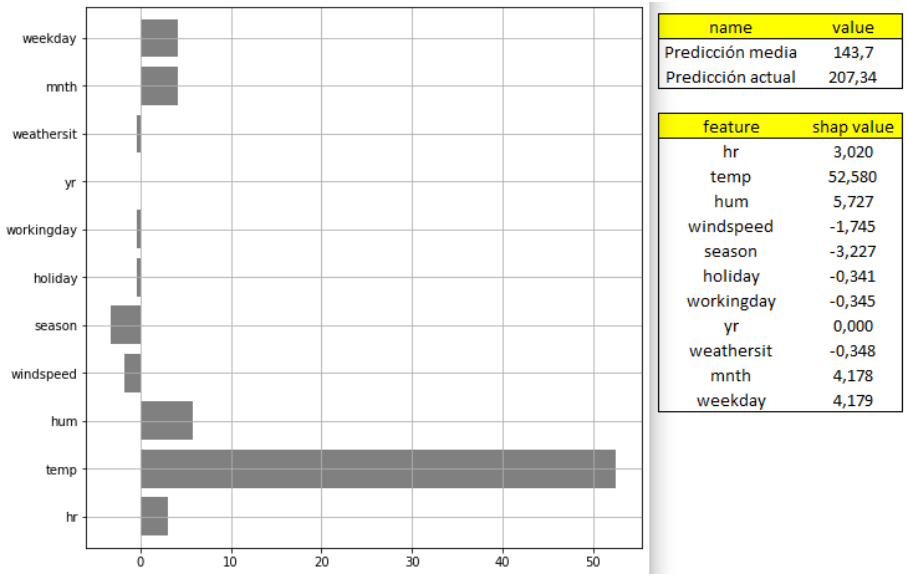
$$\hat{f}(x^i) = \phi_0 + \sum_{j=1}^{M} \phi^i_j = \mathbb{E}[f(x)] + \sum_{j=1}^{M} \phi^i_j$$

donde los valores $\phi^i_j = shap^i_j$, son las contribuciones de las características.

TreeSHAP también incluye una visualización como la de KernelSHAP, en la que las características más influyentes se ven cómo fuerzas que desplazan el valor base hasta la predicción final.

3.2.2.2.3 Ejemplo

Para el problema de regresión, obtenemos los resultados de la figura 3.7, en la que observamos los valores *shap* para todas las características, para la instancia 11728 descrita en la figura 2.2, y la predicción media del conjunto de datos.



name	value
Predicción media	143,7
Predicción actual	207,34

feature	shap value
hr	3,020
temp	52,580
hum	5,727
windspeed	-1,745
season	-3,227
holiday	-0,341
workingday	-0,345
yr	0,000
weathersit	-0,348
mnth	4,178
weekday	4,179

Figura 3.7: TreeSHAP: Valores SHAP para la instancia 11728

La suma de los valores shap, $shap_j^i$, obtenidos por este método, que vamos en la figura 3.7, y de la predicción media, ϕ_0, es igual a la predicción del modelo de random forest $\hat{f}(x^i)$:

$$\hat{f}(x^i) = \phi_0 + \sum_{j=1}^{m} shap_j^i$$

3.2.2.2.4 Ejemplo

Para el problema de clasificación, queremos explicar la predicción para la instancia 9118, descrita en la figura 2.8. Mediante KernelSHAP, obtenemos los resultados que vemos en las figuras 3.8, como ejemplo de visualización propia de SHAP, y 3.9.

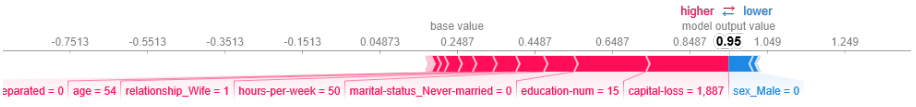
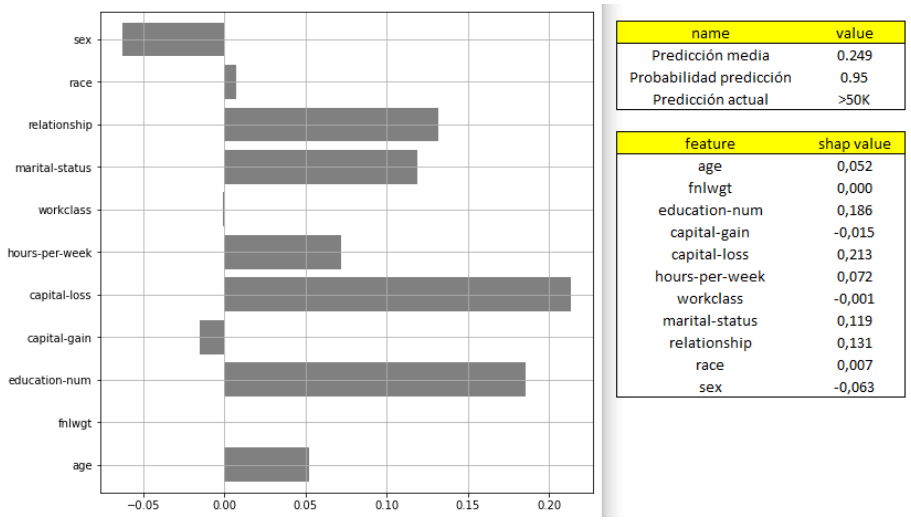


Figura 3.8: TreeSHAP: Valores SHAP para la instancia 9118, visualización propia de SHAP

La suma de los valores shap, $shap_j^i$, obtenidos en la figura 3.9, más la predicción media, ϕ_0, es igual a la predicción del modelo de random forest $\hat{f}(x^i)$:

$$\hat{f}(x^i) = \phi_0 + \sum_{j=1}^{m} shap_j^i$$



name	value
Predicción media	0.249
Probabilidad predicción	0.95
Predicción actual	>50K

feature	shap value
age	0,052
fnlwgt	0,000
education-num	0,186
capital-gain	-0,015
capital-loss	0,213
hours-per-week	0,072
workclass	-0,001
marital-status	0,119
relationship	0,131
race	0,007
sex	-0,063

Figura 3.9: TreeSHAP: Valores SHAP para la instancia 9118

3.2.2.2.5 Ventajas y desventajas

Ventajas:

- Al igual que el resto de métodos basados en el valor de Shapley tienen una fundación teórica.
- Es un método más rápido que KernelSHAP
- No rompe la correlación entre las características ausentes y las presentes.
- No necesita acceso a los datos de entrenamiento para calcular nuevas predicciones.

Desventajas:

- TreeSHAP añade un nuevo problema, pudiendo asignar valores distintos de cero a características que no tienen influencia en el modelo, debido a la correlación entre características.
- No es válido para todos los modelos.

3.2.2.2.6 Software

Para la realización de los ejemplos se ha utilizado la función TreeExplainer dentro del paquete de SHAP.

Capítulo 4

Conclusiones

El auge de las técnicas de machine learning en la vida cotidiana ha llevado a la creación de modelos con una gran capacidad predictiva. Sin embargo, estos modelos, conocidos como modelos de caja negra, suelen fallar a la hora de explicar sus propias predicciones individuales. El uso de los modelos de machine learning en la vida real, ha traído con él la necesidad de entender las predicciones de estos modelos. Es por eso, que en los últimos años se han desarrollado nuevas técnicas, cuyo objetivo es conseguir explicar las predicciones de los modelos de caja negra. En este trabajo, hemos enfocado el problema de interpretabilidad como métodos de atribuciones aditivas de las características.

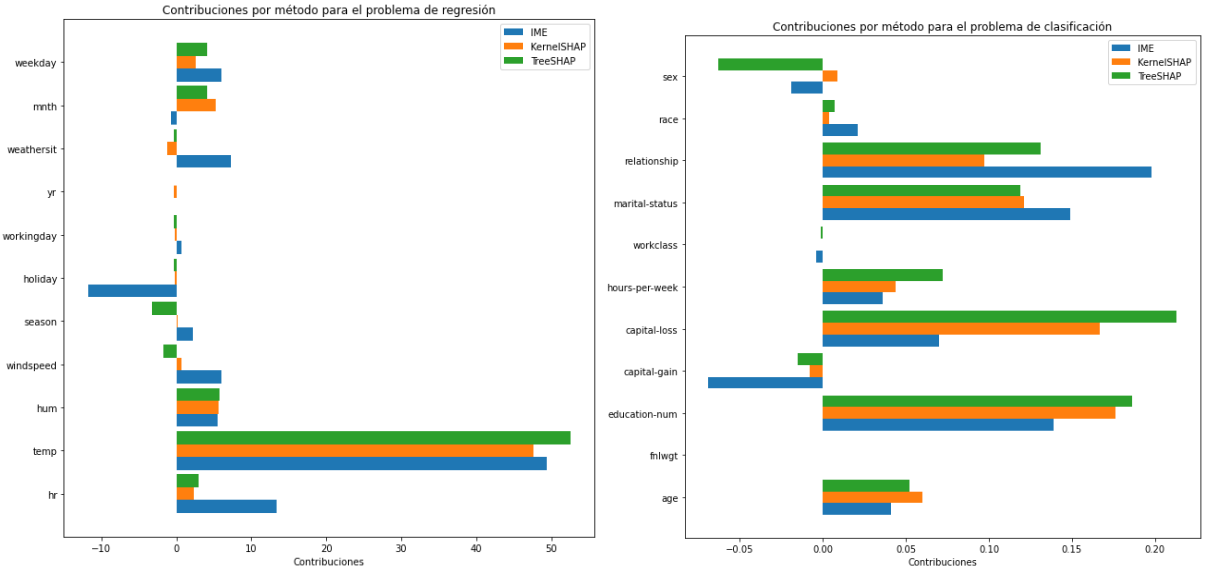
Por un lado, hemos estudiado modelos intrinsicamente interpretables (Regresión lineal y Árboles de decisión). Hemos visto que estos algoritmos no tienen gran capacidad predictiva pero destacan por su fácil interpretación. Hemos visto como interpretar las predicciones de estos modelos, y cómo los métodos post-hoc estudiados se apoyan en estos modelos para conseguir interpretabilidad.

Dentro de los métodos post-hoc, hemos visto 3 modelos agnósticos (LIME, IME y KernelSHAP) y un modelo específico (TreeSHAP). Estos métodos han sido utilizados para explicar las predicciones de un modelo de caja negra, concretamente, del Random Forest, algoritmo con gran capacidad predictiva. Dentro de estos métodos, LIME es el único que no está basado en el valor de Shapley. Sin embargo, hemos visto como KernelSHAP se basa en LIME para la obtención de los valores de Shapley. Vemos ahora una tabla resumen con una comparativa de estos métodos.

Algoritmo	LIME	IME	KernelSHAP	TreeSHAP
Explicaciones contrastables	Sí	No	No	No
Consistentes	No	Sí	Sí	Sí
Válido para cualquier modelo	Sí	Sí	Sí	No
Fundamento teórico	No	Sí	Sí	Sí
Distribución justa	No	Sí	Sí	Sí
Precisión local	No	Sí	Sí	Sí
Explicaciones dispersas	Sí	No	Sí	Sí
Cumple propiedad jugador faltante	No	Sí	Sí	No
Tiene en cuenta la dependencia entre características	No	No	No	Sí

Los métodos basados en el valor de Shapley están emergiendo como los mejores métodos de interpretación de predicciones debido a las propiedades que satisfacen.

Podemos hacer una comparativa de las atribuciones obtenidas para los métodos basados en el valor de Shapley. Vemos las distintas atribuciones dada por cada método reflejadas en la figuras 4.1.



(a) Problema de regresión

(b) Problema de clasificación

Figura 4.1: Comparativa atribuciones

Además, para los métodos basados en el valor de Shapley hemos estudiado las distintas formulaciones del juego resultantes para cada uno, y llegamos a la conclusión de que podemos definir una formulación unificada para todos ellos:

$$v_D^x = \mathbb{E}_{R\sim D}[f(z(x, R, S))] - \mathbb{E}_{R\sim D}[f(R)]$$

donde en función de como se trate al subconjunto de características desconocidas R, obtenemos la formulación del juego de uno de los métodos estudiados.

Por ejemplo cuando $R \sim D = U$, recuperamos la formulación del juego descrita en IME.

Con la más rápida adaptación de las técnicas de machine learning a distintos ámbitos de la vida cotidiana, no cabe duda, de que en los próximos años se seguirán desarrollando nuevas técnicas de predicción, para mejorar la capacidad de los algoritmos actuales.

Así mismo, se prevee también, un crecimiento, del desarrollo de técnicas para la interpretabilidad del machine learning. La limitación de la explicabilidad y el auge de las técnicas de machine learning, provocan que el campo de la interpretabilidad de machine learning sea una de las grandes líneas de investigación en los próximos años.

Capítulo 5

Referencias

[1] Molnar, Christoph. Ïnterpretable machine learning. A Guide for Making Black Box Models Explainable", 2019. https://christophm.github.io/interpretable-ml-book/.

[2] Luke Merrick and Ankur Taly. The explanation game: Explaining machine learning models with cooperative game theory. arXiv preprint arXiv:1909.08128, 2019.

[3] ¿Qué es Machine Learning?, https://cleverdata.io/que-es-machine-learning-big-data/

[4] Libro Online de IAAR, https://iaarbook.github.io/ML/

[5] Interpretabilidad de los modelos de machine learning https://quanam.com/interpretabilidad-de-los-modelos-de-machine-learning-primera-parte/

[6] Doshi-Velez, Finale, and Been Kim. "Towards a rigorous science of interpretable machine learning,"no. Ml: 1–13. http://arxiv.org/abs/1702.08608 (2017)

[7] Friedman, Jerome, Trevor Hastie, and Robert Tibshirani. "The elements of statistical learning". www.web.stanford.edu/ hastie/ElemStatLearn/ (2009).

[8] The mathematics of decision trees, Random Forest and Feature Importance in Scikit-learn and Spark, https://towardsdatascience.com/the-mathematics-of-decision-trees-random-forest-and-feature-importance-in-scikit-learn-and-spark-f2861df67e3

[9] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin, 'Why should I trust you?: Explaining the predictions of any classifier', in Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining, pp. 1135–1144. ACM, (2016).

[10] Decrypting your machine learning model using LIME, https://towardsdatascience.com/decrypting-your-machine-learning-model-using-lime-5adc035109b5

[11] Teoría de juegos cooperativos, http://bibing.us.es/proyectos/abreproy/11847/fichero/Volumen+1

[12] Juegos TU, http://eio.usc.es/eipc1/BASE/BASEMASTER/FORMULARIOS-PHP-DPTO/MATERIALES/Mat_G101145171_JuegosTU.pdf

[13] Lloyd S Shapley, 'A value for n-person games', Contributions to the Theory of Games, 2(28), 307–317, (1953).

[14] Erik Strumbelj and Igor Kononenko, 'An efficient explanation of individual classifications using game theory', Journal of Machine Learning Research, 11, 1–18, (2010).

[15] Erik Strumbelj and Igor Kononenko, 'Explaining prediction models and individual predictions with feature contributions', Knowledge and information systems, 41(3), 647–665, (2014)

[16] Scott M Lundberg and Su-In Lee, 'A unified approach to interpreting model predictions', in Advances in Neural Information Processing Systems, pp. 4765–4774, (2017).

[17] Scott M Lundberg, Gabriel G Erion, and Su-In Lee, 'Consistent individualized feature attribution for tree ensembles', arXiv preprint arXiv:1802.03888, (2018).

[18] SHAP docs, https://shap.readthedocs.io/en/latest/.

[19] Machine Learning in Python, https://scikit-learn.org/stable/.

[20] UCI Machine Learning Repository, archive.ics.uci.edu/ml/index.php.

[21] Understanding how IME explains predictions, https://towardsdatascience.com/understanding-how-ime-shapley-values-explains-predictions-d75c0fceca5a.

[22] Understanding the SHAP interpretation method: Kernel SHAP, https://data4thought.com/kernel_shap.html.

[23] Alibi: KernelSHAP, https://docs.seldon.io/projects/alibi/en/stable/methods/KernelSHAP.html.

[24] Cracking open the black box – Part III, https://deep-and-shallow.com/2019/11/24/interpretability-cracking-open-the-black-box-part-iii/

[25] Package 'iml', https://cran.r-project.org/web/packages/iml/iml.pdf

Apéndice A

Anexo

A.1. Datos

En este trabajo, todos los modelos y todas las técnicas han sido aplicadas a dos conjuntos de datos reales de acceso gratuito que puedes encontrar fácilmente online.

A.1.1. Bike rentals-Regresión

Este conjunto de datos contiene el número de bicis alquiladas por hora de la compañía de alquiler de bicis Capital-Bikeshare en Washington D.C., con información acerca de las condiciones meteorologicas y el calendario, de los años 2011 y 2012.

El objetivo es predecir cuántas bicis se alquilarán en función del tiempo, calendario y hora en 2012. Para ello dividiremos los datos en dos. Los datos de 2011, con 8645 registros, serán los datos de entrenamiento, mientras que usaremos los de 2012 como datos de test, con 8734 registros.

Puedes encontrar los datos en UCI Machine Learning Repository.

Características usadas

A continuación, vemos un listado de las características usadas en este trabajo:

- Count: Número de bicis alquiladas, incluyendo usuarios registrados y casuales. Count será el objetivo en el problema de regresión.
- Season: Estación del año. Toma los valores: SPRING, SUMMER, FALL o WINTER.
- Yr: Año. 2011 o 2012.
- Mnth: Mes del año (January to December).
- hr: Hora del día (0 a 23).
- holiday: Indica si el día es considerado festivo o no.
- weekday: Día de la semana (Monday to Sunday).
- workingday: Indica si el día es laborable o no.
- weathersit: Clima en ese momento. Toma los valores: SNOW/STORM, RAIN, MISTY, GOOD.

- temp: Temperatura en grados Celsius.
- hum: Humedad relativa en porcentaje(0 a 100).
- windspeed: Velocidad del viento medida en km por hora.

A.1.2. Adult-Clasificación

Este conjunto de datos fue extraido de la oficina del censo de Estados Unidos en 1994, y el objetivo es predecir si un adulto gana más de 50.000$ al año basándose en atributos como la educación, horas de trabajo a la semana, etc.

En el repositorio de UCI machine learning encontramos los datos ya divididos en entrenamiento y test, por lo que utilizamos los datos de test para validar nuestro modelo. Eliminamos todas las columnas que contenían datos NA, y nos quedaban 30162 instancias para entrenar el modelo y 15060 para testear el modelo.

Características usadas Las características usadas de esta base de datos son la siguientes:

- Age: Edad de la persona.
- Workclass: Área de trabajo de la persona. Toma los valores: Gov, Private, Self-emp y Other.
- fnlwgt: Peso que indica la cantidad de población representada por esa instancia.
- Education-num: Nivel de educación que toma valores del 1 al 16.
- marital-status: Estado civil de la persona: Married, Never-married, Separated, Widowed.
- relation-ship: Wife, Husband, Unmarried, Not-in-family, Own-child, Other-Relative.
- Race: Raza de la persona: Black, White, Amer-Indian-Eskimo, Asian-Pac-Islander, Other.
- Sex: Sexo de la persona: Male, Female.
- Capital-gain: Ganancias de capital.
- Capital-loss: Pérdida de capital.
- hours_per_week: Cantidad de horas medias trabajadas por semana.

A.2. Códigos

A.2.1. Regresión

Preprocesamiento

```
import numpy as np
import pandas as pd
# Cargamos los datos
df = pd.read_csv('hour.csv')
# Transformamos las características ordinales a categóricas
df.weekday =df.weekday.replace({0:"SUN", 1:"MON", 2:"TUE",
                                3:"WED", 4:"THU", 5:"FRI", 6:"SAT"})
df.holiday=df.holiday.replace({0:"NO HOLIDAY", 1:"HOLIDAY"})
df.workingday= df.workingday.replace({0:"NO WORKING DAY", 1:"WORKING DAY"})
df.season= df.season.replace({1:"SPRING", 2:"SUMMER", 3:"FALL", 4:"WINTER"})
df.weathersit= df.weathersit.replace({1:"GOOD", 2:"MISTY", 3:"RAIN",
                                      4:"STORM/SNOW"})
df.mnth= df.mnth.replace({1:"JAN", 2:"FEB", 3:"MAR", 4:"APR", 5:"MAY",
                          6:"JUN", 7:"JUL", 8:"AUG", 9:"SEP", 10:"OKT",
                          11:"NOV", 12:"DEC"})
df.yr= df.yr.replace({0:2011, 1:2012})
# Desestandarizamos las características numéricas
df['temp'] = df['temp'].map(lambda x :x*(39-(-8))+(-8))
df['atemp'] = df['atemp'].map(lambda x :x*(50-(16))+(16))
df['windspeed']=67*df['windspeed']
df['hum']=100*df['hum']
# Eliminamos las siguientes características
rent_bike=df.drop(columns=['instant', 'dteday', 'registered', 'casual', 'atemp'])
# Escribimos nuestros datos en un excel nuevo
rent_bike.to_csv('rental_bike_hour_CAT.csv', index = False)
```

Regresión Lineal

```
import numpy as np
import pandas as pd
from sklearn import linear_model
from sklearn import preprocessing
from sklearn.metrics import mean_squared_error, r2_score
import matplotlib.pyplot as plt
# Leemos los datos
df = pd.read_csv('rental_bike_hour_CAT.csv')
# Separamos el objetivo de las características
X = df.drop(columns = 'cnt')
Y = df['cnt']
#la regresión lineal no capta relaciones no lineales, por
#lo que discretizamos la característica 'hr' en 4 intervalos
f1 = lambda x: ("0-5" if (x < 5.5) else "6-11" if (x > 5.5 and x < 11.5)
                else "12-17" if (x > 11.5 and x < 17.5) else "18-23")
X['hr']=X['hr'].map(f1)
#estandarizamos las características cuantitativas
```

```
estandarizar = preprocessing.StandardScaler()
def scaleColumns(df, cols_to_scale):
    for col in cols_to_scale:
        df[col] = pd.DataFrame(estandarizar.fit_transform(pd.DataFrame(df[col])),
                               columns=[col])
    return df
X = scaleColumns(X,['temp','hum', 'windspeed'])

#weighted effect coding para las características categóricas
categorical_features = np.argwhere(np.array([len(set(X.values[:,x]))
for x in range(X.values.shape[1])]) <= 15).flatten()
feature_names = X.columns
X2 = X.copy()
for feature in categorical_features:
    get_onehot_features = pd.get_dummies(X2.iloc[:,feature],
                                         prefix = feature_names[feature],
                                         prefix_sep = '_')
    gen_effect_features = get_onehot_features.iloc[:,:-1]
    gen_effect_features[np.all(gen_effect_features == 0,
                               axis=1)] = -1.
    for col in gen_effect_features:
        a = gen_effect_features[col].value_counts()[1]
        b = get_onehot_features.iloc[:,-1].value_counts()[1]
        gen_effect_features[col] = ([a/b*x if x==-1. else x
                                     for x in gen_effect_features[col]])
    X2 = pd.concat([X2, gen_effect_features],
          axis=1)
X = X2.copy()
for feature in categorical_features:
    X = X.drop(columns = feature_names[feature])

# Separamos los datos de entrenamiento (2011) de
# los datos de test (2012)
train= X[X['yr_2011']>0.]
test= X[X['yr_2011']<0.]
labels_train = Y[:train.shape[0]]
labels_test = Y[train.shape[0]:]
# Creamos el objeto de Regresión Linear
regr = linear_model.LinearRegression()

# Entrenamos nuestro modelo
regr.fit(train, labels_train)

# Obtenemos los resultados del modelo de regresión lineal
y_pred = regr.predict(test)
SSE = mean_squared_error(labels_test, y_pred)
medidas = pd.DataFrame({'name':['Intercept','SSE', 'SST', 'R-Cuadrado'],
                        'value':[regr.intercept_,
                                 mean_squared_error(labels_test, y_pred),
                                 np.mean((labels_test.mean() - labels_test) ** 2),
```

```
                                r2_score(labels_test, y_pred)]})
# Los exportamos a un excel
medidas.to_csv('./RESULTADOS/Regresion/medidas.csv',
               index = False, sep=";", decimal =",")
#Coeficientes de la regresión lineal que también exportamos
coefficients=pd.DataFrame({'name':list(X),'value':regr.coef_})
coefficients.to_csv('./RESULTADOS/Regresion/coefficientsRegLin.csv',
                    index = False, sep=";", decimal =",")

# Calculamos los efectos de las características
feature_effect = X.loc[:,:]*regr.coef_
# Resumimos los efectos de las categóricas en una sola
feature_effect['season'] = (feature_effect['season_SPRING'] +
                            feature_effect['season_SUMMER'] +
                            feature_effect['season_FALL'])
feature_effect['holiday'] = feature_effect['holiday_HOLIDAY']
feature_effect['workingday'] = feature_effect['workingday_NO WORKING DAY']
feature_effect['yr'] = feature_effect['yr_2011']
feature_effect['weathersit'] = (feature_effect['weathersit_MISTY'] +
                                feature_effect['weathersit_RAIN'] +
                                feature_effect['weathersit_GOOD'])
feature_effect['mnth'] = (feature_effect['mnth_AUG'] +
                          feature_effect['mnth_DEC'] + feature_effect['mnth_FEB']
                          + feature_effect['mnth_JAN']+ feature_effect['mnth_JUL'] +
                          feature_effect['mnth_JUN'] + feature_effect['mnth_MAR'] +
                          feature_effect['mnth_MAY'] + feature_effect['mnth_NOV'] +
                          feature_effect['mnth_OKT'] + feature_effect['mnth_APR'])
feature_effect['weekday'] = (feature_effect['weekday_MON'] +
                             feature_effect['weekday_SAT'] +
                             feature_effect['weekday_SUN'] +
                             feature_effect['weekday_THU'] +
                             feature_effect['weekday_TUE'] +
                             feature_effect['weekday_FRI'])
feature_effect['hr'] = (feature_effect['hr_0-5'] + feature_effect['hr_12-17'] +
                        feature_effect['hr_18-23'])

# Eliminamos las distintas categorías
feature_effect = feature_effect.drop(columns = ['season_FALL',
        'season_SPRING', 'season_SUMMER', 'yr_2011', 'mnth_APR',
        'mnth_AUG', 'mnth_DEC', 'mnth_FEB', 'mnth_JAN',
        'mnth_JUL', 'mnth_JUN', 'mnth_MAR', 'mnth_MAY',
        'mnth_NOV', 'mnth_OKT', 'hr_0-5', 'hr_12-17', 'hr_18-23',
        'holiday_HOLIDAY', 'weekday_FRI', 'weekday_MON', 'weekday_SAT',
        'weekday_SUN', 'weekday_THU', 'weekday_TUE',
        'workingday_NO WORKING DAY', 'weathersit_GOOD', 'weathersit_MISTY',
        'weathersit_RAIN'])

# Predicción instancia 11728
ejemplo=X.iloc[11728].values.reshape(1,-1)
y_pred1 = regr.predict(ejemplo)
```

```
# Exportamos los efectos de la instancia de interés a un excel
effectsejemplo=pd.DataFrame({'name':list(feature_effect),
                             'value':feature_effect.iloc[11728].values})
effectsejemplo.to_csv('./RESULTADOS/Regresion/efectos11728.csv',
                      index = False, sep=";", decimal =",")

# Representamos los efectos de las características
# en un diagrama de cajas con los de la instancia 11728
plt.figure(figsize=(12,10), dpi= 80)
feature_effect.boxplot(column = ['temp', 'hum', 'windspeed',
        'season', 'holiday', 'workingday', 'yr',
        'weathersit', 'mnth', 'weekday', 'hr'], vert = False)
plt.scatter(feature_effect.iloc[11728].values,
            range(1,12), c='r', marker='X')

# Representación final de las contribuciones de las características
# para la predicción de la instancia 11728
feature_effect.iloc[11728].plot(kind='barh',width=0.8,
            color='grey', figsize=(8,8), grid=True)
```

Árbol de decisión

```
import numpy as np
import pandas as pd
from sklearn.tree import DecisionTreeRegressor
from sklearn import metrics
import matplotlib.pyplot as plt
# Leemos los datos
df = pd.read_csv('rental_bike_hour_CAT.csv')
# Separamos objetivo de las características
X = df.drop(columns = 'cnt')
feature_names = X.columns
Y = df['cnt']
# One-hot-encoding para las variables categóricas
def prepare_data_for_model(raw_dataframe, target_columns, drop_first = True):

    dataframe_dummy = pd.get_dummies(raw_dataframe, columns=target_columns,
                                     drop_first=drop_first)
    return (dataframe_dummy)

X = prepare_data_for_model(X, target_columns = ['season', 'weathersit',
                         'yr', 'workingday', 'holiday', 'mnth', 'weekday'],
                            drop_first = True)

# Separamos datos de entrenamiento (2011)
# de datos de test (2012)
train= X[X['yr_2012']==0]
test= X[X['yr_2012']==1]
labels_train = Y[:train.shape[0]]
```

```
labels_test = Y[train.shape[0]:]
# Creamos el árbol de Decisión con profundidad máxima igual a 3
clf = DecisionTreeRegressor(max_depth=3)

# Entrenamos nuestro modelo
clf.fit(train, labels_train)

# Para visualizar el árbol
from sklearn import tree
from sklearn.tree import export_graphviz
import graphviz
dot_data = tree.export_graphviz(clf, out_file=None,
                filled=True, rounded=True,
                special_characters=True,feature_names = X.columns)
graph = graphviz.Source(dot_data) #este es el árbol

# Exportamos los resultados obtenidos
y_pred2 = clf.predict(X)
prediccionesDTRegressor = pd.DataFrame({'name':['Valor medio de referencia',
                                                'Valor Predicho', 'Valor Real'],
                        'value':[labels_train.mean(), y_pred2[11728],
                                 Y.iloc[11728]]})
prediccionesDTRegressor.to_csv('./RESULTADOS/DecisionTree/prediccionesDT.csv',
                               index = False, sep=";", decimal =",")

# Calculamos las contribuciones (a mano) y las representamos
contribuciones = df.iloc[0].copy()
contribuciones = contribuciones[:-1]
contribuciones.iloc[:]=0
contribuciones.loc['hr']=-56.398
contribuciones.loc['temp']=53.827
contribuciones.plot(kind='barh', width=0.8,
                color='grey', figsize=(8,8), grid=True)
```

LIME

```
import numpy as np
import pandas as pd
from sklearn.ensemble import RandomForestRegressor
from sklearn import metrics
from sklearn import preprocessing
import matplotlib.pyplot as plt
# Leemos los datos como pandas y como numpy
df = pd.read_csv('rental_bike_hour_CAT.csv')
datos = np.genfromtxt('rental_bike_hour_CAT.csv', delimiter=',',
                      dtype=str, skip_header=1)

#Separamos objetivo de las características
X = df.drop(columns = 'cnt')
feature_names = X.columns
```

```
Y = df['cnt']
labels = datos[:,-1]
data = datos[:,:-1]

# si una característica tiene menos de 15 valores
# es considerada categórica
categorical_features = np.argwhere(np.array([len(set(X.values[:,x]))
for x in range(X.values.shape[1])]) <= 15).flatten()
# guardamos los nombres de las categorías de cada característica
categorical_names = {}
for feature in categorical_features:
    le = preprocessing.LabelEncoder()
    le.fit(data[:, feature])
    data[:, feature] = le.transform(data[:, feature])
    categorical_names[feature] = le.classes_
data = data.astype(float)

# Separamos datos de entrenamiento (2011)
# de datos de test (2012)
train= data[data[:,1]==0]
test= data[data[:,1]==1]
labels_train = labels[:train.shape[0]]
labels_test = labels[train.shape[0]:]

# One-hot-encoding a las variables categóricas para poder aplicar
# Random Forest
encoder = preprocessing.OneHotEncoder(categorical_features=categorical_features)
encoder.fit(data)
encoded_train = encoder.transform(train)
# Aplicamos RF a nuestros datos
random_forest = RandomForestRegressor(n_estimators=100)
random_forest.fit(encoded_train, labels_train)
predict_fn_rf = lambda x: random_forest.predict(encoder.transform(x)).astype(float)

#Usamos LIME para explicar las predicciones del modelo
import lime
import lime.lime_tabular

explainer = lime.lime_tabular.LimeTabularExplainer(train, feature_names=X.columns,
                                        class_names=['cnt'],
                                        categorical_features=categorical_features,
                                        categorical_names=categorical_names,
                                        verbose=True, mode='regression')

# Explicamos la instancia 11728
choosen_instance = data[11728]
exp = explainer.explain_instance(choosen_instance, predict_fn_rf)
#Visualizamos los resultados obtenidos
exp.show_in_notebook(show_all=False)
```

IME

```
import numpy as np
import pandas as pd
from sklearn.ensemble import RandomForestRegressor
from sklearn import metrics
# Leemos los datos
df = pd.read_csv('rental_bike_hour_CAT.csv')
#Separamos objetivo de las características
X = df.drop(columns = 'cnt')
feature_names = X.columns
Y = df['cnt']
# Codificamos las variables categóricas (One hot Encoding)
X2 = X.copy()
def prepare_data_for_model(raw_dataframe, target_columns, drop_first = True):

    dataframe_dummy = pd.get_dummies(raw_dataframe, columns=target_columns,
                                     drop_first=drop_first)
    return (dataframe_dummy)

X2 = prepare_data_for_model(X2,
                target_columns = ['season', 'weathersit', 'yr', 'workingday',
                        'holiday', 'mnth', 'weekday'], drop_first = True)

# Separamos datos de entrenamiento (2011)
# de datos de test (2012)
train= X2[X2['yr_2012']==0]
test= X2[X2['yr_2012']==1]
labels_train = Y[:train.shape[0]]
labels_test = Y[train.shape[0]:]

# Entrenamos nuestro modelo de RF
random_forest = RandomForestRegressor(n_estimators=100)
random_forest.fit(train, labels_train)

# Creamos IME para un RF Regressor
import random
def IME_RFReg(instance, k, df):
    predict_fn_rf = lambda x: random_forest.predict(x.reshape(1,-1)).astype(float)
    Sh = np.zeros(df.shape[1])
    feature_names = df.columns
    for col in range(df.shape[1]):
        for muestra in range(k):
            secuencia = random.sample(range(df.shape[1]), k=df.shape[1])
            x1 = np.zeros(df.shape[1])
            x2 = np.zeros(df.shape[1])
            for col2 in range(df.shape[1]):
                a = np.array(random.choice(df[feature_names[col2]].unique()), dtype=float)
                x1[col2] = (instance[col2] if
                            secuencia.index(col2) <= secuencia.index(col) else a)
```

```
                x2[col2] = (instance[col2] if
                            secuencia.index(col2) < secuencia.index(col) else a)
            Sh[col] = Sh[col] + predict_fn_rf(x1)[0] - predict_fn_rf(x2)[0]
        Sh[col] = Sh[col]/k
        prediccion = random_forest.predict(instance.values.reshape(1,-1))
        expected_value = prediccion - Sh.sum()
    return Sh, expected_value

# Explicamos las predicciones de RF mediante IME
explainer= IME_RFReg(instance = X2.iloc[11728], k=10000, df=X2)

# Esta función devuelve los valores shapley por cada categoría
# de las distintas características categóricas.
# Por la aditividad del valor Shapley tiene sentido pensar que
# el valor de Shapley de una característica categórica es la suma
# de los valores Shapley de sus categorías
# Calculemos los valores de Shapley para las características
ejemplo=X2.iloc[0].copy()
ejemplo[:]=explainer[:-1]
ejemplo['season'] = (ejemplo['season_SPRING'] +
        ejemplo['season_SUMMER'] + ejemplo['season_WINTER'])
ejemplo['holiday'] = ejemplo['holiday_NO HOLIDAY']
ejemplo['workingday'] =ejemplo['workingday_WORKING DAY']
ejemplo['yr'] =ejemplo['yr_2012']
ejemplo['weathersit'] = (ejemplo['weathersit_MISTY'] +
        ejemplo['weathersit_RAIN'] + ejemplo['weathersit_STORM/SNOW'])
ejemplo['mnth'] = (ejemplo['mnth_AUG'] +
        ejemplo['mnth_DEC'] + ejemplo['mnth_FEB'] +
        ejemplo['mnth_JAN']+ ejemplo['mnth_JUL'] +
        ejemplo['mnth_JUN'] + ejemplo['mnth_MAR'] +
        ejemplo['mnth_MAY'] + ejemplo['mnth_NOV'] +
        ejemplo['mnth_OKT'] + ejemplo['mnth_SEP'])
ejemplo['weekday'] = (ejemplo['weekday_MON'] + ejemplo['weekday_SAT'] +
        ejemplo['weekday_SUN'] + ejemplo['weekday_THU'] +
        ejemplo['weekday_TUE'] + ejemplo['weekday_WED'])
# Eliminamos las categorías
ejemplo = ejemplo.drop(['season_WINTER', 'season_SPRING',
        'season_SUMMER', 'yr_2012', 'mnth_SEP', 'mnth_AUG',
        'mnth_DEC', 'mnth_FEB', 'mnth_JAN', 'mnth_JUL', 'mnth_JUN',
        'mnth_MAR', 'mnth_MAY', 'mnth_NOV', 'mnth_OKT',
        'holiday_NO HOLIDAY', 'weekday_WED', 'weekday_MON',
        'weekday_SAT', 'weekday_SUN', 'weekday_THU', 'weekday_TUE',
        'workingday_WORKING DAY', 'weathersit_STORM/SNOW',
        'weathersit_MISTY', 'weathersit_RAIN'],axis=0)

# Representamos las contribuciones de las características en una gráfica
ejemplo.plot(kind='barh', width=0.8, color='grey', figsize=(8,8), grid=True)

# Guardamos la predicción media y la predicción actual
prediccion = random_forest.predict(X2.iloc[11728].values.reshape(1,-1))
```

```
expected_value = explainer[-1]
predicciones = pd.DataFrame({'name':['Predicción media','Predicción actual'],
                        'value':[expected_value[0],
                                 prediccion[0]]})
```

KernelSHAP

```
import numpy as np
import pandas as pd
from sklearn.ensemble import RandomForestRegressor
from sklearn import metrics
from sklearn import preprocessing
import matplotlib.pyplot as plt
# Leemos los datos como pandas y como numpy
df = pd.read_csv('rental_bike_hour_CAT.csv')
datos = np.genfromtxt('rental_bike_hour_CAT.csv', delimiter=',',
                      dtype=str, skip_header=1)

#Separamos objetivo de las características
X = df.drop(columns = 'cnt')
feature_names = X.columns
Y = df['cnt']
labels = datos[:,-1]
data = datos[:,:-1]

# si una característica tiene menos de 15 valores
# es considerada categórica
categorical_features = np.argwhere(np.array([len(set(X.values[:,x]))
for x in range(X.values.shape[1])]) <= 15).flatten()
# guardamos los nombres de las categorías de cada característica
categorical_names = {}
for feature in categorical_features:
    le = preprocessing.LabelEncoder()
    le.fit(data[:, feature])
    data[:, feature] = le.transform(data[:, feature])
    categorical_names[feature] = le.classes_
data = data.astype(float)

# Separamos datos de entrenamiento (2011)
# de datos de test (2012)
train= data[data[:,1]==0]
test= data[data[:,1]==1]
labels_train = labels[:train.shape[0]]
labels_test = labels[train.shape[0]:]

# One-hot-encoding a las variables categóricas para poder aplicar
# Random Forest
encoder = preprocessing.OneHotEncoder(categorical_features=categorical_features)
encoder.fit(data)
encoded_train = encoder.transform(train)
```

```
# Aplicamos RF a nuestro modelo
random_forest = RandomForestRegressor(n_estimators=100)
random_forest.fit(encoded_train, labels_train)
predict_fn_rf = lambda x: random_forest.predict(encoder.transform(x)).astype(float)

# Importamos el paquete SHAP
import shap
shap.initjs()
# Explicamos las predicciones de RF mediante KernelSHAP
explainer = shap.KernelExplainer(predict_fn_rf, train)

# Calculamos los valores SHAP
shap_values = explainer.shap_values(data[11728], nsamples=100,
                                    l1_reg="num_features(11)")
# Visualización propia de SHAP
shap.force_plot(explainer.expected_value, shap_values, features = X.iloc[11728])

# Representamos las contribuciones finales en la gráfica
contribuciones = df.iloc[0].copy()
contribuciones = contribuciones[:-1]
contribuciones.iloc[:] = shap_values
contribuciones.plot(kind='barh', width=0.8, color='grey',
                    figsize=(8,8), grid=True)
```

TreeSHAP

```
import numpy as np
import pandas as pd
from sklearn.ensemble import RandomForestRegressor
from sklearn import metrics
# Leemos los datos
df = pd.read_csv('rental_bike_hour_CAT.csv')
#Separamos objetivo de las características
X = df.drop(columns = 'cnt')
feature_names = X.columns
Y = df['cnt']
# Codificamos las variables categóricas (One hot Encoding)
X2 = X.copy()
def prepare_data_for_model(raw_dataframe, target_columns, drop_first = True):

    dataframe_dummy = pd.get_dummies(raw_dataframe, columns=target_columns,
                                     drop_first=drop_first)
    return (dataframe_dummy)

X2 = prepare_data_for_model(X2,
                target_columns = ['season', 'weathersit', 'yr', 'workingday',
                        'holiday', 'mnth', 'weekday'], drop_first = True)

# Separamos datos de entrenamiento (2011)
# de datos de test (2012)
```

```
train= X2[X2['yr_2012']==0]
test= X2[X2['yr_2012']==1]
labels_train = Y[:train.shape[0]]
labels_test = Y[train.shape[0]:]

# Entrenamos nuestro modelo de RF
random_forest = RandomForestRegressor(n_estimators=100)
random_forest.fit(train, labels_train)

# Importamos el paquete SHAP
import shap
shap.initjs()

# Explicamos las predicciones de RF mediante TreeSHAP
explainer = shap.TreeExplainer(random_forest)

# Calculamos los valores Shap
shap_values = explainer.shap_values(X2.iloc[11728])
# Visualización propia de SHAP
shap.force_plot(explainer.expected_value, shap_values, X2.iloc[11728])
# TreeSHAP devuelve los valores SHAP por cada categoría
# de las distintas características categóricas.
# Por la aditividad del valor Shapley tiene sentido pensar que
# el valor de Shapley de una característica categórica es la suma
# de los valores Shapley de sus categorías
# Calculemos los valores de Shapley para las características
ejemplo=X2.iloc[0].copy()
ejemplo[:]=shap_values
ejemplo['season'] = (ejemplo['season_SPRING'] +
        ejemplo['season_SUMMER'] + ejemplo['season_WINTER'])
ejemplo['holiday'] = ejemplo['holiday_NO HOLIDAY']
ejemplo['workingday'] =ejemplo['workingday_WORKING DAY']
ejemplo['yr'] =ejemplo['yr_2012']
ejemplo['weathersit'] = (ejemplo['weathersit_MISTY'] +
        ejemplo['weathersit_RAIN'] + ejemplo['weathersit_STORM/SNOW'])
ejemplo['mnth'] = (ejemplo['mnth_AUG'] +
        ejemplo['mnth_DEC'] + ejemplo['mnth_FEB'] +
        ejemplo['mnth_JAN']+ ejemplo['mnth_JUL'] +
        ejemplo['mnth_JUN'] + ejemplo['mnth_MAR'] +
        ejemplo['mnth_MAY'] + ejemplo['mnth_NOV'] +
        ejemplo['mnth_OKT'] + ejemplo['mnth_SEP'])
ejemplo['weekday'] = (ejemplo['weekday_MON'] + ejemplo['weekday_SAT'] +
        ejemplo['weekday_SUN'] + ejemplo['weekday_THU'] +
        ejemplo['weekday_TUE'] + ejemplo['weekday_WED'])
# Eliminamos las categorías
ejemplo = ejemplo.drop(['season_WINTER', 'season_SPRING',
        'season_SUMMER', 'yr_2012', 'mnth_SEP', 'mnth_AUG',
        'mnth_DEC', 'mnth_FEB', 'mnth_JAN', 'mnth_JUL', 'mnth_JUN',
        'mnth_MAR', 'mnth_MAY', 'mnth_NOV', 'mnth_OKT',
        'holiday_NO HOLIDAY', 'weekday_WED', 'weekday_MON',
```

```
        'weekday_SAT', 'weekday_SUN', 'weekday_THU', 'weekday_TUE',
        'workingday_WORKING DAY', 'weathersit_STORM/SNOW',
        'weathersit_MISTY', 'weathersit_RAIN'],axis=0)

# Representamos las contribuciones de las características en una gráfica
ejemplo.plot(kind='barh', width=0.8, color='grey', figsize=(8,8), grid=True)
```

A.2.2. Clasificación

Preprocesamiento

```
import numpy as np
import pandas as pd
#Leemos los datos de train y de test que nos encontramos en el repositorio
train = pd.read_csv('adult_train.txt', header = None, skipinitialspace=True)
test = pd.read_csv('adult_test.txt', header = None, skiprows = 1,
                   skipinitialspace=True)
train = train.replace({'?':np.nan})
test = test.replace({'?':np.nan})
train = train.dropna()
test = test.dropna()
feature_names = ["age", "workclass", "fnlwgt", "education", "education-num",
                 "marital-status", "occupation", "relationship", "race","sex",
                 "capital-gain", "capital-loss","hours-per-week",
                 "native countries", "income"]
train.columns = feature_names
test.columns = feature_names
# Agrupamos algunas categorías similares dentro de unas características
train = train.replace({'State-gov':'Gov', 'Local-gov':'Gov',
        'Federal-gov':'Gov', 'Self-emp-not-inc': 'Self-emp',
                       'Self-emp-inc': 'Self-emp', 'Without-pay': 'Other'})
train = train.replace({'Married-civ-spouse':'Married',
                       'Married-spouse-absent':'Married',
                       'Married-AF-spouse':'Married', 'Divorced':'Separated'})
train = train.replace({'<=50K':'<=50K', '>50K':'>50K'})
test = test.replace({'State-gov':'Gov', 'Local-gov':'Gov',
                     'Federal-gov':'Gov', 'Self-emp-not-inc': 'Self-emp',
                     'Self-emp-inc': 'Self-emp', 'Without-pay': 'Other'})
test = test.replace({'Married-civ-spouse':'Married',
                     'Married-spouse-absent':'Married',
                     'Married-AF-spouse':'Married', 'Divorced':'Separated'})
test = test.replace({'<=50K.':'<=50K', '>50K.':'>50K'})
# Eliminamos las siguientes columnas
train = train.drop(columns = ['native countries', 'education', 'occupation'])
test = test.drop(columns = ['native countries', 'education', 'occupation'])
# Unimos los datos de entrenamiento y test
datos = pd.concat([train,test])
# Escribimos en un excel nuestros datos nuevos
datos.to_csv('adult_data.csv', index = False)
```

Árbol de decisión

```
import numpy as np
import pandas as pd
from sklearn import preprocessing
from sklearn.tree import DecisionTreeClassifier
from sklearn import metrics
# Leemos los datos
df = pd.read_csv('adult_data.csv')
# Separamos el objetivo de las columnas
X = df.drop(columns = 'income')
feature_names = X.columns
Y = df['income']

# Codificamos las características categóricas
def prepare_data_for_model(raw_dataframe, target_columns, drop_first = True):

    dataframe_dummy = pd.get_dummies(raw_dataframe, columns=target_columns,
                                     drop_first=drop_first)
    return (dataframe_dummy)

X = prepare_data_for_model(X,
            target_columns = ['workclass', 'marital-status', 'relationship',
                              'race', 'sex'],drop_first = True)
# Separamos los datos de entrenamiento de los datos de test
train = X.iloc[:30162]
test = X.iloc[30162:]
labels_train = Y[:train.shape[0]]
labels_test = Y[train.shape[0]:]
# Creamos el árbol de decisión con profundida máxima igual a 3
clf = DecisionTreeClassifier(max_depth=3)
# Entrenamos nuestro modelo
clf = clf.fit(train,labels_train)
# Para visualizar el árbol
from sklearn import tree
from sklearn.tree import export_graphviz
from IPython.display import Image
import graphviz
dot_data = tree.export_graphviz(clf, out_file=None,
                filled=True, rounded=True,
                special_characters=True,feature_names = train.columns,class_names=['-50K','+
graph = graphviz.Source(dot_data) # este es el árbol

#Gráfico importancia características
importances = clf.feature_importances_
indices = np.argsort(clf.feature_importances_)
features = train.columns
import matplotlib.pyplot as plt
plt.figure(1)
plt.title('Feature Importances')
```

```
plt.barh(range(len(indices)), importances[indices], color='b', align='center')
plt.yticks(range(len(indices)), features[indices])

# Exportamos los resultados obtenidos
y_pred = clf.predict(test)
prediccionesDT=pd.DataFrame({'name':['Valor Predicho',
                                     'Valor actual', 'Accuracy'],
                             'value':[y_pred[9118], labels_test.iloc[9118],
                                      metrics.accuracy_score(labels_test, y_pred)]})
prediccionesDT.to_csv('./RESULTADOS/DecisionTree/prediccionesDT.csv',
                      index = False, sep=";", decimal =",")

# Calculamos las contribuciones (a mano) y las representamos
contribuciones = df.iloc[0].copy()
contribuciones = contribuciones[:-1]
contribuciones.iloc[:]=0
contribuciones.loc['education-num']=0.280
contribuciones.loc['marital-status']=0.089
contribuciones.loc['capital-gain']=-0.036
contribuciones.plot(kind='barh', width=0.8,
                    color='grey', figsize=(8,8), grid=True)
```

LIME

```
import numpy as np
import pandas as pd
from sklearn import preprocessing
from sklearn.ensemble import RandomForestClassifier
from sklearn import metrics
data = np.genfromtxt('adult_data.csv', delimiter=',',
                     dtype=str, skip_header=1, encoding=None)
df = pd.read_csv('adult_data.csv')

# Separamos objetivo de las características
X = df.drop(columns = 'income')
feature_names = X.columns
labels = data[:,-1]
le = preprocessing.LabelEncoder()
le.fit(labels)
labels = le.transform(labels)
# Guardamos los nombres de las clases del objetivo
class_names = le.classes_
data = data[:,:-1]

# si una característica tiene menos de 12 valores
# es considerada como categórica
categorical_features = np.argwhere(np.array([len(set(X.values[:,x]))
for x in range(X.values.shape[1])]) <= 12).flatten()
#guardamos los nombres de las categorías de las características
categorical_names = {}
```

```
for feature in categorical_features:
    le = preprocessing.LabelEncoder()
    le.fit(data[:, feature])
    data[:, feature] = le.transform(data[:, feature])
    categorical_names[feature] = le.classes_
data = data.astype(float)
# Separamos los datos de entrenamiento de los datos de test
train = data[:30162,:]
test = data[30162:,:]
labels_train = labels[:30162]
labels_test = labels[30162:]
# One-hot-encoding a las variables categóricas para poder aplicar
# Random Forest
encoder = preprocessing.OneHotEncoder(categorical_features=categorical_features)
encoder.fit(data)
encoded_train = encoder.transform(train)

# Aplicamos RF a nuestros datos
random_forest = RandomForestClassifier(n_estimators=100)
random_forest.fit(encoded_train, labels_train)

predict_fn_rf = lambda x: random_forest.predict_proba(encoder.transform(x))

# Usamos LIME para explicar las predicciones del modelo
import lime
import lime.lime_tabular

explainer = lime.lime_tabular.LimeTabularExplainer(train,
                                    feature_names = feature_names,
                                    class_names = class_names,
                                    categorical_features = categorical_features,
                                    categorical_names=categorical_names,
                                    verbose=True)

# Explicamos la instancia 9118 de los datos de test
choosen_instance = test[9118]
exp = explainer.explain_instance(choosen_instance, predict_fn_rf)
# Visualizamos los resultados obtenidos
exp.show_in_notebook(show_all=False)
```

IME

```
import numpy as np
import pandas as pd
from sklearn.ensemble import RandomForestClassifier
from sklearn import metrics

# Leemos los datos
df = pd.read_csv('adult_data.csv')
```

```
# Separamos el objetivo de las características
X = df.drop(columns = 'income')
feature_names = X.columns
Y = df['income']
# Codificamos las variables categóricas (One-hot-encoding)
def prepare_data_for_model(raw_dataframe, target_columns, drop_first = True):

    # dummy all categorical fields
    dataframe_dummy = pd.get_dummies(raw_dataframe, columns=target_columns,
                                     drop_first=drop_first)
    return (dataframe_dummy)

# dummify categorical columns
X = prepare_data_for_model(X,
                           target_columns = ['workclass', 'marital-status',
                                             'relationship', 'race', 'sex'],
                           drop_first = True)

# Separamos datos de entrenamiento de datos de test
train = X.iloc[:30162]
test = X.iloc[30162:]
labels_train = Y[:train.shape[0]]
labels_test = Y[train.shape[0]:]

# Entrenamos nuestro modelo de RF
random_forest = RandomForestClassifier(n_estimators=100)
random_forest.fit(train, labels_train)

# Creamos IME para un RF Classifier
import random
def IME_RFCl(instance, k, df):
    predict_fn_rf = lambda x: random_forest.predict_proba(x.reshape(1,-1)).astype(float)
    Sh = np.zeros(df.shape[1])
    feature_names = df.columns
    for col in range(df.shape[1]):
        for muestra in range(k):
            secuencia = random.sample(range(df.shape[1]), k=df.shape[1])
            x1 = np.zeros(df.shape[1])
            x2 = np.zeros(df.shape[1])
            for col2 in range(df.shape[1]):
                a = np.array(random.choice(df[feature_names[col2]].unique()), dtype=float)
                x1[col2] = (instance[col2] if
                            secuencia.index(col2) <= secuencia.index(col) else a)
                x2[col2] = (instance[col2] if
                            secuencia.index(col2) < secuencia.index(col) else a)
            Sh[col] = Sh[col] + predict_fn_rf(x1)[0,1] - predict_fn_rf(x2)[0,1]
        Sh[col] = Sh[col]/k
        prediccion = random_forest.predict_proba(instance.values.reshape(1,-1))[0,1]
        expected_value = prediccion - Sh.sum()
    return Sh, expected_value
```

```
# Explicamos las predicciones de RF mediante IME
explainer= IME_RFCl(instance = test.iloc[9118], k=2000, df=X)


# Esta función devuelve los valores shapley por cada categoría
# de las distintas características categóricas.
# Por la aditividad del valor Shapley tiene sentido pensar que
# el valor de Shapley de una característica categórica es la suma
# de los valores Shapley de sus categorías
# Calculemos los valores de Shapley para las características
ejemplo=test.iloc[0].copy()
ejemplo[:]=explainer[0]
ejemplo['workclass'] = (ejemplo['workclass_Other'] +
                        ejemplo['workclass_Private'] +
                        ejemplo['workclass_Self-emp'])
ejemplo['marital-status'] = (ejemplo['marital-status_Never-married'] +
                             ejemplo['marital-status_Separated'] +
                             ejemplo['marital-status_Widowed'])
ejemplo['relationship'] = (ejemplo['relationship_Not-in-family'] +
                           ejemplo['relationship_Other-relative'] +
                           ejemplo['relationship_Own-child'] +
                           ejemplo['relationship_Unmarried'] +
                           ejemplo['relationship_Wife'])
ejemplo['race'] = (ejemplo['race_Asian-Pac-Islander'] +
                   ejemplo['race_Black'] + ejemplo['race_Other'] +
                   ejemplo['race_White'])
ejemplo['sex'] =ejemplo['sex_Male']
ejemplo = ejemplo.drop(['workclass_Other', 'workclass_Private',
        'workclass_Self-emp', 'marital-status_Never-married',
        'marital-status_Separated', 'marital-status_Widowed',
        'relationship_Not-in-family', 'relationship_Other-relative',
        'relationship_Own-child', 'relationship_Unmarried',
        'relationship_Wife', 'race_Asian-Pac-Islander', 'race_Black',
        'race_Other', 'race_White', 'sex_Male'],axis=0)

ejemplo.plot(kind='barh', width=0.8, color='grey',
             figsize=(8,8), grid=True)

# Guardamos la predicción media y predicción para la instancia
prediccion = random_forest.predict(test.iloc[9118].values.reshape(1,-1))
probabilidad_prediccion = random_forest.predict_proba(test.iloc[9118].values.reshape(1,-1))
expected_value = explainer[-1]
predicciones = pd.DataFrame({'name':['Predicción media','Probabilidad predicción', 'Predicci
                        'value':[expected_value, probabilidad_prediccion,
                                 prediccion[0]]})
```

KernelSHAP

```
import numpy as np
import pandas as pd
from sklearn import preprocessing
from sklearn.ensemble import RandomForestClassifier
from sklearn import metrics
data = np.genfromtxt('adult_data.csv', delimiter=',',
                     dtype=str, skip_header=1, encoding=None)
df = pd.read_csv('adult_data.csv')

# Separamos objetivo de las características
X = df.drop(columns = 'income')
feature_names = X.columns
labels = data[:,-1]
le = preprocessing.LabelEncoder()
le.fit(labels)
labels = le.transform(labels)
# Guardamos los nombres de las clases del objetivo
class_names = le.classes_
data = data[:,:-1]

# si una característica tiene menos de 12 valores
# es considerada como categórica
categorical_features = np.argwhere(np.array([len(set(X.values[:,x]))
for x in range(X.values.shape[1])]) <= 12).flatten()
#guardamos los nombres de las categorías de las características
categorical_names = {}
for feature in categorical_features:
    le = preprocessing.LabelEncoder()
    le.fit(data[:, feature])
    data[:, feature] = le.transform(data[:, feature])
    categorical_names[feature] = le.classes_
data = data.astype(float)
# Separamos los datos de entrenamiento de los datos de test
train = data[:30162,:]
test = data[30162:,:]
labels_train = labels[:30162]
labels_test = labels[30162:]
# One-hot-encoding a las variables categóricas para poder aplicar
# Random Forest
encoder = preprocessing.OneHotEncoder(categorical_features=categorical_features)
encoder.fit(data)
encoded_train = encoder.transform(train)

# Aplicamos RF a nuestros datos
random_forest = RandomForestClassifier(n_estimators=100)
random_forest.fit(encoded_train, labels_train)

predict_fn_rf = lambda x: random_forest.predict_proba(encoder.transform(x))
```

```
# Importamos el paquete SHAP
import shap
shap.initjs()
# Explicamos las predicciones de RF mediante KernelSHAP
# Create object that can calculate shap values
explainer = shap.KernelExplainer(predict_fn_rf, train)
# Calculamos los valores SHAP para la instancia 9118 de los datos de test
shap_values = explainer.shap_values(data[39280], nsamples=100,
                                    l1_reg="num_features(11)")
# Visualización propia de SHAP
shap.force_plot(explainer.expected_value[1], shap_values[1],
                features = X.iloc[39280])

# Representamos las contribuciones finales en la gráfica
contribuciones = df.iloc[0].copy()
contribuciones = contribuciones[:-1]
contribuciones.iloc[:] = shap_values[1]
contribuciones.plot(kind='barh', width=0.8,
                    color='grey', figsize=(8,8), grid=True)
```

TreeSHAP

```
import numpy as np
import pandas as pd
from sklearn.ensemble import RandomForestClassifier
from sklearn import metrics

# Leemos los datos
df = pd.read_csv('adult_data.csv')

# Separamos el objetivo de las características
X = df.drop(columns = 'income')
feature_names = X.columns
Y = df['income']
# Codificamos las variables categóricas (One-hot-encoding)
def prepare_data_for_model(raw_dataframe, target_columns, drop_first = True):

    # dummy all categorical fields
    dataframe_dummy = pd.get_dummies(raw_dataframe, columns=target_columns,
                                     drop_first=drop_first)
    return (dataframe_dummy)

# dummify categorical columns
X = prepare_data_for_model(X,
                           target_columns = ['workclass', 'marital-status',
                                             'relationship', 'race', 'sex'],
                           drop_first = True)

# Separamos datos de entrenamiento de datos de test
```

```
train = X.iloc[:30162]
test = X.iloc[30162:]
labels_train = Y[:train.shape[0]]
labels_test = Y[train.shape[0]:]

# Entrenamos nuestro modelo de RF
random_forest = RandomForestClassifier(n_estimators=100)
random_forest.fit(train, labels_train)

# Importamos el paquete SHAP
import shap
shap.initjs()

# Explicamos las predicciones de RF mediante TreeSHAP
explainer = shap.TreeExplainer(random_forest)

# Calculamos los valores SHAP
shap_values = explainer.shap_values(test.iloc[9118])
# Visualización propia de SHAP
shap.force_plot(explainer.expected_value[1], shap_values[1], test.iloc[9118])

# TreeSHAP devuelve los valores SHAP por cada categoría
# de las distintas características categóricas.
# Por la aditividad del valor Shapley tiene sentido pensar que
# el valor de Shapley de una característica categórica es la suma
# de los valores Shapley de sus categorías
# Calculemos los valores de Shapley para las características
ejemplo=test.iloc[0].copy()
ejemplo[:]=shap_values[1]
ejemplo['workclass'] = (ejemplo['workclass_Other'] +
                        ejemplo['workclass_Private'] +
                        ejemplo['workclass_Self-emp'])
ejemplo['marital-status'] = (ejemplo['marital-status_Never-married'] +
                             ejemplo['marital-status_Separated'] +
                             ejemplo['marital-status_Widowed'])
ejemplo['relationship'] = (ejemplo['relationship_Not-in-family'] +
                           ejemplo['relationship_Other-relative'] +
                           ejemplo['relationship_Own-child'] +
                           ejemplo['relationship_Unmarried'] +
                           ejemplo['relationship_Wife'])
ejemplo['race'] = (ejemplo['race_Asian-Pac-Islander'] +
                   ejemplo['race_Black'] + ejemplo['race_Other'] +
                   ejemplo['race_White'])
ejemplo['sex'] =ejemplo['sex_Male']
ejemplo = ejemplo.drop(['workclass_Other', 'workclass_Private',
        'workclass_Self-emp', 'marital-status_Never-married',
        'marital-status_Separated', 'marital-status_Widowed',
        'relationship_Not-in-family', 'relationship_Other-relative',
        'relationship_Own-child', 'relationship_Unmarried',
        'relationship_Wife', 'race_Asian-Pac-Islander', 'race_Black',
```

```
        'race_Other', 'race_White', 'sex_Male'],axis=0)

ejemplo.plot(kind='barh', width=0.8, color='grey',
             figsize=(8,8), grid=True)
```