Una aproximación matemática a la Inteligencia Artificial Explicable

Aurelio Barrera Vicent

Una aproximación matemática a la Inteligencia Artificial Explicable

Aurelio Barrera Vicent

Memoria presentada como parte de los requisitos para la obtención del título de Grado en Matemáticas por la Universidad de Sevilla.

Tutorizada por

Prof. Tutor Miguel Ángel Gutiérrez Naranjo
Prof. Tutor Eduardo Paluzo Hidalgo

Índice general

English Abstract

Local Interpretable Model-Agnostic Explanations (LIME) is a well-known approach to provide local interpretability to Machine Learning models. One of the key points of these local explanations is the meaning of *locality*. LIME uses an exponential smoothing kernel based on the kernel width value, which defines the width of the local neighborhood.

In this work, we introduce the key aspects of explainable AI (XAI) and we study how distance choice influences these local explanations and how they perform when the dimension of the dataset grows. We explore the choice of kernel width to guarantee a fair performance comparison between Euclidean distance and the Manhattan distance, concluding that when defining a fair kernel width, depending on the distance of choice, they are *close* in performance.

Resumen

Local Interpretable Model-Agnostic Explanations (LIME) es un extendido método para conseguir cierta interpretabilidad en modelos de aprendizaje automático. Uno de los puntos clave de estas explicaciones locales es el significado de local. LIME usa un kernel exponencial que se basa en la elección del valor del kernel width, un argumento que define el tamaño del entorno alrededor de la instancia a explicar.

En este trabajo, introducimos los aspectos clave de la inteligencia artificial explicable (XAI) y estudiamos como la elección de la distancia influye en las explicaciones locales dadas por LIME y como es su rendimiento cuando la dimensionalidad del conjunto de datos crece. Exploramos como debe ser la elección del kernel width para garantizar una comparación justa en términos de rendimiento entre las distancias euclídea y Manhattan, concluyendo que cuando se elige el kernel width adecuado, el desempeño de ambas es parecido.

1 Introducción

El avance de la inteligencia artificial, y en concreto de los modelos obtenidos mediante técnicas de aprendizaje automático en los últimos años, ha protagonizado una de las etapas de mayor inversión en este campo. Esto, junto con la popularización de estos métodos y su aplicación en multitud de ámbitos de la vida cotidiana, ha suscitado gran expectación por los resultados que nos esperan en los años venideros, pero también una gran preocupación por la dificultad de comprender los procedimientos seguidos por el modelo que lo llevan a completar una tarea con éxito (o sin él). Como respuesta a este problema nos encontramos con el surgimiento de la inteligencia artificial explicable (XAI), que procura conseguir más trasparencia para el uso de estos modelos. Entre los métodos existentes hoy día con este objetivo, en este trabajo, nos centramos en LIME, que brinda explicaciones de los resultados obtenidos sobre instancias individuales. El uso de este método se ha extendido durante los últimos años, siendo uno de los más conocidos en este ámbito. En esta memoria nos proponemos estudiar este método, así como analizar como la elección de distancia afecta a las explicaciones finales de LIME.

El trabajo se organiza en 6 capítulos, en el Capítulo 2 presentamos el campo de la inteligencia artificial, centrándonos en el aprendizaje automático, dando las claves para poder entender la diferencia entre ambos conceptos, y qué filosofía hay detrás en la construcción de los modelos de aprendizaje automático. También damos una pequeña introducción sobre la inteligencia artificial explicable.

En el Capítulo 3, describimos las bases de tres de los métodos de aprendizaje automático más conocidos, los árboles de decisión, los random forest y las redes neuronales, dando un ejemplo práctico de la construcción de este último.

En el Capítulo 4, estudiamos uno de los métodos básicos en el ámbito de la explicabilidad, el cálculo de la importancia de atributos, en concreto explicamos el procedimiento para obtener la importancia de atributos en un random forest. Además,

introducimos el método de permutación como solución a ciertas debilidades de este tipo de técnicas. Finalmente, mostramos estos procedimientos en dos ejemplos prácticos.

En el Cápitulo 5, realizamos un estudio en profundidad del LIME, viendo el algoritmo seguido por el método paso por paso y explicando sus debilidades. Citamos algunas de las mejoras propuestas para este método en los últimos años que solucionan problemas bien conocidos del framework de LIME y mostramos un ejemplo práctico de la implementación del método en Python junto con un ejemplo que pone de manifiesto el problema de la elección del kernel width.

Finalmente, en el Cápitulo 6, recopilamos la investigación llevada a cabo durante el curso 2022/2023 en la que estudiamos la influencia de la elección de la distancia en el algoritmo de LIME proponiendo un kernel width dependiente de la distancia. En el capítulo analizamos la diferencia entra las distancias euclídea y Manhattan cuando son usadas en LIME y mostramos los resultados obtenidos en dos experimentos que apoyan la hipótesis de que se pueden obtener rendimientos equivalentes entre distancias mediante la elección adecuada del kernel width.

2 | Preliminares

En este capítulo sentamos las bases sobre inteligencia artificial y aprendizaje automático. En la sección 2.1, definimos el concepto de inteligencia artificial y hacemos un breve repaso sobre algunos momentos clave en este campo. En la sección 2.2 definimos el aprendizaje para una máquina y explicamos las bases necesarias para entender la construcción de modelos de aprendizaje automático. Finalmente, en la sección 2.3 se introduce el concepto de explicabilidad en el ámbito del aprendizaje automático.

2.1 Inteligencia artificial

Desde el nacimiento de la informática, a finales de los años 40, se ha intentado lograr que una máquina sea capaz de realizar acciones y tareas que se asemejen a las que hacen los seres humanos. Esto abarca desde algoritmos capaces de jugar al ajedrez [1], hasta los últimos modelos de procesamiento del lenguaje natural [1] o generación de imágenes [2] de los que todo el mundo habla. Todos estos acercamientos al objetivo de hacer que la máquina *piense* están englobados en el término inteligencia artificial, usado por primera vez durante la conferencia Dartmouth Summer Research Project on Artificial Intelligence por Jonh McCarthy en 1956 [2].

Por la naturaleza ilusionante de este campo, desde sus comienzos, la inteligencia artificial ha vivido momentos de mucho interés e inversión, pero también épocas de desilusión promovida por las grandes expectativas de las que suelen ir acompañados los avances en este ámbito. Durante los años 80 se dieron los primeros éxitos importantes de la inteligencia artificial gracias a lo que se llamaron sistemas expertos. Estos se basaban en la idea de crear un gran conjunto de reglas y algoritmos que permitieran

[1] `https://openai.com/blog/chatgpt`
[2] `https://openai.com/dall-e-2`

a la máquina resolver problemas como encontrar la estructura de moléculas orgánicas dadas las lecturas de un espectrómetro de masas. Este fue el caso del sistema DENDRAL [3], desarrollado por Ed Feigenbaum, acompañado de un equipo de químicos que fueron los encargados de confeccionar las reglas algorítmicas que comprendían el modelo experto. Los buenos resultados de estos sistemas para resolver problemas concretos generaron un boom que se extendió hasta finales de los años 80 cuando se vieron los principales problemas de este enfoque. Los sistemas expertos eran difíciles de mantener, costosos de desarrollar, y se limitaban a tareas concretas y bien definidas, siendo incapaces de tratar con problemas más complejos.

El siguiente gran avance en el ámbito de la inteligencia artificial se consiguió con la aparición de los modelos de *machine learning* o aprendizaje automático, cuyo uso y estudio se extiende hasta el día de hoy.

2.2 Aprendizaje automático

Los términos inteligencia artificial y aprendizaje automático son mencionados juntos frecuentemente, por lo que en ocasiones podría parecer que son lo mismo. Como se comentaba en la sección anterior, la inteligencia artificial es un concepto muy amplio que abarca multitud de enfoques que buscan hacer que un ordenador realice tareas que requieren una capacidad parecida a la de un ser humano. El aprendizaje automático, sin embargo, es un conjunto de estos enfoques caracterizado por usar datos como argumento de entrada , y recibir reglas y relaciones como argumento de salida. En concreto, el aprendizaje automático es la rama de la inteligencia artificial que se centra en desarrollar algoritmos que permiten que una máquina aprenda, es decir, que mejore su rendimiento de forma autónoma mediante la experiencia. De este propósito surge lo que François Chollet en [4, Section 1.1.2] clasifica como un nuevo paradigma para la informática. Mientras que la programación clásica se centra en describir claramente las reglas por las que el ordenador va a realizar una tarea, lo que conlleva un gran conocimiento previo por parte del humano, el aprendizaje automático se basa en la idea de hacer que la máquina deduzca sus propias reglas con el objetivo de realizar una tarea.

Definición 2.1. *[5, p. 2] Decimos que un programa informático aprende de una experiencia E con respecto a un grupo de tareas T y la medida de rendimiento P, si el rendimiento del programa en las tareas T mejora respecto a la medida P mediante la experiencia E.*

Analizamos los componentes de esta definición a continuación.

2.2.1 El programa

Como todo programa informático, un programa de aprendizaje automático, al que a partir de ahora denominaremos modelo, recibe unos argumentos de entrada, *input*, a los cuales se les aplican una serie de transformaciones y procesos de los que se espera un determinado resultado o salida, *output*. En el caso de un modelo de aprendizaje automático, los procesos por los cuales se llega de los argumentos de entrada a la salida no están dados por el ser humano, y es el modelo el que se encarga de encontrar las relaciones y reglas contenidas en los datos.

Los datos de entrada pueden tomar diversas formas dependiendo del tipo de tarea a la que se enfrenta el programa. Principalmente existen tres:

- El aprendizaje supervisado (*supervised learning*), donde se brinda experiencia al modelo mediante datos previamente anotados. Es decir, cada dato de entrada tiene asignada la salida que esperamos recibir, a la que se suele llamar etiqueta (*label*)
- El aprendizaje no supervisado (unsupervised learning), en el cual los datos no están etiquetados
- El aprendizaje por refuerzo (reinforcement learning), donde el modelo es entrenado mediante refuerzos positivos cuando genera procesos que ayudan a mejorar su rendimiento en la tarea.

En este trabajo nos centramos en el estudio de los modelos entrenados mediante aprendizaje supervisado. Existen diversos métodos para construir estos modelos de aprendizaje automático como los árboles de decisión o las redes neuronales, los cuales explicaremos en el capítulo 3.

2.2.2 La experiencia

Como explica la definición 2.1, para que un modelo aprenda hace falta una experiencia.

Definición 2.2. *El proceso por el cual un modelo de aprendizaje automático aprende, es decir, mediante la experiencia mejora su rendimiento, se denomina entrenamiento (training).*

El conjunto de datos suministrado al modelo (*data set*), en el caso de un programa entrenado mediante aprendizaje supervisado, está formado por instancias. Una instancia es el conjunto de argumentos de entrada necesarios para que el modelo de aprendizaje automático funcione. Cada uno de estos argumentos de entrada se denominan atributos o campos.

En este trabajo nos centramos en modelos entrenados mediante datos organizados en tablas, lo que se denominan datos tabulares. En dicha tabla cada fila representa una instancia y cada columna un atributo. En este escenario podemos definir estos conceptos como:

Definición 2.3. *Un conjunto de datos o data set es un subconjunto $X \subset \mathbb{R}^n$. Una instancia es un punto del conjunto de datos $x \in X$ en $\mathbb{R}^n$. Cada componente de dicho punto se denomina atributo.*

La dimensionalidad del conjunto de datos, n, viene determinada por el número de atributos.

Cuando procedemos con el entrenamiento no se usa la totalidad del conjunto de datos, sino que se consideran dos subconjuntos disjuntos de instancias: un conjunto de entrenamiento (*train set*), y un conjunto de test o validación (*test set*). Como sus nombres indican, el primero es usado para el entrenamiento del modelo, mientras que el segundo proporciona un conjunto en el que poner a prueba el modelo. La proporción de estos conjuntos suele variar dependiendo de la extensión de los datos entre 70-30 y 90-10, respectivamente.

La respuesta a la pregunta de por qué se hace esta división de los datos esconde el mayor problema que tiene el aprendizaje automático, el sobreajuste (*overfitting*), que consiste en que el modelo mejora su rendimiento en el conjunto de entrenamiento, sin transmitir esta mejora a instancias nuevas. Esto supone un problema, ya que el objetivo del modelo es que generalice el conocimiento obtenido con el entrenamiento, el sobreajuste es justo lo contrario a este objetivo, el modelo aprende relaciones contenidas en los datos dadas únicamente en la muestra, pero que no representan utilidad alguna para la resolución del problema en la "vida real", para nuevas instancias. Para poder analizar si el modelo se está sobre ajustando es por lo que el conjunto de datos se divide, y es que cuando un modelo está sobre ajustado al conjunto de entrenamien-

to, se aprecia como la mejora del rendimiento no se traduce a las instancias contenidas en el conjunto de test, que recordemos son datos que el modelo no ha usado durante el entrenamiento.

Los datos que se proporcionan a la hora de entrenar el modelo son realmente importantes para conseguir buenos resultados con este tipo de técnicas. Una gran parte del trabajo detrás de los modelos de aprendizaje automático se basa en conseguir un conjunto de entrenamiento fiable, imparcial y representativo de la realidad en la que se quieren aplicar los conocimientos obtenidos por el modelo.

2.2.3 La tarea

El aprendizaje automático y, en concreto, los modelos basados en el aprendizaje supervisado han conseguido con éxito afrontar tareas en multitud de áreas como la clasificación de imágenes [6, 7], sistemas de recomendación [8] usados por Google [9] o Twitter [10], detección de cáncer [11, 12], entre otros. En parte, el éxito reciente de estos modelos se puede explicar gracias al avance de la tecnología, que permite hacer los procesos de entrenamiento mucho más rápidos y almacenar una gran cantidad de datos, necesarios para obtener resultados favorables en tareas complejas que antes eran inabarcables por una máquina.

Estas tareas afrontadas por modelos de aprendizaje supervisado se pueden dividir en problemas de clasificación, donde el modelo intenta predecir la categoría asociada a una instancia entre un número finito de posibilidades, en este marco se encuentran problemas como la detección del cáncer [12], donde el modelo debe seleccionar entre una respuesta positiva o negativa para cada caso; y problemas de regresión, donde la salida del modelo toma valores continuos y el objetivo es encontrar el ajuste funcional que consiga aproximar *mejor* la salida deseada.

Para poder mejorar el desempeño del modelo en una tarea se necesita definir qué hay que mejorar, a esto lo denominamos función de error o (*loss function*), que se encarga de establecer una medida entre el resultado obtenido y el resultado esperado. Después, el modelo, mediante diferentes reglas o métodos de optimización, es capaz de minimizar el valor de la función, gracias a lo cual se consigue el aprendizaje. Esta función de error no tiene una definición única, y se pueden agrupar en las usadas en problemas de regresión y las usadas en problemas de clasificación[3]. Algunas de las

[3]En lo que sigue denotaremos con n el número de instancias de entrenamiento, $y(i)$ a la etiqueta para la instancia i, $y'(i)$ a la predicción del modelo sobre la instancia i.

más conocidas son [13]:

Problemas de clasificación:

- Cross-Entropy: $-\frac{1}{n}\sum_{i=1}^{n} y(i)\log_2(y'(i))$ Usada en casos donde el modelo devuelve la probabilidad de pertenencia a una categoría y basada en la definición de entropía que vemos con detalle en la sección 3.1.1

Problemas de regresión:

- Error cuadrático medio: $\frac{1}{n}\sum_{i=1}^{n}(y'(i)-y(i))^2$
- Error absoluto medio: $\frac{1}{n}\sum_{i=1}^{n}|y'(i)-y(i)|$
- Error medio: $\frac{1}{n}\sum_{i=1}^{n}(y'(i)-y(i))$

2.2.4 El rendimiento

Una vez el modelo ha sido entrenado, se necesita medir su capacidad para realizar la tarea. Para esta evaluación se usan distintas medidas según el problema que se está tratando. En problemas de regresión dicha medida suele coincidir con las funciones de error arriba comentadas, en problemas de clasificación, es común usar el porcentaje de error, donde para evaluar el rendimiento del modelo bastaría ver qué porcentaje de instancias se clasificaron de forma correcta.

Para la evaluación del rendimiento del modelo usamos el conjunto de test. Esto se debe a que el modelo podría estar sobre ajustándose a los datos del conjunto de entrenamiento, por lo que los resultados del rendimiento del modelo en estas instancias serían más altos de lo que el modelo realmente será capaz de hacer en un entorno real. Para comprobar si el modelo está sobre ajustado a los datos de entrenamiento, basta con comparar los valores de la medida de rendimiento en el conjunto de entrenamiento con la obtenida en el conjunto test, si la primera es más alta que la segunda el modelo está sobre ajustado.

2.3 Explicabilidad

El avance del campo del aprendizaje automático ha venido acompañado de un gran aumento en la complejidad de los modelos. Esto, junto con la naturaleza autónoma de estos, ha separado el entendimiento y la lógica humanas del procedimiento seguido por la máquina para llegar al resultado, lo que provoca que el conocimiento adquirido por el modelo no sea mostrado en términos entendibles para los seres humanos, a lo que nos solemos referir como *black box models*. El problema que se plantea es que no podemos justificar la predicción que nos brinda el sistema por la complejidad de este.

2.3.1 Motivación

En los últimos años, ha habido un gran progreso en el ámbito del aprendizaje automático, haciendo que ahora modelos de este tipo participen en numerosos ámbitos de la vida diaria, desde la recomendación de contenidos del algoritmo de Google [9] a los procesos por los cuales se te concede o no una hipoteca. La relevancia que tienen estos modelos en la actualidad hace que debamos preguntarnos, ¿podemos fiarnos de sus predicciones? Hasta ahora la única medida que tenemos de lo confiable que es un modelo es su rendimiento en el conjunto de test en términos de precisión, pero lo cierto es que esto es insuficiente para la mayoría de problemas de la vida real [14]. Uno de los cometidos de la explicabilidad es el poder juzgar mejor la utilidad de un modelo en la práctica.

Por otro lado, el gran progreso en este campo ha estado marcado desde sus comienzos por un claro empuje empírico. La complejidad de los modelos y su buen rendimiento en la práctica han fomentado un ambiente de grandes resultados, pero en ocasiones no suficientemente respaldados por una explicación teórica. Esta necesidad se ve reflejada en la General Data Protection Regulation (GDPR) de la Unión Europea, que regulaba en 2016 modelos de este tipo que manejaran datos personales y afectaran en gran medida a los usuarios: *"In any case, such processing should be subject to suitable safeguards, which should include specific information to the data subject and the right to obtain human intervention, to express his or her point of view, to obtain an* ***explanation*** *of the decision reached after such assessment and to challenge the decision."* [15, Recital 71]. Esta regulación pone de manifiesto la necesidad de desarrollar una teoría robusta que explique los resultados que nos brindan estos sistemas.

Por último, una gran utilidad de la explicabilidad, es la de extraer los conocimien-

tos aprendidos por el modelo durante el entrenamiento, lo que la convierte en un aspecto clave a tener en cuenta para usar modelos de aprendizaje automático como motor de avance en el ámbito científico.

Este conocimiento está formado por reglas y relaciones que en [16] se denominan interpretaciones. Las interpretaciones recogen el conocimiento acumulado por el modelo después de ser entrenado y son muy útiles para, como comentábamos, evaluar el modelo, entender mejor el resultado o para ayudar en el ámbito científico, extrayendo conocimiento provechoso para el avance en numerosos ámbitos como la medicina o la biología. El problema es que estas interpretaciones no se expresan en términos manejables por los seres humanos, el propósito de la explicabilidad es, por tanto, presentar estas interpretaciones de forma clara y entendible.

2.3.2 Definiciones

En [16] se define la interpretabilidad [4] como la extracción de conocimiento relevante del modelo sobre las relaciones contenidas en los datos o aprendidas por este (lo que antes llamábamos interpretaciones).

Podemos preocuparnos de la explicabilidad durante el diseño del modelo, lo que se conoce como interpretabilidad intrínseca (*model-based interpretability*), en este caso acotamos la complejidad del modelo para que sea suficientemente sencillo de comprender por el humano, generalmente usando regresiones lineales o árboles de decisión no demasiado profundos. Los inconvenientes de esta técnica son claros, estamos consiguiendo extraer las interpretaciones del modelo afectando negativamente (dependiendo del problema) al rendimiento de este, ya que no todos los problemas que enfrenta el aprendizaje automático son susceptibles de ser resueltos por modelos sencillos.

La otra opción que tenemos para conseguir la interpretabilidad del modelo se basa en aplicar métodos que analizan el modelo después del entrenamiento (*post-hoc interpretability*). De esta forma preservamos el buen rendimiento del modelo complejo, pero obtener buenas explicaciones se vuelve una tarea mucho más complicada.

Como comenta Christoph Molnar en [17], en estas dos técnicas encontramos la diferencia de perspectiva entre la estadística y el aprendizaje automático. Mientras

[4] En este trabajo no distinguimos entre interpretabilidad y explicabilidad, sin embargo, hay autores que consideran que estos términos no son intercambiables.

que en estadística nos centramos en el estudio de la distribución de los datos para poder sacar conclusiones de modelos de por sí interpretables muy respaldados por una teoría sólida, los "machine learning practitioners" desarrollan modelos complejos en los que se hace necesaria una última capa de explicabilidad con la que construir cierta confianza en el modelo.

Dentro de los métodos *post-hoc* podemos distinguir dos tipos, los denominados *model-agnostic* y los *model-specific*. Los métodos *model-agnostic* son aquellos métodos que no dependen de la naturaleza del modelo en el cual se aplican, su gran ventaja es la versatilidad. Estos métodos se aplican desde redes neuronales hasta random forest. Por otro lado, los *model-specific* usan información sobre el diseño del modelo para poder extraer las interpretaciones aprendidas por este.

3 Métodos del aprendizaje automático

En este capítulo presentamos de manera concisa algunos de los métodos más usados en el ámbito del aprendizaje automático, en la sección 3.1 damos las bases sobre la teoría de la información que soportan los algoritmos de generación de árboles de decisión como ID3, que estudiamos más en profundidad. En la sección 3.2 presentamos como se construye un random forest y en la sección 3.3 mostramos los conceptos básicos en cuanto a redes neuronales acompañados de un ejemplo práctico que pone en práctica los conceptos mostrados tanto en esta sección como los presentados en la sección 2.2.

3.1 Árboles de decisión

3.1.1 Nociones básicas sobre Teoría de la Información

La teoría de la información es un campo de las matemáticas que surge a finales de los años 40 gracias a los trabajos de Claude E. Shannnon [18], donde se pregunta cuanta información contiene un mensaje. Shannon llega a la conclusión de que la información que aporta un evento se puede medir basándonos en la probabilidad de ocurrencia de este. Con esta idea se define el concepto de entropía.

Definición 3.1. *Sea S un conjunto y p_x el ratio de ocurrencia de un elemento $x \in S$. Entonces la entropía de S, H(S), viene dada por la expresión*

$$H(S) = \sum_{x \in \Omega} -p_x \log_2(p_x)$$

Es fácil ver que si todos los elementos de S pertenecen a la misma clase, entonces la entropía del conjunto es 0. La idea intuitiva que hay detrás de la entropía es la de medir la impureza o heterogeneidad de un conjunto.[1]

Existen distintas alternativas a la entropía como medida de la heterogeneidad de un conjunto dado. Podemos destacar el uso del coeficiente de Gini como uno de los más extendidos.

Definición 3.2. *Sea S un conjunto y p_x el ratio de ocurrencia de un elemento $x \in S$. Entonces el coeficiente de Gini de S, Gini(S), viene dada por la expresión*

$$Gini(S) = \sum_{x \in \Omega} p_x(1 - p_x)$$

3.1.2 Generación de un árbol mínimo con ID3

La generación de un árbol de decisión es un método de aprendizaje automático para problemas de clasificación, que ha demostrado su utilidad en numerosos problemas en ámbitos como la medicina. Entre sus ventajas encontramos que es un método intrínsecamente interpretable. Existen numerosos algoritmos para generar un árbol de decisión, nosotros nos centramos en ID3, que nos da un árbol de decisión mínimo, eligiendo en cada paso la opción de la que más información se puede extraer. En este apartado nos centramos únicamente en problemas de clasificación, en los cuales se puede aplicar el algoritmo ID3.

La idea que tenemos con los árboles de decisión es usar las categorías de los atributos para dividir las instancias del conjunto de entrenamiento. Al final del árbol queremos que las instancias queden clasificadas por el valor que toman en la etiqueta.

El algoritmo ID3 construye el árbol de arriba abajo y se centra en la elección del atributo que aporta más información al dividir las instancias del conjunto de entrenamiento. Para tomar esta decisión, el algoritmo usa información estadística de la muestra, la ganancia de información, definida como la reducción de la entropía al hacer la clasificación mediante ese atributo. Podemos definir entonces:

Definición 3.3. *Sea S el conjunto de instancias inicial, A un atributo y n el número de*

[1] En cálculos que tienen que ver con la entropía consideramos $log_2(0) \cdot 0 = 0$

categorías de A. Entonces la ganancia de información G(S,A) viene dada por la expresión:

$$G(S, A) = H(S) - \sum_{i=1}^{n} \frac{|S_i|}{|S|} H(S_i)$$

Donde S_i denota el conjunto de instancias que comparten el valor i en el atributo A y $|S_i|$ su cardinalidad [5].

Con este criterio, ID3 comprueba qué atributo es el mejor en cada paso, sin volver en ningún momento atrás para revisar la decisión, lo que podría provocar la convergencia del método a una solución óptima local, donde la optimalidad está definida como el menor árbol posible, siguiendo el principio de la Navaja de Ockham [2]. Al usar todas las instancias en cada toma de decisión, el algoritmo es bastante robusto ante errores en instancias individuales.

3.2 Random Forest

En esta sección damos una pequeña introducción de cómo funciona y cómo se construye un random forest.

Un random forest es un método predictivo basado en la creación de múltiples árboles de decisión mediante los cuales se llega a la predicción final, a lo que se suele llamar un *ensemble method*. Para ello, en problemas de clasificación, el modelo devuelve la clase con más "votos" entre todos los árboles. De esta forma, el random forest consigue lidiar con uno de los problemas más importantes de los árboles de decisión, el sobreajuste, pero para que esto sea posible necesitamos obtener a partir del mismo conjunto de datos de entrenamiento distintos árboles de decisión. Para ello existe una técnica llamada *bagging*.

El *bagging* es el término usado para referirse a la aplicación del método *bootstrap* en modelos de aprendizaje automático. En nuestro caso esto consiste en la generación de subconjuntos de entrenamiento a partir del conjunto original, estos subconjuntos no son necesariamente disjuntos. Los árboles de decisión se entrenan en estos subconjuntos. A este método se le suele añadir otro paso debido a la naturaleza de los algoritmos de creación de estos árboles para evitar que salgan demasiado parecidos. Lo que se hace es elegir también un subconjunto aleatorio de atributos con los que

[2] `https://es.wikipedia.org/wiki/Navaja_de_Ockham`

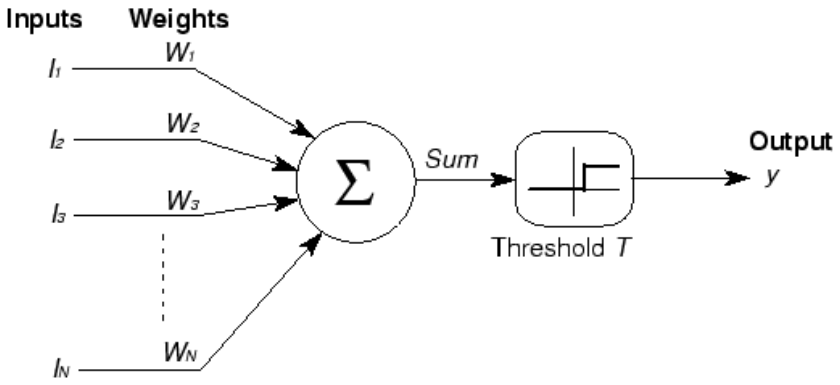

Figura 3.1: McCulloch-Pitts neuron. Credits:[21]

entrenar el árbol, generalmente si tenemos p atributos se escogen $\sqrt{p}$ atributos para el entrenamiento de un árbol, aunque el número óptimo dependerá de cada problema [19].

3.3 Redes Neuronales

Si bien, la existencia de las redes neuronales se remonta al surgimiento de la inteligencia artificial en los años 50, no fue hasta los años 90 cuando se extendió su popularidad y mostraron su verdadero potencial. Los primeros ensayos que desembocaron en las redes neuronales con las que trabajamos hoy en día se dieron como intentos de modelar el comportamiento de las neuronas del cerebro humano. Con esta idea, Warren S. McCulloch y Walter Pitts [20] presentaron en 1943 lo que se conoce como McCulloch-Pitts neuron, un modelo de neurona que funciona recibiendo ciertos inputs en forma de suma ponderada y que según cierto umbral esta se activa o no, devolviendo un 1 o un 0.

Este modelo sería la base sobre la cual, en 1958, Frank Rosenblatt presentaría el Mark I Perceptron [22], la primera implementación del modelo propuesto por McCulloch y Pitts. El Mark I perceptron estaba diseñado para ir actualizando sus pesos o parametros para alcanzar el mínimo error en los outputs del sistema. Este error está dado por una función denominada Loss Function.

En este caso se usaba una única capa de neuronas, lo limitaba al Mark I perceptron

a problemas de clasificación donde podían separarse las clases mediante una expresión lineal. Los siguientes intentos se enfocarían, por lo tanto, en remediar este problema, para lo que se implementaron sistemas con múltiples capas o layers conectadas entre sí. Este enfoque es el que da nombre a las redes neuronales y es el que se sigue usando hoy en día.

A pesar de estos avances, en la década de los 60 se empiezan a vislumbrar las limitaciones de este tipo de modelos, principalmente los tiempos de ejecución de estas redes, que eran demasiado grandes en el hardware de la época. Este y otros problemas hacen que se abandonara el desarrollo de esta técnica hasta principios de los años 90, cuando la implementación de los algoritmos de gradiente, junto con una técnica denominada backpropagation conforman las redes neuronales como las conocemos hoy en día.

Las técnicas de gradiente no son desconocidas en el campo de las matemáticas. Se usan para encontrar el mínimo de una función, y se basan en el uso de la derivada para establecer una dirección de descenso que apunta hacia el mínimo que buscamos, o en ciertas ocasiones un mínimo local. Este método es de gran utilidad en las redes neuronales, donde la acción de cada capa puede ser descrita mediante una función diferenciable en casi todo punto:

$$f(input \cdot w + b) \tag{3.1}$$

donde f representa la función de activación, $input$ el vector de los valores de entrada, w los pesos de la neurona y b el peso independiente o bias.

Tenemos, por tanto, que la acción de la red neuronal puede escribirse como una función F, composición de expresiones como 3.1, y queremos encontrar el mínimo de la función $L(y, F)$, donde L es la función de error e y la predicción que se busca que devuelva el modelo. Dada entonces $L(y, F)$ diferenciable en casi todo punto, podemos usar algoritmos de gradiente para encontrar dicho mínimo, para lo cual necesitaremos calcular el gradiente.

Para el cálculo de la derivada se hace uso de la regla de la cadena. Al conocer los valores del gradiente en las funciones 3.1, el cálculo del gradiente de F es mucho más sencillo. Además, haciendo uso de esta técnica podemos identificar la contribución de cada peso en el valor final de la función de error recorriendo hacia atrás la red mediante estos gradientes. Esta técnica es la que se conoce como backpropagation [23]. El resultado es una red de neuronas que va actualizando sus pesos para alcanzar el mínimo buscado.

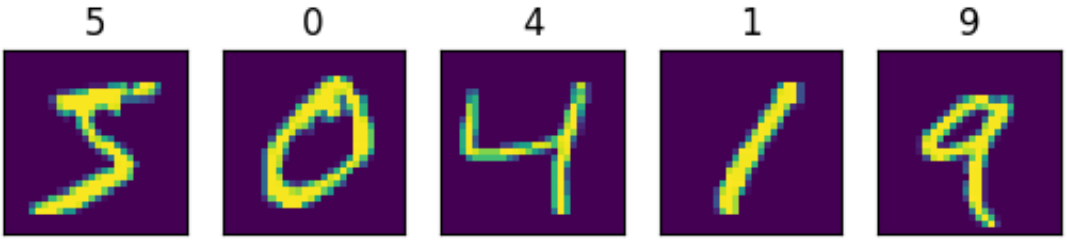

Figura 3.2: Primeras 5 instancias del data set MNIST acompañadas de sus etiquetas.

3.3.1 Un ejemplo práctico

Para ilustrar los conceptos vistos en las secciones anteriores procedemos a realizar un ejercicio práctico. Para ello usaremos un ejemplo clásico de aplicación del aprendizaje automático, el *data set* MNIST [24]. Este conjunto de datos está formado por 60.000 instancias anotadas de imágenes de dígitos escritos a mano. El objetivo es que el modelo aprenda a reconocer qué dígito es el representado en cada imagen, nos encontramos, por tanto, ante un problema de clasificación. Los mejores resultados hasta la fecha para la realización de esta tarea han sido obtenidos mediante la utilización de redes neuronales, con un porcentaje de error del 0,17 %. Los próximos pasos están basados en los resultados obtenidos en [25].

Primero visualizamos cómo son las primeras instancias del conjunto de datos[3]. Como podemos ver en la figura 3.2, las instancias estan formadas por imagenes de 28x28 píxeles y cada una contiene un dígito. Podemos ver también la etiqueta asociada a cada instancia encima de cada imagen.

Para este ejemplo usaremos una red neuronal definida en Python con 2 capas de 5 y 10 neuronas respectivamente (a parte de una capa para la ingestión de los datos). La primera con la sigmoide como función de activación y la segunda con la función *softmax*, para devolver las probabilidades de pertenencia a cada clase. El modelo usará como función de error la *categorical cross entropy* y, como métrica de rendimiento, el ratio de imágenes bien clasificadas. Podemos ver en la tabla 3.1 los valores obtenidos tanto en el conjunto de entrenamiento como en el conjunto de test al final del entrenamiento del modelo.

[3] `https://github.com/skorch-dev/skorch/blob/master/notebooks/MNIST.ipynb`

	Función de error	Métrica de rendimiento
Conjunto de entrenamiento	0.8525	0.7542
Conjunto de test	0.8133	0.7737

Cuadro 3.1: Valores obtenidos para las funciones de error y rendimiento en los conjuntos de entrenamiento y test.

Podemos observar que obtenemos un ratio de éxito del 75 % en el conjunto de test. Con esta técnica podríamos observar, en caso de que hubiera *overfitting* como la métrica del rendimiento en el conjunto de entrenamiento sería mayor que la obtenida en el conjunto de test.

4 | Importancia de atributos

El método más usado para intentar arrojar luz a las interpretaciones del modelo es sin duda el cálculo de la importancia de atributos. En numerosas ocasiones este método es un primer paso de cara a entender el modelo. En este capítulo estudiamos este método aplicado a árboles de decisión y random forest en la sección 4.1. Además, vemos una implementación mediante la librería sklearn en python para random forest en la sección 4.2.

4.1 Importancia de atributos en árboles de decisión

Como primer paso lo que hacemos es calcular la importancia de los nodos. Sea n_0 el nodo a estudiar, y sean $n_1, ..., n_r$ las hojas de este. Entonces dados, el número de instancias totales con las que se entrena el árbol N, el número de instancias que llegan a n_0, N_0 y su medida de impureza (dada por Gini o entropía) C_0, el número de instancias que llegan a cada hoja N_i y sus respectivas impurezas C_i con $i = 1, ..., r$, tenemos que la importancia de n viene dada mediante la expresión:

$$I_{n_0} = \frac{N_0}{N} C_0 - \sum_{i=1}^{r} \frac{N_i}{N} C_i$$

Es fácil ver que a mayor sea la impureza que reciba el nodo y menor sea la impureza de sus hojas, mayor importancia tendrá este, ponderando estos valores con la proporción de instancias sobre las que actúa el nodo.

Podemos entonces calcular ahora la importancia de un atributo en un árbol concreto:

$$I_A = \frac{\sum_{n \in S} I_n}{\sum_{n \in T} I_n}$$

Donde A es el atributo del que queremos calcular la importancia en el árbol, S el conjunto de nodos donde A participa y T el conjunto de nodos total del árbol. Decimos que A participa en un nodo si es el atributo que se toma para hacer la clasificación en ese paso. Vale la pena observar que $I_A \in [0, 1]$. Cuanto más cercano a 1 más importancia acumula el atributo A en el árbol. Finalmente, la importancia del atributo A viene dada por la media de las importancias obtenidas en cada árbol del random forest.

Pero este método para calcular la importancia de atributos tiene deficiencias, como se muestra en [26], tiende a dar mucha importancia a atributos con un gran número de clases o atributos continuos. Esto se debe a que la ganancia de información de este tipo de atributos suele ser alta, ya que en muchos casos nos permiten separar instancias, es decir, dos instancias generalmente no van a compartir la misma clase de ese atributo, lo que te permite clasificarlas. Veremos este efecto en el ejemplo práctico de la sección 4.2. Para evitar este problema se aplica el denominado método de permutación.

El método de permutación consiste en medir la influencia que tiene en el error del modelo permutar los valores que toman las instancias en un atributo fijado. Si el error después de este proceso aumenta, significa que el atributo era importante. La intuición detrás de esta idea es que si los valores de un atributo no son realmente importantes para la predicción, entonces cambiar este valor por otro no debería alterar demasiado el resultado del modelo, preservando de esta manera la precisión de este.

Algorithm 1 Permutation Feature Importance. Christopher Molnar [17]

Input: modelo $\hat{f}$, matriz de datos X, target y, error $L(y, \hat{f})$
Output: vector de importancia de atributos
1. Calculamos el error original $e = L(y, \hat{f}(X))$
2.
for j **in** atributos **do**
Crear matriz X'_j permutando los valores de la columna X_j
Calculamos el error sobre los nuevos datos $e' = L(y, \hat{f}(X'_j))$
Calculamos la importancia del atributo como el coeficiente $I_j = e'/e$
end for
3. Ordenamos los atributos de mayor a menor importancia

4.2 Un ejemplo práctico

4.2.1 Entrenamiento de un árbol en sklearn y cálculo de importancias

La naturaleza interpretable de los árboles de decisión se pierde en los random forest. Por ello se hace necesaria la aplicación de alguna técnica de interpretabilidad para construir confianza en el modelo e intentar entender los procesos que llevan a sus predicciones. En esta sección estudiamos la integración en sklearn [27] del cálculo de importancia de atributos para random forest.

Aparte de ID3, para la generación de los árboles de decisión, sklearn usa CART [28], que también sirve en problemas de regresión. Además usa el coeficiente de Gini, aunque también podemos decidir usar entropía.

Procedemos a ver cómo se aplica este método en un ejemplo concreto usando sklearn. Para ello vamos a crear un data set artificial para un problema de clasificación con dos clases.

```
from sklearn.datasets import make_classification
import random

#Generamos un dateset random con 3 atributos importantes y 1 redundantes
X, y = make_classification(n_samples=1000, n_features=4, n_informative = 3,
                           n_redundant = 1, n_classes=2, flip_y=0.001,
                           class_sep= 3, random_state=20)
```

En este caso contamos con cuatro atributos, tres de los cuales son relevantes para la resolución del problema y el otro es redundante. Vemos si es el método de importancia de atributos para un único árbol, es capaz de detectar este hecho.

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.90,
                                                    random_state=4)
classifier = DecisionTreeClassifier()
classifier.fit(X_train, y_train);
```

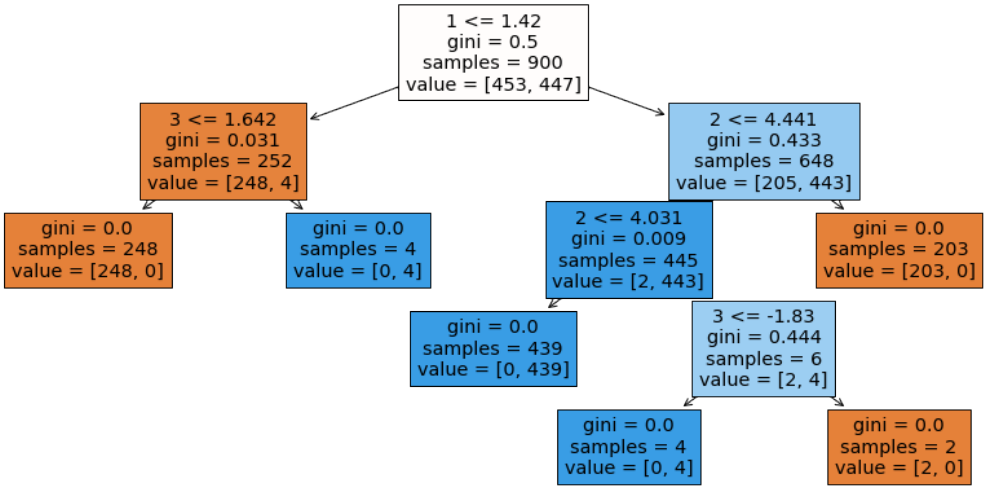

Figura 4.1: Árbol obtenido en el entrenamiento

Código 1: Entrenamiento de un árbol de decisión con sklearn

Para ello entrenamos un árbol de decisión en nuestro conjunto de entrenamiento, como se ve en el código 1, lo que nos da como resultado el árbol de la figura 4.1, en la que podemos ver en cada nodo en orden descendente, el atributo que participa, el valor del coeficiente de Gini, el número de instancias que llegan al nodo y por último cuáles de estas pertenecen a cada clase.

Con la información que obtenemos de cada nodo en la figura 4.1 podemos realizar los cálculos introducidos anteriormente a mano, y así comprobar que coincide con lo que obtenemos mediante la implementación de sklearn. Obteniendo el vector de importancias, $(0.0, 0.3596, 0.6169, 0.0234)$ mediante la implementación de sklearn y el vector $(0.0, 0.3591, 0.6176, 0.0232)$ mediante nuestros cálculos. La pequeña diferencia que observamos se debe a la aproximación del coeficiente de Gini.

4.2.2 Método de permutación

Ahora procedemos a aplicar el método de permutación en otro ejemplo práctico como se ve en [26]. Primero volvemos a crear un data set artificial para un problema de clasificación con 2 clases y 15 atributos, de los cuales 5 son informativos y los demás redundantes. De nuevo entrenamos un random forest en este data set y computamos

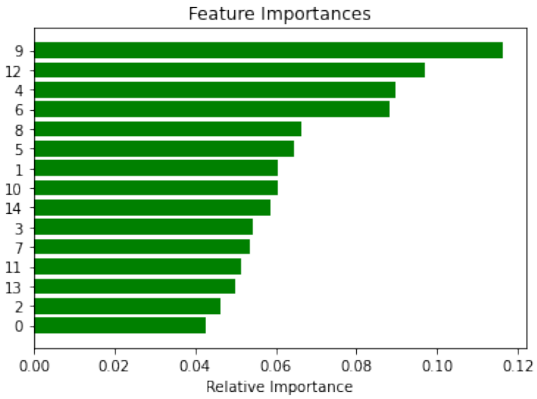

Figura 4.2: Importancia de atributos en el Random Forest.

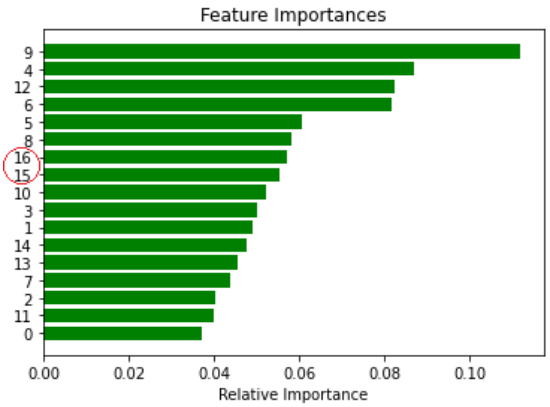

Figura 4.3: Importancia de atributos antes de la aplicación del método de permutación.

la importancia de atributos obteniendo la figura 4.2.

Después añadimos dos atributos nuevos al data set. Uno estará conformado por valores continuos asignados aleatoriamente mediante una distribución uniforme en el rango [0, 100], en el experimento aparece como atributo 15. El otro es un atributo que toma valores discretos y asigna un valor diferente a cada instancia, en el experimento aparece como atributo 16.

Volvemos a entrenar el modelo y calculamos las importancias obteniendo la figura 4.3. Donde se aprecia como, a pesar de que los atributos 15 y 16 no aportan ningún tipo de información al modelo, ya que fueron generados aleatoriamente, consiguen clasificar por delante de muchos atributos, que a pesar de poder ser redundantes, sí que tienen que ver con el output deseado.

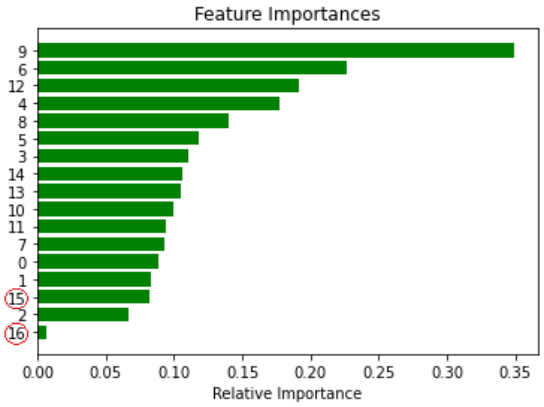

Figura 4.4: Importancia de atributos después de la aplicación del método de permutación.

Esto es un problema, ya que el método estará *premiando* atributos con un gran número de clases, lo que no es una medida real de la importancia del atributo. Para solucionar este problema contamos con el método de permutación descrito en la sección 4.1 . Aplicando este método al ejemplo práctico utilizado anteriormente, obtenemos la figura 4.4 donde vemos como el método es capaz de identificar los atributos que no aportaban verdadera información de cara a la resolución del problema.

5 LIME

Uno de los métodos que ha ganado más popularidad recientemente en el ámbito de la interpretabilidad es LIME (Local Interpretable Model-Agnostic Explanations), presentado por Marco Tullio Riberio, Sameer Singh y Carlos Guestring en [29]. En este capítulo, nos centramos en el estudio de este método, así como sus variantes, para ello introducimos el problema y el método en la sección 5.1. En la sección 5.2 mostramos el algoritmo de LIME en profundidad, para en la sección 5.3 presentar las principales debilidades de este método junto con la mención de las mejoras propuestas en los últimos años. Finalmente, en la sección 5.4 mostramos la implementación del método en Python junto con un ejemplo que pone de manifiesto el problema de la elección del kernel width.

5.1 Introducción

En los últimos años, los modelos obtenidos mediante las técnicas de aprendizaje automático han conseguido un gran avance y, actualmente, son capaces de resolver problemas de la vida cotidiana que eran intratables para una máquina hace tan solo unos años. Esta evolución ha sido motivada por el desarrollo tanto del hardware y las arquitecturas de estos modelos, como de algunos resultados teóricos. Uno de los principales problemas que presentan es que el conocimiento obtenido no se expresa en formas entendibles para los humanos, lo que hace que estos modelos, en muchas ocasiones, se consideren *black boxes*. Esta falta de explicación de las decisiones es un problema para el uso de estas tecnologías en ámbitos como la medicina o las ciencias sociales, entre muchos otros. En este sentido, ciertas entidades han empezado a confeccionar el marco legal para la aplicación de la inteligencia artificial en nuestro día a día. En esta línea podemos citar el *White Paper on Artificial Intelligence (AI)* [30] de la Unión Europea o el *Four Principles of Explainable Artificial Intelligence* [31] por el

National Institute of Standards and Technology of the US Department of Commerce.

La incompatibilidad entre el desarrollo de la IA y su posible uso en el ámbito social ha provocado que la comunidad científica haya desarrollado en los últimos años un nuevo área de investigación denominado Inteligencia Artificial Explicable (XAI). Aunque todavía no hay una definición universal de lo que trata este campo, de acuerdo con la literatura, XAI debe contener una serie de técnicas para proporcionar unas explicaciones que sean claras, entendibles, confiables e interpretables de las decisiones, predicciones y razonamientos detrás de los procesos seguidos por los modelos de inteligencia artificial. XAI ha sido aplicada con éxito en multitud de áreas como la medicina [32], las finanzas [33]y la conducción autónoma [34], por nombrar algunos.

Desde un punto de vista técnico, hay muchos criterios por los que guiarse a la hora de clasificar las técnicas de XAI [17]: Model agnostic vs. model specific; intrinsic vs. post hoc; etc. Uno de ellos es considerar cuándo la explicación dada es *local* o *global* [35]. Por un lado, las explicaciones globales intentan entender el modelo inspeccionando sus parámetros y arquitectura para poder así dar explicaciones sobre el comportamiento general de este. Por otro lado, las explicaciones locales intentan mostrar las razones que llevan a una predicción individual. Normalmente este último enfoque se centra en medir la contribución de cada atributo en la predicción final.

LIME es un método que no depende del modelo al que es aplicado (model-agnostic) propuesto por Marco Tulio Riberio *et al.* [29] para conseguir interpretabilidad para predicciones de instancias individuales. Cada explicación consiste en una lista de pares atributo-valor que determina qué atributos contribuyen más a la predicción final del modelo junto a un valor numérico que mide esta contribución.

La idea geométrica detrás de LIME se puede resumir como sigue: las predicciones del modelo complejo se pueden ver como una hipersuperficie en $\mathbb{R}^n$ donde n es el número de atributos contenido en el conjunto de datos. LIME intenta aproximar la predicción sobre una instancia concreta construyendo un modelo más simple, generalmente lineal, que aproxime al modelo original en un entorno de la instancia a explicar. Una solución teórica a este problema sería tomar el hiperplano tangente a dicha superficie en el punto de interés, pero generalmente, esta tangente no existe, ya que es bien conocido que las hipersuperficies dadas por las predicciones de estos modelos no son necesariamente continuas, por lo que perdemos la noción de derivabilidad y por ello la tangente que buscábamos. LIME resuelve este problema aproximando dicha tangente mediante una regresión lineal, siendo los coeficientes de dicha aproximación lo que tomamos como explicación.

5.2 El algoritmo

El algoritmo de LIME intenta maximizar la adherencia local de las explicaciones aproximando el modelo complejo a explicar mediante un modelo más simple e intrínsecamente interpretable. En LIME, este segundo es, por defecto, el modelo lineal Ridge [36]. El algoritmo recibe como entrada un modelo complejo f y una instancia a explicar x. Para generar el data set para entrenar el modelo Ridge, primero obtenemos una *versión interpretable* del data set original. Para ello, se discretiza , por defecto en cuartiles y se genera un conjunto Z de instancias aleatorias al rededor de x. A estas instancias les es asignado un peso dependiendo de la distancia que separa las representaciones binarias de $z \in Z$ y de x, generalmente la medida de proximidad es definida por una función kernel:

$$\Pi_x(z) = exp(-D(x,z)^2/\tau^2) \tag{5.1}$$

donde τ es el kernel width y D la distancia elegida. Obsérvese que el kernel width es el valor que define la localidad del modelo. Después, se calcula la representación binaria de cada instancia, donde 1 codifica la coincidencia con el cuartil de x y 0 el caso contrario. Este data set binario se usa para entrenar el modelo Ridge, donde las etiquetas a cada instancia son generadas usando el modelo complejo sobre la versión continua de dicho registro.

Finalmente, el modelo Ridge es entrenado en este data set de instancias aleatorias. Ridge está basado en los mínimos cuadrados, pero impone una penalización sobre el tamaño de los coeficientes. Intenta encontrar los coeficientes w que minimizan la expresión

$$min_w |Xw - y|_2^2 + \alpha |w|_2^2 \tag{5.2}$$

donde α es un parámetro de complejidad.

Los coeficientes de dicho modelo lineal se usan como explicación de la influencia de cada atributo en la predicción final hecha sobre x.

Podemos ver una explicación más detallada del algoritmo en el algoritmo 2.

5.3 Las debilidades de LIME

Aunque LIME se ha usado recientemente de manera extendida para dar cierto grado de interpretabilidad a las predicciones de los modelos de machine learning, el

framework cuenta con algunas debilidades.

El primer problema al que se enfrenta el uso de LIME es la elección del entorno correcto para la instancia que queremos explicar. Este hecho está principalmente manejado por la elección del valor del kernel width. Lo que es un problema, ya que es una elección arbitraria que además es muy influyente en la explicación que obtenemos de LIME, es decir, para distintos valores del kernel width es fácil obtener distintas explicaciones para una misma instancia, como se ve en [17, Section 9.2.1].

El otro problema que encontramos es la inestabilidad de las explicaciones debida principalmente al proceso de muestreo aleatorio con el que se entrena el modelo local. Con este método no es extraño encontrarse casos en los que cuando ejecutamos varias veces el algoritmo puede devolvernos explicaciones distintas para una misma instancia.

En este sentido podemos encontrar numerosos estudios de LIME que se centran en proponer diferentes mejoras al algoritmo original. Podemos citar DLIME [37], donde se usan técnicas de clustering para agrupar el training data; ALIME [38], donde se usan un autoencoder como función para decidir los pesos usados por el modelo local, o LEDSNA [39] donde los autores usan Support Vector Regression.

Recientemente, una versión de LIME llamada OptiLIME [40] justifica que existe un balance entre la localidad o adherencia del modelo simple, es decir, cuanto se parece la predicción del modelo simple a la del modelo complejo en el entorno definido, y la estabilidad, que mide cuanto varían las explicaciones de LIME de una ejecución a otra. El artículo muestra como, de nuevo, la elección del kernel width es clave en este sentido, para valores pequeños de este parámetro obtendremos buenos resultados de adherencia, pero que van en detrimento de la estabilidad, mientras que para valores grandes del kernel width sucede lo contrario. OptiLIME propone entonces un framework por el que se fija la adherencia que queremos en el modelo lineal, medida con el estadístico R^2, y busca el kernel width más grande posible para obtener dicha adherencia, maximizando de esta manera la estabilidad para el valor del R^2 dado.

5.4 Un ejemplo práctico

Procedemos a realizar un ejemplo práctico de la implementación del framework de LIME en Python. Para ello usaremos el conjunto de datos *California Housing* [1], en el que cada fila recoge los datos de un grupo de viviendas (*block group*), que son agregaciones de edificios con una población total de entre 600 y 3000 personas. Algunos de los datos que nos encontramos en cada fila son; el ingreso medio por grupo (*MedInc*), la antigüedad media de los pisos (*HouseAge*), el número medio de habitaciones por piso y grupo (*AveRooms*),... Estos datos vienen acompañados del precio medio de la vivienda en cientos de miles de dólares americanos, que actúa como etiqueta.

Nos encontramos, por lo tanto, ante un problema de regresión, en el que queremos que nuestro modelo, dados los datos de una vivienda (o grupo de viviendas del mismo grupo) sea capaz de predecir el precio.

El conjunto de datos cuenta con 8 atributos y más de 20000 instancias. Usaremos una partición de los datos 90-10 y entrenaremos un random forest en el conjunto de entrenamiento. Para esto ejecutamos el código 2.

```
from sklearn import datasets, ensemble
from sklearn.model_selection import train_test_split
import pandas as pd
import matplotlib.pylab as plt
import numpy as np

data = datasets.fetch_california_housing() #20000 instancias y 8 atributos
X = data['data']
y = data['target']
features = data['feature_names']
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.90,
                                                    random_state=50)
regressor = ensemble.RandomForestRegressor()
regressor.fit(X_train, y_train);
```

Código 2: Entrenamiento de un random forest en sklearn.

Ahora usamos el random forest para predecir el valor de una vivienda del conjunto

[1] https://scikit-learn.org/stable/modules/generated/sklearn.datasets.fetch_california_housing.html

de test, en este caso usaremos la instancia número 39. La predicción es de 4.34, y el valor real 4.959. Para poder explicar esta predicción usamos el framework de LIME como se observa en el código 3.

```
pip install -q lime
import lime.lime_tabular
#Primero instanciamos un explainer
explainer = lime.lime_tabular.LimeTabularExplainer(X_train,
                                                   feature_names=features,
                                                   mode='regression')
#Explicamos la instancia en concreto
exp_eu = explainer.explain_instance(X_test[39],
                                    regressor.predict,
                                    num_features=5,
                                    num_samples=4000,
                                    distance_metric="euclidean")
exp_eu.show_in_notebook()
```

Código 3: Uso básico de la implementación en Python de LIME.

De esta última ejecución obtenemos la salida en consola que mostramos en la figura 5.1, donde observamos los valores que toman los 5 atributos más importantes a la izquierda, los coeficientes del modelo Ridge creado por LIME en el centro, y el intervalo que contiene las predicciones del modelo complejo sobre las instancias perturbadas usadas por LIME a la izquierda.

Mediante la explicación dada por LIME podemos entender que para el modelo influyó de forma positiva el ingreso medio y la longitud, por ejemplo, mientras que la latitud fue en detrimento de la predicción del precio de la vivienda.

Como se comentaba en las secciones anteriores, LIME cuenta con diversas debilidades, entre ellas estaba la elección del kernel width, ya que era un argumento que podía alterar las explicaciones a las que llegábamos mediante LIME. En la figura 5.1 usábamos el kernel width por defecto, es decir $\sqrt{n} \cdot 0.75$, siendo n el número de atributos, en nuestro caso 8.

En la figura 5.2 podemos observar los cambios producidos en la explicación de LIME para la instancia 39 del conjunto de test considerando $\sqrt{n} \cdot 0.15$ como kernel width. Dichos cambios se producen tanto en los valores de los coeficientes del modelo lineal, afectando también al orden de importancia de los atributos seleccionados.

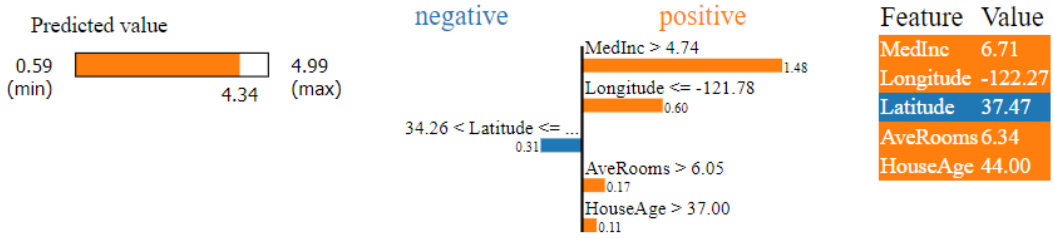

Figura 5.1: Salida por consola de la explicación para la instancia 39.

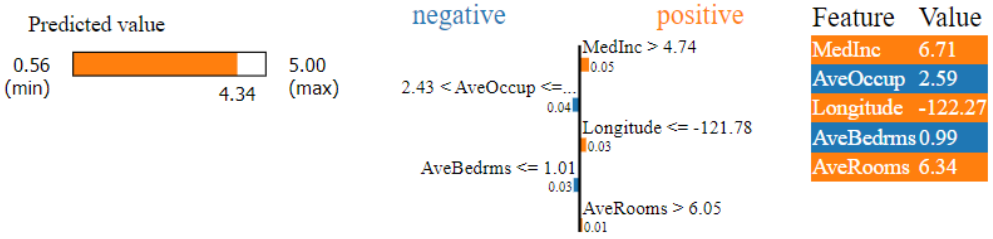

Figura 5.2: Salida por consola de la explicación para la instancia 39 para el kernel width $\sqrt{8} \cdot 0.15$.

Algorithm 2 LIME Framework

Input: Modelo complejo f, matriz de datos X, instancia de interés x, número de perturbaciones p y número de atributos para la explicación n.

Output: Explicación local para la predicción del f sobre x, $f(x)$.

1. Se discretizan los atributos de X, por defecto en cuartiles.

2. Generar p nuevas instancias en el espacio discretizado a partir de una normal tratando de imitar la distribución original de los datos usando la media y la varianza de cada atributo.

3. Generar el data set discretizado binario y sus etiquetas:

for toda instancia aleatoria i generada en el paso 2 **do**

- Usando la instancia i, tomar una instancia continua generando un valor aleatorio para cada atributo que comparte el cuartil a_i
- Obtenemos la etiqueta para la instancia i: $f(i)$

for todo atributo a **do**
  if $a_i == a_x$ **then**
    $a_i = 1$
  else
    $a_i = 0$
  end if
end for
end for

Después de este paso x pasa a estar representado por el vector de unos.

4. Obtenemos los pesos para cada i:

for toda toda instancia aleatoria i **do**

- Calcular la distancia entre el vector x e i. Por defecto la distancia euclídea.
- Determinar el peso de i aplicando la función kernel a la distancia obtenida.

end for

5. Entrenamos un modelo Ridge con los n atributos más importantes, tomados por forward selection, con los datos y sus pesos obtenidos en los dos pasos previos.

6. Los coeficientes de estos n atributos conforman la explicación para la predicción $f(x)$

6 La elección de distancia en el algoritmo de LIME

Uno de los puntos esenciales en la búsqueda de una explicación local al modelo complejo es el significado de entorno local. Como vimos antes, LIME usa un kernel exponencial, que basándose en el valor del kernel width, define el tamaño del entorno alrededor de la instancia a explicar. Esta función kernel tiene como entrada la distancia entre los datos perturbados y la instancia original. La distancia por defecto usada en LIME es la distancia euclídea, pero se abre la posibilidad a usar otras métricas.

En este capítulo se presentan los resultados obtenidos como parte de la investigación realizada durante el curso 2022/2023, donde buscamos respuesta a la pregunta de si la elección de distancia tiene efectos en las explicaciones proporcionadas por LIME, para ello, en la sección 6.1 presentamos las herramientas necesarias para entender el experimento, en la sección 6.2 comparamos la distancia euclídea y la distancia de Hamming, popularmente usada en escenarios con vectores binarios e introducimos una definición de kernel width para las distancias inducidas por las métricas L_k. En la sección 6.3 brindamos resultados empíricos mostrando la comparación de ambas distancias, concluyendo en la sección 6.4 que el desempeño de ambas es *equivalente* cuando el kernel width es tomados adecuadamente.

6.1 Antecedentes

Con el objetivo de poder comparar en términos de estabilidad el rendimiento de ambas distancias introducimos en esta sección el CSI, una medida propuesta en [41] que mide la similitud entre los coeficientes para diferentes ejecuciones de LIME sobre la misma instancia. El CSI toma valores entre 0 y 100, siendo los valores altos los que

muestran mayor similitud entre las explicaciones y, por lo tanto, mayor estabilidad. Para obtener esta medida, para cada atributo, usando que los coeficientes de un modelo Ridge siguen una distribución Normal, se calculan los intervalos de confianza para estos al 98 %. Después se toma la intersección de los intervalos para un mismo atributo en diferentes ejecuciones, codificando como 1 si los intervalos tienen intersección no nula y 0 en caso contrario. Finalmente, la media de los valores obtenidos se toma como medida de concordancia de los coeficientes para las distintas repeticiones.

Por otro lado, es conocido el empeoramiento de la fiabilidad de la distancia euclídea a la hora de capturar la noción de proximidad en espacios con alta dimensionalidad. Por ejemplo, en [42] los autores explican como usando métricas tradicionales L_k

$$L_k(x, y) = \sum_{i=1}^{n} |x_i^k - y_i^k|^{1/k}$$

en problemas con alta dimensionalidad conduce a una perdida de la noción de proximidad. Esto es un problema que deberíamos tener en cuenta en trabajos donde la proximidad juega un papel importante. En este artículo, los autores brindan un ejemplo empírico usando diferentes distancias en el popular algoritmo k-means para un problema de clasificación. Los resultados muestran como el rendimiento de la distancia euclídea, L_2, es peor comparándolo con la distancia inducida por la métrica L_1, de hecho se justifica que para mayores valores de k, peores son las distancias para establecer una buena noción de proximidad, por ello la mejor opción entre estas métricas sería tomar L_1, que induce la conocida como distancia Manhattan. En el artículo los autores van aún más lejos proponiendo lo que denominan distancias fraccionarias, con k entre 0 y 1.

A pesar del hecho de que la distancia euclídea no es suficientemente *expresiva* para describir la proximidad en espacios de alta dimensionalidad, espacios no poco frecuentes en el ámbito del aprendizaje automático, a veces se pasa por alto esta elección.

6.2 Relación entre las distancias en LIME

Conviene apreciar, como se explica en el algoritmo 2 que la distancia en el framework de LIME se toma entre vectores binarios, lo que simplifica mucho la diferencia existente entre las métricas L_k.

En este contexto la distancia de Hamming se puede expresar como:

$$Hamming(x, y) = \sum_{i=1}^{n} |x_i - y_i|$$

Que coincide exactamente con la distancia Manhattan, recordemos inducida por L_1.

Como primera observación, en el caso de vectores binarios, la única diferencia entre aplicar la distancia de Hamming o la euclídea es la raíz cuadrada, dado que $1^2 = 1$ y $0^2 = 0$, por lo tanto podemos anticipar algunos efectos que se producirán cuando usemos una o la otra.

En el caso de vectores binarios, la máxima distancia dada por Hamming en $\{0, 1\}^n$ es n, mientras que la dada por la distancia euclídea será $\sqrt{n}$, por lo explicado anteriormente. Ilustremos esta observación con un ejemplo. Consideremos todos los vectores binarios en $\{0, 1\}^n$, es decir $\{(0, 0), (0, 1), (1, 0), (1, 1)\}$. En este caso, obtenemos 2^2 vectores, las distancias dadas por Hamming están en el intervalo $[0, 2]$ y las dadas por la euclídea en $[0, \sqrt{2}] \approx [0, 1.41]$. Cuando consideramos los posibles 2^{40} vectores en el espacio $\{0, 1\}^{40}$, las distancias dadas por Hamming estarán contenidas en $[0, 40]$, mientras que las dadas por la distancia euclídea tomaran valores en $[0, \sqrt{40}] \approx [0, 6.32]$. La longitud del intervalo en el caso de la distancia euclídea crece al ritmo de la raiz cuadrada, pero el número de vectores en el espacio crece exponencialmente. Este hecho pone de manifiesto el problema estudiado en [42] y explica por qué la distancia Hamming, equivalentemente Manhattan en este contexto, preserva mejor la noción de proximidad.

En la implementación de LIME, el valor por defecto dado para el kernel width es $0.75 \times \sqrt{n}$, lo que, con la interpretación dada en el párrafo anterior, puede leerse como el 75 % del valor máximo dado por la distancia euclídea en el espacio $\{0, 1\}^n$, que es de hecho en el que estaríamos trabajando, ya que la dimensionalidad del espacio coincidirá con el número de atributos del data set. Esta interpretación explicaría que el kernel width por defecto estaría hecho a medida para la distancia euclídea, lo que podría ir en detrimento del rendimiento de otras distancias. Por ello, usando la misma lógica, definimos el kernel width especifico para la distancia Manhttan como $0.75 \cdot n$. Podemos ir más allá y definir un kernel width que *replicaría* el comportamiento de la distancia euclídea en el framework de LIME para cada distancia inducida por una métrica L_k mediante la siguiente expresion:

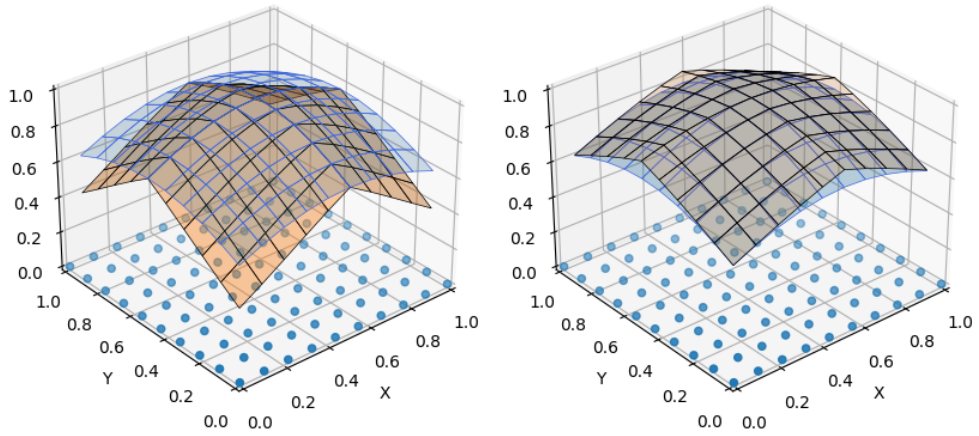

Figura 6.1: A la izquierda, el kernel usando las distancias euclídea y Manhattan con el kernel width $0.75 \cdot \sqrt{n}$. A la derecha, el kernel width usado para la distancia Manhattan es el propuesto $0.75 \cdot n$.

$$kw = 0.75 * (n)^{1/k}$$

En la figura 6.1, se muestra la comparación de los valores dados por la función kernel para las dos distancias tomando los diferentes kernel width. Para una malla 2D, se muestra el valor del kernel del grid al punto central $(\frac{1}{2}, \frac{1}{2})$. Observemos que ambos valores del kernel son más próximos en la imagen de la derecha, donde el cálculo se hizo con el kernel width propuesto.

Recordemos que en este escenario la distancia de Hamming es equivalente a la distancia Manhattan, dada por la norma L_1, y preserva mejor la noción de proximidad que la distancia euclídea. En la siguiente sección exploramos el comportamiento de estas dos distancias para data sets con distinta dimensión y también como LIME actúa cuando aplicamos el kernel width apropiado dependiendo de la distancia usada.

6.3 Experimentación

En esta sección realizamos dos experimentos que comparan la distancia euclídea y la distancia Hamming, equivalentemente Manhttan. En el primer experimento, la estabilidad y adherencia de LIME es comparada usando el kernel width por defecto y el kernel width propuesto para la distancia Manhattan, con el objetivo de encontrar

una comparación justa entre las dos distancias. En el segundo experimento la similitud entre las dos métricas es estudiada comparando el orden de importancia dado a los atributos en las explicaciones dadas por LIME en cada caso.

6.3.1 Selección del kernel width para una comparación justa

En este primer experimento, estudiamos el kernel width apropiado dependiendo de la metrica. Primero, generamos data sets sintéticos con 500 instancias y diferentes dimensiones, entre 10 y 40 con un paso de 10. Después, para cada dimensión, entrenamos un random forest usando un train set del 90 %, entonces usamos LIME para explicar las predicciones dadas por el random forest sobre todo el test set, compuesto por el restante 10 % del data set. Para estas explicaciones usamos tantos atributos como tenga el data set para asegurar que la comparación se realiza sobre los mismos atributos. Finalmente el CSI es calculado para medir la estabilidad de los coeficientes y el estadístico R^2 como medida de adherencia. Compararemos la media de estos valores sobre el test set y en cada data set.

El último paso fue calculado usando las dos opciones de kernel width. Específicamente, para la distancia euclídea usamos el 75 % de la distancia máxima, como se explica en la Sección 6.2, que es el predefinido en la implementación de LIME. Para la distancia Manhattan usamos tanto esté valor del kernel width como el propuesto en la Sección 6.2. Finalmente, el procedimiento fue repetido 10 veces. En la figura 6.2, se muestra el valor medio del CSI para las diferentes dimensiones. Se puede observar como existe una convergencia si el kernel elegido para la distancia Manhattan es el propuesto en este trabajo, y como divergen cuando el kernel es el predefinido. De la misma forma podemos apreciar los valores obtenidos para el R^2 en la figura 6.3.

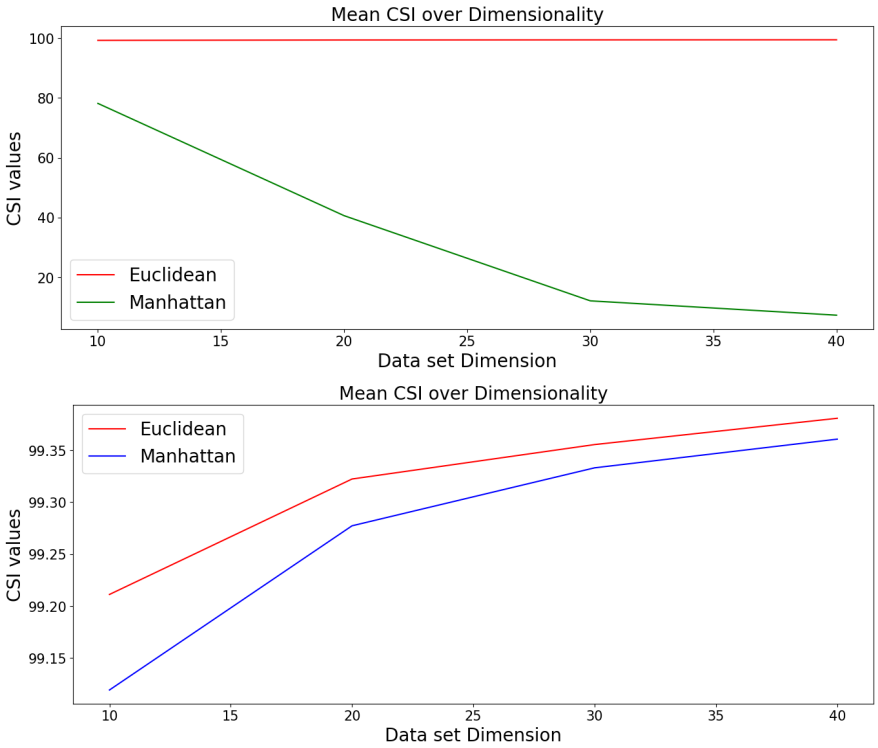

Figura 6.2: Arriba, se muestra el valor del CSI para las distancias euclídea y Manhattan usando el kernel width predefinido. Abajo, se muestra el CSI usando en el caso de la distancia euclídea el kernel width predefinido y para la distancia Manhattan el kernel width propuesto. El eje x corresponde a las dimensiones de los data sets, y el eje y a los valores del CSI. Los resultados mostrados son la media de las 10 repeticiones del experimento.

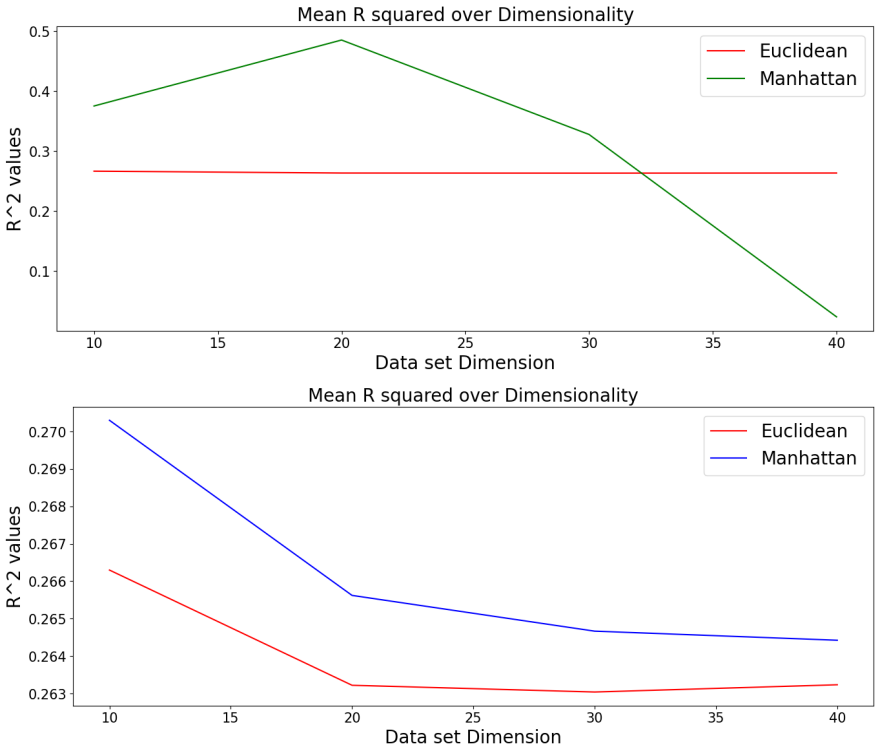

Figura 6.3: Arriba se muestran los valores del R^2 para las distancia euclídea y Manhattan cuando usamos el kernel width por defecto. Abajo, el R^2 se muestra para ambas distancias usando el kernel width por defecto en el caso de la euclídea, y el propuesto para la Manhattan. De nuevo, mostramos la media de las 10 repeticiones del experimento.

6.3.2 Comparación de las explicaciones

En este experimento, se compara la importancia dada a los atributos usando LIME. El data set usado es el UCI ML Breast Cancer (Diagnostic) dataset [43], compuesto por 569 instancias de dimension 30 para un problema de clasificación. El data set lo dividimos en train y test set en una proporción $90 - 10$ para entrenar un Random Forest. Después calculamos las explicaciones en las instancias del test set, obteniendo un vector de 30 atributos ordenados por importancia para cada instancia.

Esto lo hacemos usando el kernel width predefinido para la distancia euclídea y tanto este como el kernel width porpuesto para la distancia Manhattan. Los vectores de atributos fueron comparados como sigue: Dados dos vectores x, y con longitud

igual al número de atributos del data set, para cada atributo calculamos la diferencia entre las coordenadas que le fueron asignadas en x y en y. Finalmente, calculamos la media sobre todos los atributos de estas diferencias.

Esto nos brinda una medida de la similitud en términos de importancia de los atributos. En la Tabla 6.1, se muestran los valores del CSI y el R^2, pudiendo observar la estabilidad y adherencia alcanzada por LIME usando los diferentes kernel width. Se aprecia como los resultados del experimento 1 se mantienen en el data set real. Podemos ver los altos valores de estabilidad obtenidos por la distancia euclídea usando el kernel width predefinido y por la distancia Manhattan usando el kernel width propuesto, lo que justifica el siguiente paso del experimento. En la Tabla 6.2, se muestra la medida de similitud presentada anteriormente. Podemos ver como ambas distancias tienen un desempeño similar, mostrando que dan importancias similares a los mismos atributos. Es decir, obtenemos explicaciones similares aun cambiando las distancias, cuando elegimos el kernel width adecuado.

		CSI		R^2	
Distance	Kernel Width	Mean	Variance	Mean	Variance
Euclidean	$0.75 \cdot \sqrt{n}$	99.39	0.1	0.24	0.02
Manhattan	$0.75 \cdot \sqrt{n}$	9.48	8.53	0.47	0.08
Manhattan	$0.75 \cdot n$	99.35	0.12	0.24	0.02

Cuadro 6.1: Valores del CSI y R^2 para las distintas elecciones de Kernel width. Mayores valores del CSI se traducen en una mayor estabilidad, mayores valores del R^2 en mayor adherencia.

kernel width	Max	Mean	Min
$0.75 \cdot \sqrt{n}$	23.93	8.69	0.28
$0.75 \cdot n$	1.37	0.19	0

Cuadro 6.2: Medida de similitud entre el orden de importancia dado por la distancia euclídea y la distancia Manhattan para las dos elecciones de kernel width. Los resultados mostrados en la tabla son los valores medios sobre el test set. En esta tabla podemos apreciar el desempeño similar entre las dos distancias cuando usamos el kernel width propuesto.

6.4 Conclusiones de la experimentación

La explicabilidad va a ser una de las partes clave de todo modelo de inteligencia artificial en los próximos años. Solo modelos trasparentes y confiables serán aceptados para su uso en un contexto social. En este escenario, será totalmente necesario una buena base teórica de cara a dar las explicaciones. Esta investigación es un paso en dicha dirección.

En este capítulo hemos estudiado la relación entre las distancias euclídea y Manhattan en este contexto, concluyendo su similitud. De forma empírica, hemos mostrado que la estabilidad de LIME converge y que el orden de importancia de atributos es similar cuando se adapta el kernel width a la distancia.

En los fundamentos del aprendizaje automático, es bien conocido que la distancia euclídea pierde la noción de proximidad para alta dimensionalidad, mientras que la distancia Hamming realiza un mejor desempeño en ese aspecto. Sin embargo, la aplicación de la función kernel junto con la elección correcta del kernel width parecen eliminar dichas posibles diferencias. En un trabajo futuro se podría investigar como afecta el uso de la función kernel en la eliminación de diferencias entre distancias (usando el kernel width adecuado), provocando que el mejor desempeño de distancias distintas a la euclídea en altas dimensionalidades se vea *diluido*, posiblemente por el hecho de que la función kernel sea contractiva.

7 | Conclusiones

Los hitos conseguidos por el aprendizaje automático en la última década son de una magnitud que era impensable hace no mucho. La aplicación de estos modelos en muchos ámbitos de la vida cotidiana no va a hacer más que crecer durante los próximos años, lo que hace ineludible la importancia de la comprensión de los procesos que llevan a estos sistemas a dar ciertas predicciones. Es por eso que la explicabilidad será un factor clave a la hora de poder implementar este tipo de soluciones, ya que solo modelos transparentes y confiables serán aceptados para su uso.

Recientemente, el campo de la explicabilidad ha sido sujeto de una gran atención por parte de la comunidad científica, lo que ha generado multitud de resultados y métodos como LIME (2016) y su multitud de mejoras propuestas, DLIME (2019), ALIME (2019) o OptiLIME (2020). Las explicaciones generadas por LIME consiguen, mediante la aproximación con un modelo lineal, arrojar luz sobre qué atributos y con qué influencia participan en la predicción del modelo. En este trabajo hemos estudiado dichas explicaciones para datos tabulares, observando que la elección de distancia en el framework de LIME era una opción tal vez poco estudiada.

Los resultados mostrados en el Capítulo 6 hacen ver que el valor del kernel width por defecto propuesto en LIME está sesgado por la distancia euclídea, haciendo que la aplicación de otras distancias sea poco óptima si se pasa por alto la modificación de este argumento, y que en caso de modificarlo, pueda ser difícil encontrar un intervalo que produzca resultados con sentido. En este trabajo mostramos una definición de kernel width que permite equiparar los resultados obtenidos con la distancia euclídea con los obtenidos mediante las distancias inducidas por las métricas L_k.

Este último resultado hace pensar que la aplicación de la función kernel en el framework de LIME junto con la simplificación a un espacio binario hace que las posibles diferencias entre las distancias se hagan menos importantes cuando se toma el kernel width adecuado. En futuros trabajos podría estudiarse, por lo tanto, si para toda

distancia, se puede encontrar un kernel width que la hace equivalente a la distancia euclídea en LIME y entonces a todas las demás, y si así fuera, si el argumento para la elección de la distancia en LIME sobra.

Por otro lado, uno de los motivos que nos hizo investigar las repercusiones de usar distintas distancias en LIME fue el conocido como *curse of dimensionality*, que se refiere a la perdida de la noción de proximidad en espacios de alta dimensionalidad con el uso de distancias tradicionales como la euclídea. En futuros trabajos podría ser interesante estudiar en profundidad este hecho cuando los espacios son binarios, como en el caso de la implementación de LIME, ya que en caso de que el problema persistiera de manera notable en este tipo de espacios, la aplicación de LIME en problemas con una gran cantidad de atributos podría llevar a explicaciones erróneas o poco fiables.

Para finalizar, desde mi punto de vista, las bases del aprendizaje automático están pobladas por matemáticas, de hecho, el último gran avance de las redes neuronales se consiguió gracias a la aplicación de algo tan extendido en el ámbito de las matemáticas como los algoritmos de gradiente y la regla de la cadena. Sin embargo, el motor del campo del aprendizaje automático parece estar alimentado por resultados empíricos y poco respaldados por una teoría sólida. Parece que esta tendencia ha cambiado en los últimos años y esto me hace pensar que viene una época de grandes avances gracias a la mayor aplicación de las matemáticas en el desarrollo de este campo.

Bibliografía

[1] Murray Campbell, A. Joseph Hoane, and Feng-hsiung Hsu. Deep blue. *Artif. Intell.*, 134(1–2):57–83, jan 2002.

[2] J. McCarthy, M. L. Minsky, N. Rochester, and C. E. Shannon. A PROPOSAL FOR THE DARTMOUTH SUMMER RESEARCH PROJECT ON ARTIFICIAL INTELLIGENCE. http://www-formal.stanford.edu/jmc/history/dartmouth/dartmouth.html, 1955.

[3] Edward A. Feigenbaum and Bruce G. Buchanan. Dendral and meta-dendral: roots of knowledge systems and expert system applications. pages 233–240, 1994.

[4] François Chollet. *Deep Learning with Python.* Manning, November 2017.

[5] Tom M Mitchell. *Machine learning*, volume 1. McGraw-hill New York, 1997.

[6] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C.J. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012.

[7] Pierre Sermanet, David Eigen, Xiang Zhang, Michael Mathieu, Rob Fergus, and Yann LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. 2014. Publisher Copyright: © 2014 International Conference on Learning Representations, ICLR. All rights reserved.; 2nd International Conference on Learning Representations, ICLR 2014 ; Conference date: 14-04-2014 Through 16-04-2014.

[8] Ivens Portugal, Paulo S. C. Alencar, and Donald D. Cowan. The use of machine learning algorithms in recommender systems: A systematic review. *CoRR*, abs/1511.05263, 2015.

[9] Jiahui Liu, Peter Dolan, and Elin Pedersen. Personalized news recommendation based on click behavior. pages 31–40, 02 2010.

[10] Amr Ahmed, Bhargav Kanagal, Sandeep Pandey, Vanja Josifovski, Lluis Pueyo, and Jeff Yuan. Latent factor models with additive and hierarchically-smoothed user preferences. pages 385–394, 02 2013.

[11] Andre Esteva, Brett Kuprel, Roberto A. Novoa, Justin Ko, Susan M. Swetter, Helen M. Blau, and Sebastian Thrun. Dermatologist-level classification of skin cancer with deep neural networks. *Nature*, 542:115–, January 2017.

[12] Olvi L. Mangasarian, W. Nick Street, and William H. Wolberg. Breast cancer diagnosis and prognosis via linear programming. *Operations Research*, 43(4):570–577, 1995.

[13] Ravindra Parmar. Common loss functions in machine learning.

[14] Finale Doshi-Velez and Been Kim. Towards a rigorous science of interpretable machine learning, 2017. cite arxiv:1702.08608.

[15] European Commission. Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation) (Text with EEA relevance), 2016.

[16] W. James Murdoch, Chandan Singh, Karl Kumbier, Reza Abbasi-Asl, and Bin Yu. Definitions, methods, and applications in interpretable machine learning. *Proceedings of the National Academy of Sciences*, 116(44):22071–22080, oct 2019.

[17] Christoph Molnar. *Interpretable Machine Learning*. 2 edition, 2022.

[18] Claude Elwood Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27:379–423, 1948.

[19] T. Hastie, R. Tibshirani, and J.H. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer series in statistics. Springer, 2009.

[20] Warren Mcculloch and Walter Pitts. A logical calculus of ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:127–147, 1943.

[21] Jaspreet. A concise history of neural networks, 2016.

[22] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, 1958.

[23] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[24] Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010.

[25] Muhammad Ardi. Simple neural network on mnist handwritten digit dataset.

[26] Ali Soleymani. Stop using random forest feature importances take this intuitive approach instead, 2022.

[27] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[28] L. Breiman, Jerome H. Friedman, Richard A. Olshen, and C. J. Stone. Classification and regression trees. 1984.

[29] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. "why should i trust you?": Explaining the predictions of any classifier, 2016.

[30] European Commision. White Paper On Artificial Intelligence - a European approach to excellence and trust, Brussel, 19.2.2020, com(2020) 65 final. Accessed: 2023-03-14.

[31] P. Jonathon Phillips, Carina A. Hahn, Peter C. Fontana, Amy N. Yates, Kristen Greene, David A. Broniatowski, and Mark A. Przybocki. Four principles of explainable artificial intelligence. national institute of standards and technology. us department of commerce. https://nvlpubs.nist.gov/nistpubs/ir/2021/NIST.IR.8312.pdf. Accessed: 2023-03-14.

[32] Yiming Zhang, Ying Weng, and Jonathan Lund. Applications of explainable artificial intelligence in diagnosis and surgery. *Diagnostics*, 12(2), 2022.

[33] Ouren Kuiper, Martin van den Berg, Joost van der Burgt, and Stefan Leijnen. Exploring explainable ai in the financial sector: Perspectives of banks and supervisory authorities, 2021.

[34] Shahin Atakishiyev, Mohammad Salameh, Hengshuai Yao, and Randy Goebel. Explainable artificial intelligence for autonomous driving: A comprehensive overview and field guide for future research directions, 2021.

[35] Mengnan Du, Ninghao Liu, and Xia Hu. Techniques for interpretable machine learning, 2018.

[36] A. E. Hoerl and R. W. Kennard. Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, 12:55–67, 1970.

[37] Muhammad Rehman Zafar and Naimul Mefraz Khan. Dlime: A deterministic local interpretable model-agnostic explanations approach for computer-aided diagnosis systems, 2019.

[38] Sharath M. Shankaranarayana and Davor Runje. Alime: Autoencoder based approach for local interpretability, 2019.

[39] Sheng Shi, Yangzhou Du, and Wei Fan. An extension of LIME with improvement of interpretability and fidelity. *CoRR*, abs/2004.12277, 2020.

[40] Giorgio Visani, Enrico Bagli, and Federico Chesani. OptiLIME: Optimized LIME explanations for diagnostic computer algorithms. *CoRR*, abs/2006.05714, 2020.

[41] Giorgio Visani, Enrico Bagli, Federico Chesani, Alessandro Poluzzi, and Davide Capuzzo. Statistical stability indices for LIME: Obtaining reliable explanations for machine learning models. *Journal of the Operational Research Society*, 73(1):91–101, feb 2021.

[42] Charu C. Aggarwal, Alexander Hinneburg, and Daniel A. Keim. On the surprising behavior of distance metrics in high dimensional space. In *Lecture Notes in Computer Science*, pages 420–434. Springer, 2001.

[43] William Wolberg, Olvi Mangasarian, Nick Street, and W. Street. Breast Cancer Wisconsin (Diagnostic). UCI Machine Learning Repository, 1995. DOI: https://doi.org/10.24432/C5DW2B.