UNIVERSIDAD
DE GRANADA

Desarrollo de modelos de Inteligencia Artificial Interpretables en Aprendizaje Federado: Fusión de árboles de decisión

MEMORIA PRESENTADA POR

Alberto Argente del Castillo Garrido

PARA OPTAR AL GRADO DE DOCTOR
POR LA UNIVERSIDAD DE GRANADA DENTRO DEL
PROGRAMA DE DOCTORADO EN TECNOLOGÍAS DE
LA INFORMACIÓN Y LA COMUNICACIÓN

DIRECTORES

María Victoria Luzón García y **Francisco Herrera Triguero**

Granada, 9 de Abril de 2024

Editor: Universidad de Granada. Tesis Doctorales
Autor: Alberto Argente del Castillo Garrido
ISBN: 978-84-1195-393-1
URI: https://hdl.handle.net/10481/93123

Esta tesis doctoral ha sido soportada por los fondos del Plan de Recuperación, Transformación y Resiliencia, financiadas por la Unión Europea (Next Generation), el proyecto del Gobierno de España que traza la hoja de ruta para la modernización de la economía española, la recuperación del crecimiento económico y la creación de empleo, para la reconstrucción económica sólida, inclusiva y resiliente tras la crisis de la COVID19, y para responder a los retos de la próxima década.

Agradecimientos

Gracias a todos y a todas

Índice

Lista de Figuras

Lista de Tablas

Resumen

Los sistemas de Inteligencia Artificial están presentes en el día a día de las personas, ofreciendo múltiples avances a los usuarios de estos. El aumento de estos sistemas crece en paralelo con la preocupación por la privacidad de los datos que se usan para crear estos sistemas por parte de los usuarios, y de diversas instituciones oficiales. Algunos de estos sistemas de Inteligencia Artificial pueden tener un impacto muy alto en las personas, como por ejemplo un sistema de diagnóstico de cáncer, y es inaceptable correr el riesgo de usar estos sistemas en beneplácito de sus características. Es necesaria la creación de sistemas de Inteligencia Artificial fiables para garantizar que estos sean robustos, transparentes, no discriminatorios, accesibles y respetuosos con la privacidad de los datos de los usuarios.

El Aprendizaje Federado es un paradigma de Aprendizaje Automático distribuido y colaborativo que surge como una solución para asegurar la privacidad y gobernanza de los datos de los usuarios, dado que no es necesario el intercambio de datos privados entre los nodos que entrenan el modelo de Inteligencia Artificial. Además de que un sistema de Inteligencia Artificial mantenga la privacidad y la seguridad de los datos de los usuarios, es necesario que éste sea interpretable, sobre todo en campos donde manejan datos sensibles. De esta forma se podrían utilizar estos sistemas, que ayudarían a los expertos en la toma de decisiones final, no sólo por ofrecer la predicción del sistema, sino porque también proporcionan los motivos por los que se ha tomado esta decisión.

La hipótesis central de esta tesis es que los modelos interpretables tradicionales del Aprendizaje Automático centralizado

pueden mantener su interpretabilidad en un entorno federado. La creación de estos modelos permitirá crear sistemas de Inteligencia Artificial fiables que puedan llegar a ser utilizados en ámbitos donde la privacidad de los datos sea imperativa. Para corroborar la hipótesis de partida, se han marcado los siguientes objetivos:

1. Desarrollo de una metodología que permita unificar el proceso de entrenamiento de nuevos modelos en entornos de Aprendizaje Federado.

2. Diseño, implementación y evaluación de un agregador de Aprendizaje Federado para Árboles de Decisión basados en reglas. Como resultado se ha desarrollado la propuesta de agregación de árboles de decisión ICDTA4FL.

3. Desarrollar un software de Aprendizaje Federado que permita la implementación de agregadores para los diferentes modelos de Aprendizaje Federado interpretables y explicables. Este software permitirá añadir agregadores para modelos de Aprendizaje Automático interpretables, y que permita realizar la experimentación en diferentes entornos federados. Como resultado se ha desarrollado el software FLEX-Trees.

Esta tesis logra aunar la privacidad de los datos junto a la interpretabilidad de sistemas de Inteligencia Artificial en entornos de Aprendizaje Federado, con la creación de un proceso de agregación de modelos interpretables, que permite adaptar un modelo clásico de Aprendizaje Automático, como lo son los Árboles de Decisión, a un entorno de Aprendizaje Federado. También se ha realizado un software que incluye algunos de los modelos interpretables del estado del arte para Aprendizaje Federado, y todo esto llevando a cabo una metodología de trabajo de entrenamiento y evaluación en entornos de Aprendizaje Federado.

Contexto y objetivos

Contexto

Las aplicaciones de Inteligencia Artificial (IA) están presentes en nuestro día a día gracias a que se dispone de una cantidad de datos nunca antes vista. Debido a la cantidad de dispositivos que están recogiendo datos diariamente, y al crecimiento de la IA, hace que sea necesaria la búsqueda de soluciones de sistemas de IA confiables (Trustworthy Artificial Intelligence (TAI)) [TLS20]. Tradicionalmente el enfoque de los modelos de Aprendizaje Automático (AA) y de Aprendizaje Profundo (AP), buscan modelos con mejor rendimiento y que puedan capturar la complejidad de los datos. Sin embargo, la búsqueda de estos modelos ha dejado de lado la privacidad y la protección de datos de los usuarios, haciendo que sea fundamental la regulación de los sistemas de IA para que estos sean justos. La regulación más avanzada actualmente es la propuesta de Ley de IA de la Comisión Europea [Act21]. En esta ley se definen cuatro niveles de riesgo para los sistemas de IA para catalogarlos en base al riesgo de los mismos, y estos son: (1) Riesgo mínimo, como en los sistemas de recomendación en comercios electrónicos, (2) Riesgo limitado, como *chatbots*, (3) Riesgo alto, como vehículos autónomos, diagnóstico médico, y (4) Riesgo inaceptable, como sistemas de reconocimiento facial o puntuación social.

Algunos de estos sistemas, como los de riesgo mínimo, están muy presentes en el día a día, y no suponen apenas riesgo para los usuarios. Con los sistemas de riesgo limitado, los usuarios deben saber que están interactuando con una máquina, para

saber si deciden continuar con la interacción o no. Los sistemas de alto riesgo pueden llegar a tener un impacto muy alto en las personas, y es necesario tratarlos específicamente para poder llegar a utilizarlos en nuestro día a día si se elimina el riesgo de estos. Los sistemas cuyo riesgo es inaceptable, como los de reconocimiento facial, son un claro problema para la seguridad y los derechos de las personas y están prohibidos en el mercado europeo [DRDC+23]. Para lograr afrontar estos riesgos, es necesario crear sistemas TAI. La confianza de un sistema de IA se observa comprobando si el sistema funciona como debería para el problema que va a resolver, y bajo los correctos criterios de seguridad [TG20]. Aún así, la confianza del sistema no involucra sólo al sistema, sino que también involucra a otros actores externos al sistema y que participan en su ciclo de vida. Por ello es necesario crear un análisis sistemático con pilares y requisitos que contribuyan a la creación de sistemas de IA fiables.

La confianza de un sistema de IA fiable, desde un punto de vista técnico, y de acuerdo a la propuesta europea [Act21] se basa en siete requisitos técnicos que están sujetos a tres pilares clave: (1) ética, (2) robustez, y (3) legalidad [DRDC+23]. Por su parte, los siete requisitos son: 1. agencia humana y supervisión, 2. robustez técnica y seguridad, 3. privacidad y gobernanza de datos, 4. transparencia, 5. diversidad, no discriminación y equidad, 6. bienestar social y ambiental, y 7. responsabilidad. La Figura 1 muestra los pilares y los requisitos de un sistema de IA fiable comentados previamente.

Todos los requisitos mostrados en la Figura 1 son muy importantes de cara a desarrollar un sistema de IA fiable, sin embargo, cabe destacar la importancia del requisito 3. El requisito 3, privacidad y gobernanza de datos, juega un rol fundamental dado que los sistemas de IA se crean a partir de los datos de los usuarios, por lo que es importante salvaguardar los datos privados de estos durante el ciclo de vida del sistema [DRDC+23]. Los usuarios deben tener el control total sobre sus datos durante el ciclo de vida del sistema de IA que, junto a la privacidad, generará confianza en los usuarios del sistema a la hora de usarlo [TLS20].

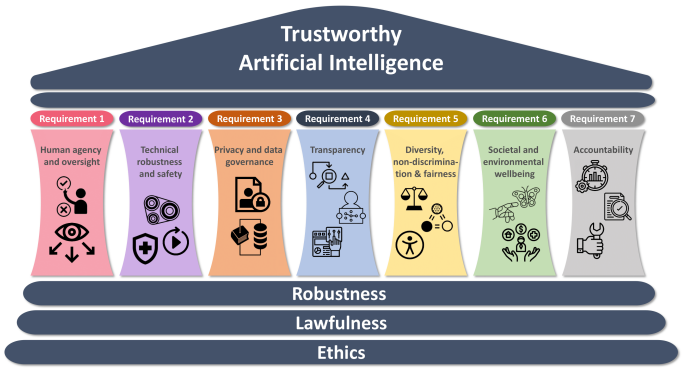


Figura 1: Pilares y requisitos de un sistema de IA fiable [DRDC+23].

El Aprendizaje Federado (AF) es un paradigma de AA que surge como una solución a la intersección entre privacidad y el AA distribuido [MMR+17], que es particularmente relevante en el contexto de la privacidad y la protección de datos, donde la confianza en el ciclo de vida de un sistema de IA es necesario [AASB+23]. El AF permite el entrenamiento de modelos de AA de forma distribuida sin la necesidad de que los nodos participantes intercambien sus datos privados, por lo que no se compromete la privacidad de los datos de estos nodos a la hora de entrenar estos modelos. En un entorno de AF los nodos no envían ningún dato privado al resto de nodos, en su lugar envían el modelo que se está entrenando, el cual será agregado dando lugar a un modelo global. De esta forma se logra agregar en un único modelo todo el conocimiento de los nodos participantes sin la necesidad de comprometer los datos privados de los nodos. El proceso de agregación es clave, ya que de este depende el rendimiento del modelo global obtenido.

Hay escenarios, como la medicina, en los que es imperativo que los datos no salgan del propio centro de datos para salvaguardar la seguridad y privacidad de los datos de los pacientes. En estos casos es posible que se disponga de pocos datos, por lo que es posible que no se pueda disponer de un modelo de AA con buen rendimiento. En este escenario, el AF ayudaría a que varios centros pudieran obtener un modelo de AF con un buen rendimiento sin compartir ningún dato privado, manteniendo la seguridad de los datos de los pacientes, y aumentando el conjunto de datos de entrenamiento del modelo [LMM+22].

Por otra parte, entre los requisitos de un sistema de IA está la transparencia y la explicabilidad, que son cruciales en muchos escenarios, por lo que es importante la creación de sistemas de IA que ofrezcan estos aspectos [ADRD+20] para así poder ser utilizados en campos que manejan datos sensibles como la medicina o las finanzas. Los sistemas de IA basados en modelos de AP logran un excelente rendimiento, tanto en entornos centralizados tradicionales como en AF [MMR+17]. Sin embargo estos sistemas son modelos de caja negra y, pese a que se esté mejorando la transparencia y la explicabilidad de estos modelos, no hay que descartar otros modelos clásicos de AA tradicional que, aunque no obtengan el mismo rendimiento que los modelos de caja negra, sí que integran interpretabilidad y explicabilidad a las predicciones del modelo, haciéndolos adecuados para este tipo de campos [ADRD+20].

Un ejemplo canónico de modelos interpretables son los árboles de decisión (AD), cuya arquitectura en forma de árbol se asemeja al proceso de razonamiento del humano, lo cual los hace inherentemente interpretables [RM14], facilitando así la transparencia del modelo y proporcionan confianza a las partes interesadas de un entorno de AF. Los AD son ligeros y computacionalmente eficientes, lo cual los hace adecuados para ser desplegados en dispositivos con pocos recursos, habituales en los entornos de AF. Estas propiedades habilitan el entrenamiento de los modelos de AD sin comprometer el rendimiento de los dispositivos, o sin requerir muchos recursos de los mismos.

Sin embargo, los AD no se pueden agregar directamente como si los modelos de AA que se pueden expresar como vectores de parámetros. En este caso, los árboles de decisión tienen una arquitectura jerárquica con caminos de decisión ramificados. La agregación de AD requiere de técnicas que puedan fusionar los caminos de decisión de los árboles de los diferentes nodos del entorno federado, mientras se conserve la interpretabilidad y la integridad del modelo global resultante. Además, los AD tienden a ser sensibles a las distribuciones de datos y la varianza de estos, la cual es probable que se de en entornos federados.

Motivación e hipótesis

Para poder llevar a cabo el entrenamiento de un modelo de IA en un entorno de AF es necesario seguir una metodología que asegure que el modelo global resultante va a cumplir con los requisitos de privacidad del AF, como el salvaguardar la privacidad de los datos de los clientes, y que sirva para otros investigadores como guía para llevar a cabo el entrenamiento de un modelo en un entorno de AF. Sin embargo, no hay una metodología clara que defina cómo se debe realizar el proceso de entrenamiento del modelo en un entorno de AF y que ayude al resto de investigadores o desarrolladores a unificar este proceso.

En la literatura se encuentran múltiples *frameworks* que facilitan el entrenamiento de un modelo de AA en un entorno de AF, como TensorFlow Federated [The18] o Flower [BTM+20], entre otros. Sin embargo, estos *frameworks* no permiten una rápida simulación en multitud de entornos que se pueden dar en AF, por lo que es necesario la creación de un *framework* de AF que permita la simulación de experimentos federados y que permita que los investigadores puedan desarrollar nuevos métodos novedosos. Además es importante que los *frameworks* permitan realizar una metodología similar a la que se daría en el mundo real para que se pueda obtener el rendimiento de los modelos novedosos de manera simulada, antes de ser llevados a un entorno real.

La agregación de múltiples AD distribuídos sin que estos introduzcan sobreajuste, sesgos o ruido al modelo global, es también un reto en la agregación de AD federados. Es necesario la creación de un agregador para AD que asegure que el modelo global sea robusto y generalizable en todos los nodos que han participado en el entrenamiento del modelo.

La hipótesis de partida que sustenta esta tesis doctoral es que los modelos interpretables existentes en la literatura pueden mantener su interpretabilidad en AF mediante la creación de nuevos mecanismos de agregación que permitan su correcta adaptación. El uso de modelos interpretables en AF permite la creación de sistemas de IA fiables y que se adaptan a los requisitos y pilares que deben de tener estos sistemas para poder ser utilizados en entornos reales.

Objetivos

Para poder comprobar la hipótesis se plantean varios objetivos, que se presentan a continuación:

1. Desarrollo de una metodología que permita unificar el proceso de entrenamiento de nuevos modelos en entornos de AF.

2. Diseño, implementación y evaluación de un agregador de AF para AD basados en reglas.

3. Desarrollar un software de AF que permita la implementación de agregadores para los diferentes modelos de AF interpretables y explicables. Este software permitirá añadir agregadores para modelos de AA interpretables, y que permita realizar la experimentación en diferentes entornos federados.

Estructura de la memoria

La memoria está estructurada en cuatro capítulos, según se detalla a continuación:

Capítulo 1. Se presenta el AF y sus bases, así como su flujo de trabajo y los elementos clave. Este capítulo aborda el *objetivo 1*, y presenta una metodología de trabajo para entrenar modelos en AF, además de mostrar varios ejemplos de cómo debe aplicarse.

Capítulo 2. En este capítulo se aborda el *objetivo 2*, presentando un proceso de agregación de árboles de decisión para un entorno de AF, llamado *ICDTA4FL*. Para ello, se presentan dos modelos, *ICDTA4FL-ID3* y *ICDTA4FL-CART*, con los que se valida el proceso propuesto.

Capítulo 3. Este capítulo aborda el *objetivo 3* mediante el desarrollo de un software de AF, *FLEX-Trees*, que incluye modelos de AF interpretables. Este software permite llevar a cabo correctamente la metodología propuesta, y ayuda a la integración de nuevos modelos de AD a un entorno de AF.

Capítulo 4. Este último capitulo recoge las conclusiones obtenidas de los estudios previos, así como las publicaciones asociadas y las futuras líneas de trabajo.

1 Preliminares: Aprendizaje Federado

Cuando la privacidad es obligatoria para entrenar un modelo de AA, surge el AF como un paradigma de AA descentralizado que permite entrenar modelos de AA de forma distribuida sin que los nodos implicados compartan sus datos privados, reduciendo la exposición de datos sensible y la posibilidad de que estos se filtren. El AF ha ganado mucha popularidad en los últimos años en las aplicaciones del mundo real, gracias a la eficiencia a la hora de entrenar modelos de AA entre múltiples nodos participantes. El objetivo de este capítulo es el de presentar el AF, y se divide en las siguientes secciones: (1) la Sección 1.1 presenta los fundamentos del AF; (2) la Sección 1.2 presenta una metodología de trabajo a seguir para AF, para así poder trabajar en un entorno federado: (3) la Sección 1.3 presenta los conjuntos de datos federados por excelencia, así como diferentes *frameworks* para realizar experimentos de AF; (4) la Sección 1.4 muestra casos de uso de cómo realizar diferentes experimentos federados; y, (5) la Sección 1.5 presenta unas breves conclusiones del capítulo.

1.1. Fundamentos del Aprendizaje Federado

El incremento de dispositivos electrónicos como *tablets*, móviles, cámaras, etc., permiten el acceso a un número de datos sin precedentes. La disponibilidad de esta cantidad de datos, acompañada de una época donde la IA está cada vez más presente en nuestro entorno, alimenta el crecimiento de aplicaciones inteligentes que puedan ayudarnos en el día a día. Tradicionalmente los datos utilizados para entrenar estos modelos de IA se concentraban en un servidor central, recogiendo los datos privados y sensibles de los usuarios. La recolección de estos datos significa que existen riesgos y responsabilidades al almacenarlos en un servidor central, vulnerando así la privacidad de los usuarios [GXP+20]. Estos motivos, acompañados de la necesidad de crear sistemas de IA fiables [DRDC+23], conllevan a nuevos retos como son:

- Privacidad de los datos: En el Aprendizaje Automático Centralizado (AAC), los datos de los usuarios son recogidos en un servidor central, vulnerando la privacidad de estos, al tener que ceder el usuario sus datos a un tercero. Esto es importante sobre todo en campos como la salud y las finanzas, entre otros [GXP+20]. Además, con la creciente preocupación por salvaguardar la privacidad de los datos de los usuarios, que se manifiesta en el ámbito legal, como por ejemplo las publicadas recientemente en [Eur19], conllevan a la necesidad de desarrollar sistemas de IA que preserven la privacidad de los usuarios.

- Costes de comunicación y latencia: En el AAC es necesario mandar todos los datos a un servidor central para poder procesarlos y usarlos para crear un sistema de IA. El envío de datos supone un problema especialmente con datasets largos, puesto que los errores en la conexión pueden suponer la pérdida de datos. Además, debido al aumento de dispositivos electrónicos, ya no sólo por móviles o *tablets*, sino también por dispositivos de IoT (Internet of Things) [MNG+17], supone un reto el procesar y almacenar constantemente nuevos datos de diferentes fuentes.

- Limitación al acceso de los datos: En muchos casos los datos se encuentran distribuídos entre diferentes instituciones u organizaciones, haciendo que sea difícil que compartan datos entre ellos. Un ejemplo claro es el de los hospitales, que no pueden compartir datos de los pacientes por motivos de privacidad, pero puede ser interesante una colaboración entre ellos con tal de poder obtener un sistema de IA que permita ayudar a los expertos en su trabajo en el día a día.

Para afrontar estos retos surge el AF [MMR+17], un paradigma de AA que permite el entrenamiento de modelos de IA de forma descentralizada y colaborativa entre múltiples nodos sin compartir datos privados entre los nodos participantes. El AF involucra la colaboración de clientes o nodos propietarios de datos $C_1, C_2, ..., C_n$, y se divide en dos fases principales:

1. Fase de entrenamiento: En esta fase se entrena un modelo de IA entre los diferentes clientes sin que se intercambie ningún dato de estos. Cada cliente entrena localmente un modelo con sus datos, y comparte información de este modelo. Tras ello, los modelos locales son agregados para formar un modelo de aprendizaje global.

2. Fase de inferencia: El modelo entrenado de forma colaborativa se utiliza en la predicción de nuevas instancias.

Este proceso colaborativo puede ser tanto síncrono como asíncrono, dependiendo de la disponibilidad de los datos, los nodos disponibles y el modelo a entrenar. Lo importante de este proceso es que, además de asegurar la privacidad de los datos, se logra compartir los beneficios de entrenar un modelo de forma colaborativa con una mayor cantidad de datos locales de las que disponían los clientes originalmente.

Después de haber visto el proceso general del entrenamiento en un entorno de AF, se puede definir este proceso formalmente como sigue. Se asume un conjunto de clientes $C_1, C_2, ..., C_n$, con sus respectivos datos locales $D_1, D_2, ..., D_n$. Cada cliente C_i entrena un modelo local L_i, teniendo así n modelos locales $L_1, L_2, ..., L_n$. El AF tiene como objetivo entrenar un modelo global G, a través de un proceso iterativo de aprendizaje conocido como rondas de

aprendizaje. En cada ronda de aprendizaje t, cada cliente C_i entrena un modelo de aprendizaje sobre sus datos locales D_i^t, actualizando los parámetros locales L_i^t a $\hat{L}_i^t$. Tras esto, los clientes mandan al servidor los parámetros locales para que los agregue, dando lugar a los parámetros globales G^t. Los parámetros globales se obtienen agregando los parámetros locales $\hat{L}_1, \hat{L}_2, ..., \hat{L}_n$ mediante un operador de agregación Δ. El servidor mandará los parámetros globales a los clientes para que estos actualicen sus parámetros locales del siguiente modo:

$$\begin{aligned} G^t &= \Delta(\hat{L}_1^t, \hat{L}_2^t, \ldots, \hat{L}_n^t) \\ L_i^{t+1} &\leftarrow G^t, \quad \forall i \in \{1, \ldots, n\}. \end{aligned} \tag{1.1}$$

Las rondas de aprendizaje continúan realizándose hasta que se cumplan los criterios de parada del modelo. De este modo, el modelo global G, contendrá el conocimiento general de todos los clientes que participaron en el entrenamiento.

El diseño de experimentos en AF, por tanto, es capaz de proporcionar soluciones a los retos vistos que se dan en el AA clásico [LRBAG+24]:

- Privacidad de los datos: El AF permite que los modelos se entrenen donde los datos estén almacenados, sin necesidad de compartir la información con un servidor central. Esto permite que los usuarios mantengan sus datos en sus dispositivos y no se vulnere su privacidad.

- Costes de comunicación y latencia: El AF afronta este reto limitando las comunicaciones a tan solo el intercambio de las actualizaciones del modelo a entrenar. De esta forma se solucionan los problemas de latencia y ancho de banda que se presentan en un entrenamiento distribuído, suponiendo así una mejora sobre estos sistemas de AA distribuído.

- Limitación al acceso de los datos: El AF soluciona el problema del acceso a los datos de los usuarios habilitando la colaboración entre los diferentes nodos sin tener que compartir sus datos [MMR+17]. De esta forma se elimina la necesidad

de un servidor central en el que almacenar los datos, y se posibilita la creación de aplicaciones de AA en campos donde la privacidad de los datos limita que estos se compartan.

El AF ofrece un enfoque prometedor para entrenar modelos de AA en entornos distribuídos, mientras que es capaz de afrontar los retos de privacidad, comunicación y limitación de acceso a los datos propios del AAC. En campos como el de la salud, que tienen mayores restricciones de privacidad de los datos por el tipo de dato propio de estos, el AF es una alternativa muy llamativa, ya que habilita a las diferentes organizaciones o instituciones a querer entrenar un modelo de forma colaborativa sin el riesgo de vulnerar la privacidad de sus clientes. De esta forma se logra entrenar un modelo de AA sobre una mayor cantidad de datos de las que se disponían originalmente.

1.1.1. Flujo de trabajo y elementos clave

Tras haber introducido el AF, se procede a hablar del flujo de trabajo y de los elementos clave del proceso de AF. En la Figura 1.1 se muestran los pasos que componen el entrenamiento de un modelo federado. A continuación se explican detalladamente los pasos del flujo de trabajo de un entorno federado, y se especifican los elementos clave [RBSJL+20] que surgen de cada uno de estos pasos.

Entrenamiento local El flujo de trabajo comienza con el entrenamiento de los modelos locales por parte de los nodos propietarios de datos. Por normal general, estos nodos comparten la misma arquitectura del modelo que están entrenando, si bien los hiperparámetros del modelo pueden diferenciar entre los diferentes nodos. Los elementos clave del flujo de trabajo son los siguientes:

- Datos descentralizados: Los datos están distribuídos entre los diferentes nodos propietarios de datos, o clientes, en lugar de estar agrupados en un servidor central. Dado que no se comparten los datos de los clientes, se preserva la privacidad de sus datos durante el entrenamiento del modelo, siendo estos inaccesibles para el resto de los nodos. Las posibles distribuciones de datos que se pueden dar son:

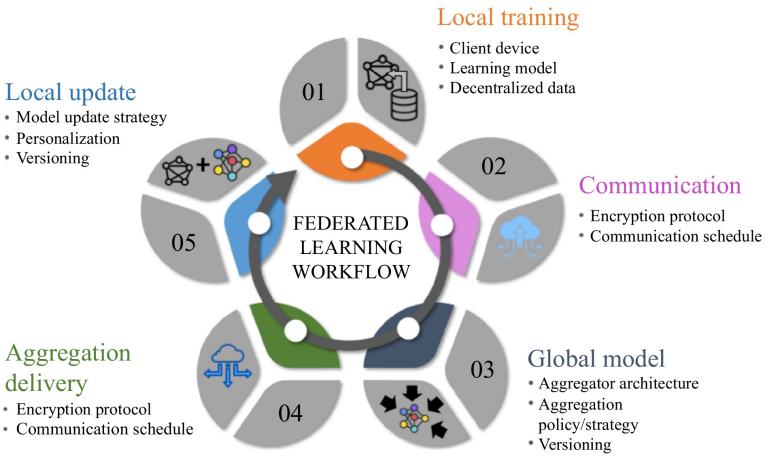


Figura 1.1: Flujo de trabajo de AF.

- Homogénea o Independiente e Idénticamente Distribuida (Independent and Identically Distributed (IID)): Los clientes comparten las características del problema, y además disponen del mismo número de clases e instancias.
- Heterogénea o no independiente e idénticamente distribuida (Non Independent and Identically Distributed (non-IID)): La distribución es diferente entre los clientes. Formalmente se pueden definir tres tipos de distribución heterogénea: (1) cuando el espacio de características es diferente entre los clientes, pero comparten el mismo objetivo; (2) cuando los clientes comparten el espacio de características pero difieren en el espacio de las clases; (3) cuando hay diferencias tanto en el espacio de características como en el espacio de las clases.

- Modelo de aprendizaje: El entrenamiento del modelo se realiza de forma descentralizada, a través de los diferentes clientes participantes. Tras terminar el entrenamiento del modelo local, los clientes compartirán este modelo con el servidor, por ejemplo, en redes neuronales se comparten los pesos de la red. Este proceso mejora el rendimiento del modelo y ayuda

a generalizar su aprendizaje gracias al acceso a una mayor cantidad de datos.

- Clientes: Estos son los nodos propietarios de datos, que además entrenan los modelos locales. Normalmente son llamados clientes.

- Servidor: Este nodo es el nodo de agregación, que se encarga de la agregación de los modelos locales en uno solo, modelo global, mediante el uso de un operador de agregación. Este nodo no suele participar en el entrenamiento del modelo y no suele disponer de datos, por lo que no juega un rol en el entrenamiento del modelo local, pero sí es importante en la generación del modelo global. También se encarga de orquestar el entrenamiento, indicando en cada momento en qué fase del entrenamiento se encuentran.

Comunicación Tras el entrenamiento de los distintos modelos locales, la comunicación entre los clientes y el servidor habilita la agregación de los modelos. Los clientes deben mandar al servidor los modelos entrenados localmente para que el servidor realiza la agregación de estos. La comunicación juega un rol clave en la protección de la privacidad y la seguridad de los datos cuando se empareja con técnicas de seguridad como la Privacidad Diferencial (PD) o *Secure Multiparty Computation (SMC)*. Se pueden destacar los siguientes elementos clave de este paso:

- Organización de la comunicación: La comunicación cliente-servidor puede ser síncrona o asíncrona, dependiendo de la configuración. El servidor es el que se encarga de recoger los modelos locales de los clientes.

- Protocolos de privacidad: Aunque no se compartan datos durante la comunicación en AF, la información compartida puede ser susceptible de sufrir filtraciones de privacidad o corromper todo el proceso de aprendizaje [RBJLL$^+$23]. Las comunicaciones son, por tanto, puntos débiles del AF ya que son susceptibles de ataques. Por esta razón, se suelen combinar con mecanismos de privacidad.

Agregación En este paso se genera el modelo global a través de la combinación de los modelos locales recibidos por los clientes, utilizando un mecanismo de agregación. Este mecanismo de agregación es clave, y varía según el modelo y la tarea que se está afrontando. El agregador más común en AF es *Federated Averaging (FedAvg)* [MMR+17], que se utiliza en modelos que se pueden expresar como un vector de parámetros. En otros casos, como por ejemplo en AD, los cuales no se pueden expresar como un vector de parámetros, es necesario diseñar un agregador que sea capaz de combinar los modelos de cada cliente.

Actualización local El último paso consiste en la actualización de los modelos locales de los clientes que han participado en el entrenamiento del modelo. En este paso el servidor envía a estos clientes el modelo global. En el mejor caso, los clientes simplemente actualizan su modelo local por el global. Sin embargo, se pueden dar diferentes estrategias de actualización que consisten en la combinación del modelo local y el global en lugar de reemplazarlos directamente.

1.1.2. Arquitecturas del Aprendizaje Federado

La combinación de los elementos clave explicados anteriormente generan múltiples arquitecturas federadas, que definen su relación [YLC+19]. Las posibles arquitecturas son:

- **Arquitectura cliente-servidor**: En esta arquitectura el servidor es el responsable de la coordinación y la agregación de los modelos locales, mientras que los clientes son los propietarios de datos y los responsables de entrenar los modelos locales. Esta arquitectura requiere de un alto nivel de confianza en el servidor. Esta es su principal debilidad, ya que es vulnerable ante posibles ataques. Se muestra un ejemplo de la arquitectura en la Figura 1.2

- **Arquitectura *peer-to-peer***: Todos los nodos actúan como clientes y como servidores, es decir, poseen datos de entrenamiento y agregan los modelos de otros nodos. No es necesario un coordinador del proceso de aprendizaje. Esta arquitectura

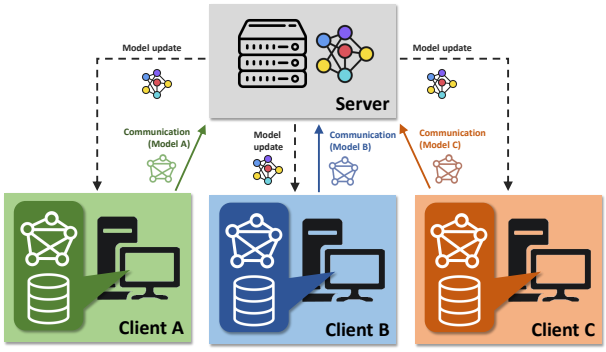


Figura 1.2: Representación de una arquitectura cliente-servidor con 3 clientes.

es compleja, y los costes de comunicación aumentan considerablemente, pero a cambio se obtiene un mayor nivel de seguridad y privacidad de los datos. Un ejemplo de este tipo de arquitectura se muestra en la Figura 1.3.

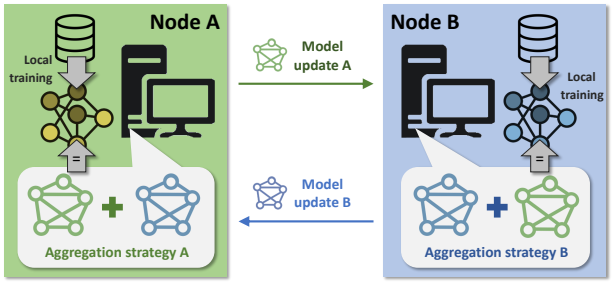


Figura 1.3: Representación de una arquitectura *peer-to-peer* con 2 nodos.

La arquitectura cliente-servidor es la más común en AF, por lo que a partir de este punto será la arquitectura por defecto cuando se hable de AF.

1.1.3. Categorías del Aprendizaje Federado

Según las propiedades de los elementos clave, en AF se pueden presentar diferentes categorías. Se considera que las siguientes propiedades de los elementos clave de los datos descentralizados son las que generan las categorías de AF más relevantes:

Características de los datos, etiqueta y espacio muestral La naturaleza descentralizada del AF puede introducir sesgos y heterogeneidad en la distribución de los datos locales debido a diferentes factores, como el cultural, la etnia o la diferencia de edades entre los usuarios que generan los datos. En base a cómo están particionados los datos a través de los clientes, se dan diferentes categorías [YLC+19]. Estas categorías se definen en términos de espacio de características (X), espacio de etiquetas (Y) y espacio de instancias (I), de la siguiente manera:

- Aprendizaje Federado Horizontal (AFH): Los datos están particionados entre los clientes a nivel de muestra, esto es, que cada cliente tendrá instancias de datos diferentes del resto. Formalmente se puede definir como:

$$X_i = X_j,\ Y_i = Y_j,\ I_i \neq I_j,\ \forall D_i, D_j,\ i \neq j \tag{1.2}$$

  donde el espacio de características y etiquetas de los clientes (i, i) se representa por (X_i, Y_i) y (X_j, Y_k) y se asume que son iguales, mientras que las instancias I_i e I_j no son iguales. D_i y D_j representan los datos de los clientes i y j. Esta categoría es adecuada para el entrenamiento de modelos en entornos donde los nodos disponen de un numero similar de datos, como los dispositivos móviles.

- Aprendizaje Federado Vertical (AFV): La partición de datos entre los clientes se da a nivel de características, por lo que los clientes compartirán las instancias del problema, pero un el espacio de características. Se puede definir como:

$$X_i \neq X_j,\ Y_i \neq Y_j,\ I_i = I_j,\ \forall D_i, D_j,\ i \neq j. \tag{1.3}$$

  Esta categoría es adecuada cuando hay un bajo número de nodos con diferente espacio de características. Un caso de uso

puede ser el de la Salud, donde diversos hospitales o centros médicos pueden disponer de diferentes datos de las mismas personas.

- Aprendizaje Federado de Transferencia (AFT): Los datos están distribuídos entre los nodos sin ningún tipo de solapamiento entre instancias o características. Se puede definir como:

$$X_i \neq X_j,\ Y_i \neq Y_j,\ I_i \neq I_j,\ \forall D_i, D_j,\ i \neq j. \tag{1.4}$$

En esta categoría los datos de entrenamiento y validación son distintos y están definidos en distinto espacio de características.

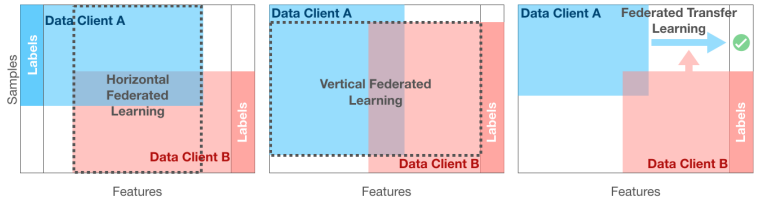


Figura 1.4: Representación de AFV, AFH y AFT en una arquitectura cliente-servidor [YLC+19].

La Figura 1.4 muestra las diferencias entre las distintas categorías AFH, AFV y AFT, dependiendo de los datos y las características compartidos entre los clientes.

1.1.4. Privacidad de los datos: Enfoques avanzados

En AF los clientes mantienen sus datos privados durante el entrenamiento del modelo. Sin embargo, es posible romper dicha privacidad a través del intercambio de modelos en el proceso de aprendizaje, puesto que los modelos locales tienden a memorizar el entrenamiento local. Un nodo malicioso puede intentar recuperar parte de los datos privados de otro cliente, generando una

filtración de privacidad. Por tanto, las técnicas de privacidad de datos son necesarias para asegurar la privacidad de un modelo de AF. Estas técnicas pueden ser empleadas en múltiples elementos de una arquitectura federada y se presentan de manera concisa a continuación.

Secure Multiparty Computation & Homomorphic Encryption SMC busca la seguridad de las comunicaciones en las rondas de aprendizaje, sobre todo prestando atención al proceso de agregación. Los canales de comunicación se mantienen seguros gracias a Homomorphic Encryption (HE) [ZLX+20]. SMC suele utilizar HE como una herramienta para asegurar que no hay nodos manipulando los protocolos de comunicación o interceptando los modelos intercambiados. SMC se ocupa de hacer la agregación de los modelos de forma que los datos sensibles, como los parámetros de los modelos, se mantienen ocultos a los nodos que los manipulan [BBG+20]. Mientras que estas técnicas evitan el acceso interno y externo del modelo durante las rondas de aprendizaje, el modelo resultante sigue siendo vulnerable a ataques que extraen la información del propio modelo agregado [RBJLL+23]. Esto motiva el uso de técnicas de privacidad de datos que modifican el proceso de aprendizaje para asegurar que el modelo federado agregado está protegido al igual que los modelos de los clientes.

Privacidad Diferencial La PD es una técnica de privacidad que busca asegurar la indistinguibilidad de los datos utilizados, es decir, oculta la presencia de los individuos. Esto se consigue añadiendo ruido a los datos [DMNS06]. La PD se puede aplicar en AF en dos etapas diferentes: (1) en entrenamiento del modelo local con PD en el cliente, conocida como Privacidad Diferencial Local (PDL) [WZFY20], y (2) en el paso de agregación, creando una versión diferencialmente privada de FedAvg conocida como Privacidad Diferencial Central (PDC). La PDL provee indistinguibilidad a los datos de los clientes, proporcionando una mayor seguridad a costa de perder rendimiento en el modelo, y PDC asegura indistinguibilidad al determinar qué clientes participan o no en el proceso de agregación, garantizando una menor privacidad pero un mayor rendimiento del modelo comparado con PDL [NHDC22].

La privacidad de los datos debe ser abordada en las tareas de AF. Sin embargo, aumentar la privacidad de los datos suele resultar de un menor rendimiento del modelo federado, una compensación que se debe ajustar a cada escenario federado.

1.2. Metodología para diseñar un experimento simulado en un entorno de Aprendizaje Federado.

En esta sección se presentará la metodología de trabajo para simular escenarios de AF dado que no hay datasets realmente federados, es decir, que estén alojados en múltiples dispositivos. En las simulaciones, los clientes no acceden a los datos de otros clientes. En algunas situaciones, este ejemplo de simulación se asemejará mucho a una situación real, en el sentido en el que algún dataset puede haber sido creado a partir de coger datos de múltiples fuentes, por lo que realizar una distribución de datos según estas fuentes se acerca a un esquema federado. La Figura 1.5 muestra el flujo de trabajo que se debe realizar para desarrollar un escenario federado. En esta figura se presta atención a varios aspectos: el problema a afrontar, la distribución de los datos, la selección del modelo, la estrategia de entrenamiento, y los métodos de evaluación.

1.2.1. Escenario de Aprendizaje Federado, problema y distribución de datos

El primer paso en el diseño de un escenario de AF es diferenciar si se está antes un problema de AFH o AFV. Independientemente del escenario, se debe tener en cuenta si el dataset es inherentemente federado o no. Por norma general, los datasets se pueden particionar entre diferentes clientes, en base a las características o a la distribución de datos del mismo, o bien se puede utilizar un dataset inherentemente federado. Un dataset inherentemente federado es aquel que dispone de una característica, identificador de nodo, por el cual se puede federar de forma directa, y que es natural del dataset por cómo

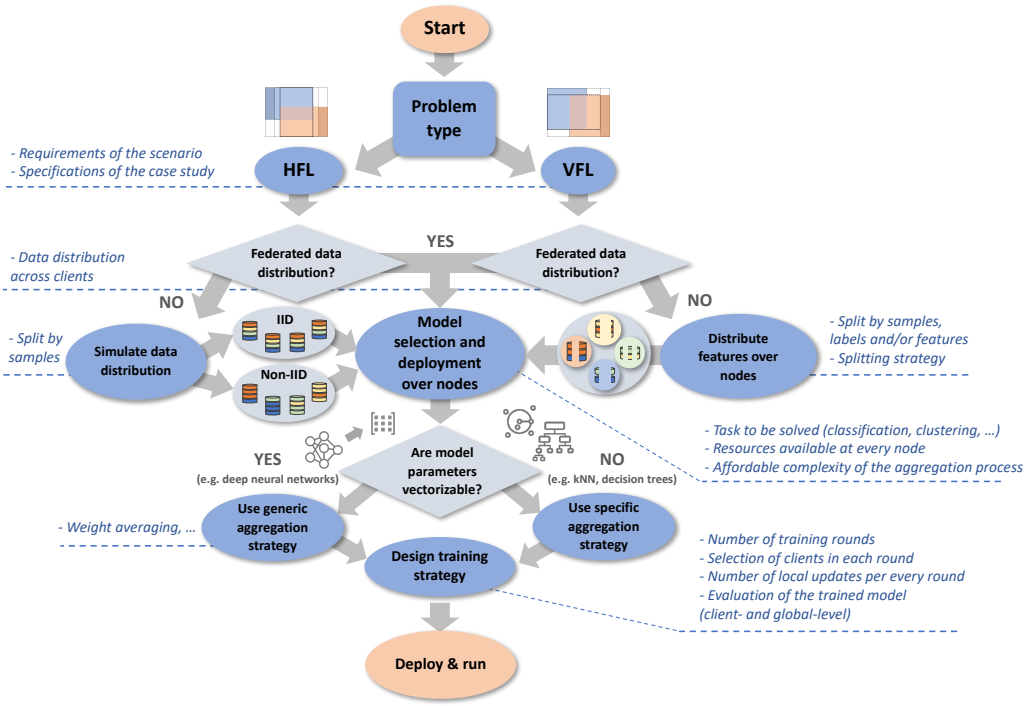


Figura 1.5: Flujo de trabajo para diseñar un escenario federado. Debe leerse de arriba a abajo, comenzando con la elección del tipo de problema, AFH o AFV, y continúa eligiendo la distribución de datos hasta llegar a elegir la estrategia de entrenamiento. Una vez hecho esto, el escenario federado estará preparado para ser desplegado y ejecutado.

se realizó la recolección de datos del mismo. Por el contrario, aquellos datasets que no son inherentemente federados, como lo tradicionales del AA, necesitan ser federados de forma manual para así poder utilizarlos en un experimento federado.

En el escenario AFH, si el dataset no es inherentemente federado, se puede realizar la partición de datos de estos por instancias. En este caso existen dos distribuciones de datos posibles: (1) IID y non-IID. La distribución IID distribuye los datos uniformemente entre los clientes, dando lugar a un escenario poco común en un entorno federado. La distribución non-IID, sin embargo, implica que debe realizarse una distribución en la que los clientes tienen diferentes cantidades de datos, teniendo diversas distribuciones en su espacio de características, además de información sobre clases diferentes en las etiquetas del problema, o incluso, una combinación de estas. En general, un distribución non-IID se encuentra con frecuencia en un entorno federado. En cualquier caso, la distribución de datos debe seleccionarse en función del problema que se va a simular. En el caso de una distribución IID, bastaría con hacer una partición aleatoria de los datos. En cambio, para realizar una simulación de un escenario non-IID, la distribución de datos se realiza asignando determinadas instancias a cada cliente, a veces siguiendo una distribución de Dirichlet, donde el desbalanceo de los datos se realiza mediante un parámetro β [LDCH22]. Una vez realizado el particionado de los datos, se distribuyen entre los clientes, obteniendo cada uno un conjunto de datos diferente y que no será accesible por el resto de clientes.

Al enfrentar un escenario de AFV donde no hay datos inherentemente federados, la partición se hace por las características del problema y no por las instancias de este. En AFV la naturaleza non-IID de los datos nace de no haber solapamiento en la distribución de las características entre los clientes, ni entre el número o tipo de características disponibles en cada cliente [ZXLJ21]. En cualquier caso, una vez se han particionado los datos, cada cliente recibirá el mismo número de instancias, pero cada conjunto tendrá diferentes características, donde la etiqueta de la clase la tendrá solamente un nodo.

Si se enfrenta un escenario de AFT, donde no hay datos inherentemente federados, la partición debe hacerse por características y por instancias del problema, dado que en este escenario será habitual que no se de solapamiento de instancias, y que el solapamiento del espacio de características sea bajo. Una vez particionados los datos, se repartirán entre los clientes, pudiendo estos tener diferente número tanto de instancias como de características.

1.2.2. Selección del modelo

Una vez se han distribuido los datos entre los clientes, el siguiente paso es la selección y despliegue del modelo. Dependiendo de la tarea se pueden seleccionar modelos de aprendizaje supervisado, como AD; aprendizaje no supervisado, como un modelo de *clustering*; o cualquier otro modelo de AA tradicional o de AP. El modelo a elegir en un enfoque federado variará según la tarea a resolver, y las necesidades de esta. Por ejemplo, las CNN han demostrado obtener buenos resultados en tareas de clasificación de imágenes [CLB+21], las Recurrent Neural Networks (RNN) son muy utilizadas en tareas de texto como Análisis de Opiniones (AO) o en series temporales [LZS+20, WCP20]. Otros modelos tradicionales de AA como AD o SVM han sido utilizados en tareas que involucran datasets tabulares como es la predicción del riesgo de conceder un crédito [BMR20]. Cuando no hay muchos datos disponibles, o se da una distribución non-IID de los datos, puede ser interesante utilizar técnicas de *fine-tuning* de modelos pre-entrenados en la tarea a resolver [HCB+21, LKSJ20a]. Como resultado, se pueden entrenar modelos grandes y aprovechar su conocimiento, obteniendo mejores modelos que requerirían de una menor cantidad de recursos si se entrenasen desde cero [LM20].

1.2.3. Estrategias de agregación

Los modelos mencionados previamente tienen diferentes adaptaciones a una configuración de AF, guiada por el proceso de agregación, donde se combinan los modelos locales. Si un modelo puede expresarse como un vector de parámetros, como los modelos de AP, un agregador simple y genérico como FedAvg se puede

elegir. FedAvg realiza la media de los pesos de los modelos agregados de los clientes, donde cada modelo es expresado como un vector de parámetros. Existen variantes más complejas de FedAvg diseñadas para aliviar las dificultades de una distribución non-IID, como los ataques Bizantinos [QCG+24]. Aquellos modelos que no pueden expresarse como vectores de parámetros, como los AD, necesitan un agregador más avanzado y específico para poder entrenar un modelo en un escenario federado. Por norma general, los clientes seleccionan el mismo modelo, y una primera versión se reparte entre estos. Sin embargo, otros enfoques permiten que cada cliente utilice diferentes estructuras, siendo la agregación un *ensemble* de los modelos recogidos [LKSJ20b, GSK+22]. Mientras que un enfoque en el que los clientes entrenan el mismo modelo hace, generalmente, más fácil la agregación de los modelos locales y acelera la convergencia el modelo global, el enfoque de *ensemble* trae problemas en los que si hay pocos clientes participando en el entrenamiento, estos no tendrán el acceso a los mismos recursos, y puede darse que algunos clientes no puedan ejecutar modelos complejos.

1.2.4. Estrategias de entrenamiento y evaluación

El último paso de la metodología consiste en el diseño de la estrategia de evaluación, la cual involucra muchos aspectos como definir el número de rondas de entrenamiento, esto es, el número de comunicaciones cliente-servidor para que converja el modelo global. También se debe de tener en cuenta el número de clientes que participan en cada ronda de entrenamiento, en algunos casos sólo un porcentaje de los clientes participa en cada ronda. Además se deben de tener en cuenta los hiper-parámetros del modelo a entrenar, así como el método de evaluación del modelo seleccionado. La evaluación debe considerarse en dos niveles:

1. A nivel de cliente, se considera un conjunto de métricas para evaluar el modelo federado respecto a las necesidades de cada cliente. Al afrontar potencialmente una distribución non-IID en cada cliente, la evaluación del modelo por parte de estos

no es suficiente para asegurar el rendimiento del modelo federado. Sin embargo, la combinación de estas métricas, junto con otras propiedades estadísticas pueden ayudar en la selección de clientes a agregar en cada ronda [SZHL23].

2. A nivel de servidor, también se considera un conjunto de métricas para evaluar el rendimiento del modelo federado desde una perspectiva más amplia, puesto que considera evaluar la escalabilidad del modelo federado, es decir, el rendimiento del sistema federado.

Es común asumir que todos los clientes quieren participar en el entrenamiento del modelo, implicando que no hay una recompensa para los participantes además del modelo global entrenado. Idealmente, cada cliente que participa en un proceso federado contiene un dataset local único y privado, y espera que se le recompense con un modelo mejor que el que pueda obtener entrenando un modelo local con tan solo sus datos [KMY+16, mel19]. Sin embargo, los clientes, en realidad, tienen la potestad de decidir cuándo participar o no en el proceso federado, basándose en cómo su colaboración va a ser recompensada. En este sentido, desde la perspectiva de la teoría de juegos, es posible dar a cada cliente una recompensa justa por su participación [SPP21].

1.3. Datasets y frameworks de Aprendizaje Federado

Esta sección introduce el ecosistema disponible para diseñar modelos y estudios en AF. Primero se presentarán los datasets más utilizados en la literatura para realizar experimentos federados, y después se mostrarán los *frameworks* disponibles para diseñar estos estudios, analizándolos desde diferentes perspectivas.

1.3.1. Datasets federados

Los datasets tradicionales de AAC se pueden reutilizar para realizar simulaciones mediante técnicas de particionado de datos y distribuyéndolos entre los diferentes nodos que entrenarán el modelo. Sin embargo, hay una gran variedad de datasets que

se consideran plenamente federados por sus características o su distribución de datos. Respecto a ello, hay que mencionar a LEAF [CDW+19], un *framework* de *benchmarking* que provee de múltiples datasets federados, y a *TensorFlow Federated* (TFF) [The18] que también dispone de algunos datasets federados.

La Tabla 1.1 muestra los datasets más comunes de la literatura en AF, incluyendo CelebA [1] , Cifar100 [2], Fashion MNIST [3], FEMNIST [1.3.1], Google landmark v2 [1.3.1], iNaturalist [1.3.1], MedMNIST [4], MNIST [5], Shakespeare [1.3.1], Reddit [1.3.1], Stack Overflow [1.3.1], Sentiment140 [1.3.1], Adult [6], and Credit2 [7]. Los datasets de la tabla cubren diferentes tareas como la Visión por Computador (VC), el Procesamiento del Lenguaje Natural (PLN) y datasets tabulares tradicionales de clasificación, así como los diferentes escenarios federados como AFH, AFV y AFT. Los datasets utilizados para AFV y AFT se han particionado manualmente, pero no son inherentes de AF.

1.3.2. *Frameworks* de Aprendizaje Federado

En la literatura se encuentran múltiples *frameworks* de AF que permiten realizar experimentos en un escenario federado. Se ha realizado una búsqueda de los *frameworks* de código abierto en el estado del arte disponibles, y se han seleccionado diferentes aspectos importantes para comprobar si estos *frameworks* cumplimentan cada uno de estos aspectos o no. La Tabla 1.2 muestra los *frameworks* encontrados y si estos cumplen o no los aspectos importantes para simular un entorno federado [8]. Esta tabla puede ayudar a los usuarios a elegir qué *framework* seleccionar para realizar sus experimentos. Por la limitación de espacio, y dado que se muestra

[1] `https://leaf.cmu.edu/`

[2] `https://www.tensorflow.org/federated/api_docs/python/tff/simulation/datasets`

[3] `https://www.kaggle.com/datasets/zalando-research/fashionmnist`

[4] `https://medmnist.com/`

[5] `http://yann.lecun.com/exdb/mnist/`

[6] `https://archive.ics.uci.edu/ml/datasets/adult`

[7] `https://archive.ics.uci.edu/ml/datasets/default+of+credit+card+clients`

[8] La información en la tabla está actualizada en enero de 2023

Tabla 1.1: Datasets de *benchmarking* utilizados. #Instancias indica el número total de instancias en cada dataset y #Clientes, el número de clientes de un escenario federado, en aquellos casos en los que el dataset es federado por naturaleza. Fed. dist. indica si la distribución del dataset es inherentemente federada o no y Refs. hace referencia a los estudios en los que se ha usado el dataset. Las tareas indicadas en la tabla son: Clasificación de Imágenes (Img), predicción de texto (Txt), análisis de opiniones (AO) y clasificación (cls).

Dataset	Tarea	#Instancias	#Clientes	Fed. dist.	Categoría	Refs.
CelebA	Img	200,288	9,343	Sí	AFH	[RBMCLH22a, QZL+22, ZLL+22]
Cifar100	Img	60,000	-	No	AFH	[CHMS21, CGH+21, HAA20]
Fashion MNIST	Img	70,000	-	No	AFH	[WKNL20, LWS21, RBMCLH22b]
FEMNIST	Img	805,263	3,550	Sí	AFH	[CHMS21, RBMCLH22b, ZDL+21]
Google landmark v2	Img	164,172	1,262	Sí	AFH	[HQB20, CCC22, KS22]
iNaturalist	Img	155,941	9,275	Sí	AFH	[HQB20, MXNV20, RVTM22]
MedMNIST	Img	708,069	-	No	AFH	[KZPP+21, MKT22, KT21]
MNIST	Img	70,000	-	No	AFH	[WKNL20, LWS21, ZDL+21]
Shakespeare	Txt	4,226,150	1,129	Sí	AFH	[CGH+21, MXNV20, AKL21]
Reddit	Txt	56,587,343	1,660,820	Sí	AFH	[BVH+20, LDS+22, CCA+21]
Stack Overflow	Txt	168,895,995	585,323	Sí	AFH	[CGH+21, AKL21, SSG+21]
Sentiment140	AO	1,600,498	660,120	Sí	AFH	[CHMS21, ZDL+21, MXNV20]
Adult	Cls	48,842	-	No	AFV	[LHL+22a, QWW+22, CSP+21]
Credit2	Cls	30,000	-	No	AFV /AFT	[CZG+21, WCX+20, LKX+20]

un número largo de *frameworks*, los nombres de los *frameworks* se han recortado. Los *frameworks* revisados son: PySyft, TensorFlow Federated TFF, FATE, PaddleFL, Flower, Xaynet, IBM FL, Substra, OpenFL, FedML, FedJax, ackdoors101, FedLab, FLSim, easyFL, TorchFL, APPFL, NVFlare

Grados de cumplimiento de cada *framework* Se han considerado tres grados de cumplimiento dependiendo de qué aspectos soportan cada *framework* o no. Los círculos verdes indican que el *framework* soporta el aspecto en su totalidad. El círculo naranja indica que el aspecto está soportado parcialmente, es decir, cubre algunos casos. Finalmente, el círculo rojo indica que el *framework* no soporta el aspecto. Adicionalmente, los círculos grises indican que no se ha podido determinar con exactitud si el *framework* soporta el aspecto.

Aspectos de AF soportados por cada *framework* La Tabla 1.2 está particionada horizontalmente en 4 grupos. El primer grupo recoge los aspectos principales de AF, indicando si los *frameworks* soportan o no la ejecución de modelos propios de AFH, AFV o AFT, si soportan *frameworks* de AA o AP como PyTorch,

Tabla 1.2: *Frameworks* del estado del arte de AF. (●: Soporte total; ●: Soporte parcial; ●: no soportado; -: indeterminado).

	PySyft	TFF	FATE	PadFL	Flower	Xaynet	IBM	Subs	OpenFL	FML	FedJax	back101
Aprendizaje Federado:												
Horizontal Federated Learning	●	●	●	●	●	●	●	●	●	●	●	●
Vertical Federated Learning	●	●	●	●	●	●	●	●	●	●	●	●
Federated Transfer Learning	●	●	●	●	●	●	●	●	●	●	●	●
Support other ML frameworks	●	●	●	●	●	●	●	●	●	●	●	●
Sampling IID or non-IID distribution	●	●	●	-	●	●	●	●	●	●	●	●
Federated aggregation mechanisms	●	●	●	●	●	●	●	●	●	●	●	-
Adversarial attacks in FL:												
Privacy attacks	●	●	●	●	●	●	●	●	●	●	●	●
Defenses against Privacy attacks	●	●	●	●	●	●	●	●	●	●	●	●
Attacks to the federated model	●	●	●	●	●	●	●	●	●	●	●	●
Defenses against attacks to the model	●	●	●	●	●	●	●	●	●	●	●	●
Differential Privacy (DP):												
Mechanisms: Exponential, Laplacian...	●	●	●	●	●	●	●	●	●	●	●	●
Subsampling methods to increase privacy	●	●	●	●	●	●	●	●	●	●	●	●
Advanced DP Composition	●	●	●	●	●	●	●	●	●	●	●	●
Advanced properties:												
Interpretability / Explainability	●	●	●	●	●	●	●	●	●	●	●	●
Personalization	●	●	●	●	●	●	●	●	●	●	●	●
Documentation and tutorials	●	●	●	●	●	●	●	●	●	●	●	●
High-level API	●	●	●	-	●	●	●	●	●	●	●	●
Ability to extend the framework	●	●	●	-	●	-	-	●	●	●	●	●
Actively maintained	●	●	●	●	●	●	●	●	●	●	●	●

TensorFlow, o Scikit/Learn. También si soportan distribuciones de datos IID y non-IID, y si incluyen una variedad de mecanismos de agregación de la literatura. El segundo grupo indica si los *frameworks* soportan ataques y defensas a los modelos durante el entrenamiento federado o no. El tercer grupo indica si el *framework* incluye mecanismos de PD o no. Finalmente, el cuarto grupo indica otras propiedades avanzadas de los *frameworks*, como por ejemplo si dan soporte a la interpretabilidad de los modelos. También se incluye si soportan técnicas de personalización a nivel de cliente o si ofrecen una documentación detallada del funcionamiento del *framework*, así como la facilidad de añadir nuevas propiedades cada *framework*, o si se siguen manteniendo o no.

1.4. Casos de uso de Aprendizaje Federado

En esta sección se van a mostrar, desde un punto de vista práctico, diferentes casos de uso que abarcan múltiples problemas de AA tradicional para demostrar el proceso de cómo resolverlos siguiendo la metodología propuesta en la Sección 1.2. Se utilizarán los *frameworks* y datasets mostrados en la Sección 1.3. Esta sección queda dividida como sigue: (1) la Sección 1.4.1 muestra el Caso de Uso 1, que demuestra cómo resolver un problema de clasificación de imágenes utilizando modelos de AP en un entorno de AFH; (2) la Sección 1.4.2 muestra el Caso de Uso 2, que muestra cómo resolver una tarea de AO aplicando modelos de AP; (3) la Sección 1.4.3 muestra el Caso de Uso 3, donde se presenta cómo aplicar técnicas de PD en problemas de clasificación de imágenes con AP en un entorno de AFH; (4) la Sección 1.4.4 muestra el Caso de Uso 4, en el que se muestra cómo entrenar un modelo de aprendizaje no supervisado, entrenando un modelo de *clustering* en un entorno de AFV; y, (5) la Sección 1.4.5 muestra el Caso de Uso 5, donde se enseña cómo realizar una simulación de un experimento para entrenar un AD en AFV.

1.4.1. Caso de uso 1: Clasificación de imágenes con aprendizaje profundo en AFH

En esta sección se muestra el caso de uso 1, que consiste en el entrenamiento de un modelo de AP para la clasificación de imágenes. Se utilizarán tres *frameworks* diferentes para resolver la tarea, que son: (1) TFF, (2) Flower, (3) FATE.. Para resolver el caso de uso, se va a diseñar el entrenamiento del modelo siguiendo la metodología explicada, y que se observa en la Figura 1.5:

- *Tipo de problema*: En este caso de uso el problema a resolver es en un entorno de AFH, donde los clientes comparten espacio de características pero no espacio de instancias.
- *Distribución federada*: Se consideran dos distribuciones de datos diferentes. La primera, la distribución non-IID, es inherente por el dataset que se va a utilizar. La segunda, la distribución IID, se logra agrupando el dataset distribuído en uno solo, y dividiéndolo de manera artificial para que todos los clientes tengan el mismo número de instancias y de clases.
- *Selección del modelo*: Se utilizan dos redes neuronales diferentes: La primera incluye una capa CNN, dado que es una capa muy utilizada en clasificación de imágenes [CLB+21], y la segunda es una red que sólo consta de capas densas, la cual es menos compleja.
- *Estrategia de agregación*: Como una red neuronal se puede expresar como un vector de parámetros, se utilizará el agregador FedAvg.
- *Estrategias de entrenamiento y evaluación*: Para hacer la simulación lo más parecida a la que se daría en el mundo real, la evaluación se realizará sobre el conjunto de evaluación local de cada cliente, y se mostrará la media de las métricas obtenidas por cada cliente.

Primero se va a demostrar el proceso de resolver el caso de uso utilizando TFF. En TFF, hay dos formas de cargar los datos: (1) cargar los datasets propios de TFF, los cuales están directamente federados en una distribución non-IID, o (2) cargar cualquier dataset y distribuirlo entre los clientes de la forma preferida. En

Código 1, el dataset MNIST viene particionado en una distribución de datos non-IID, al ser propio de TFF; mientras que en Código 2, se carga el dataset MNIST propio de centralizado utilizando la librería *tensorflow datasets*, y transformándolo para que éste siga una distribución IID. Tras cargar los datos, se deben preprocesar para poder ser utilizados por el modelo.

```
mnist_train, mnist_test = emnist.load_data(only_digits=True)
```

Código 1: Caso de Uso 1: clasificación de imágenes usando AP en AFH con TFF. Carga de datos del dataset MNIST con TFF.

```
# Load MNIST from tfds (train partition)
mnist_train = tfds.load('mnist', split='train')

# Transform the data to a dataframe
mnist_train_df = tfds.as_dataframe(mnist_train)

# Random list of ids to add to the dataframe
ids_train = ... # Random values in [0, NCLIENTS-1]
mnist_train_df['id'] = ids_train
```

Código 2: Caso de Uso 1: clasificación de imágenes usando AP en AFH con TFF. Carga de datos desde otras fuentes.

Para definir los modelos, se ha creado un método que devuelve un modelo de *Keras*. En Código 3 se muestra cómo crear una CNN, el cual será similar para crear otras redes. Además del método que crea la red a utilizar, es necesario crear un método que incluya aquellos parámetros de la red, como la función de pérdida o las métricas de evaluación. En este caso se utilizará la función de pérdida *Categorical Cross Entropy*, y como métrica se utilizará el *accuracy*.

Para entrenar el modelo utilizando TFF, primero se debe definir la estrategia de entrenamiento. Dado que el modelo se puede expresar como un vector de parámetros, se utilizará un agregador genérico como FedAvg. En este caso todos los clientes tendrán la misma importancia en el proceso de aprendizaje. Sin embargo, si se utiliza un agregador *FedAvg* con pesos, la contribución de cada cliente sobre el modelo local podrá variar en función de los datos locales disponibles. Como optimizador se ha utilizado *Adam*; hay que tener en cuenta que se pueden utilizar múltiples tasas de

```
def create_keras_CNN():
  model = models.Sequential([
    layers.Reshape((28, 28, 1), input_shape=(28, 28)),
    layers.Conv2D(32, kernel_size=(5, 5), activation="relu",
    ↪  padding="same", strides=1),
    layers.MaxPooling2D(pool_size=2, strides=2, padding='valid'),
    layers.Flatten(),
    layers.Dense(10, activation="softmax"),
  ])
  return model

def model_fn():
  keras_model = create_keras_CNN()

  return tff.learning.from_keras_model(
    keras_model,
    input_spec=train_data[0].element_spec,
    loss=losses.SparseCategoricalCrossentropy(),
    metrics=[metrics.SparseCategoricalAccuracy()]
  )
```

Código 3: Caso de Uso 1: clasificación de imágenes usando AP en AFH con TFF Creación del modelo CNN.

aprendizaje. Una vez se inicializan los modelos, el modelo colaborativo se entrena durante varias rondas. La elección del número de rondas vendrá dada por los requisitos del problema, así como por las restricciones de los dispositivos. Este proceso se presenta en Código 4.

```
# Define training process
training_process = build_unweighted_fed_avg(
  model_fn,
  client_optimizer_fn=lambda: optimizers.Adam(learning_rate=0.001),
  server_optimizer_fn=lambda: optimizers.Adam(learning_rate=0.01)
)

train_state = training_process.initialize()
for round_num in range(1, NROUNDS+1):
  result = training_process.next(train_state, train_data)
```

Código 4: Caso de Uso 1: clasificación de imágenes usando AP en AFH con TFF. Entrenamiento en el escenario federado.

Por su parte, Flower no proporciona ni la distribución de datos non-IID ni la forma de crear este tipo de distribuciones. Dado que el dataset MNIST no está directamente federado, se cargará utilizando un paquete de *PyTorch torchvision*, y después se realizará la partición de datos IID entre los diferentes clientes,

siguiendo una distribución aleatoria uniforme (Código 5).

```
# Load train from torchvision and transform to tensor
mnist_train = MNIST("./dataset", train=True,
↪  transform=transforms.ToTensor())

# Split into NCLIENTS partitions to simulate the individual datasets
train_lengths = [len(mnist_train) // NCLIENTS] * NCLIENTS
train_splits = random_split(mnist_train, train_lengths)

# Create DataLoaders for each client
train_data = []
for i in range(NCLIENTS):
    train_data.append(DataLoader(train_splits[i]))
```

Código 5: Caso de Uso 1: clasificación de imágenes usando AP en AFH con Flower. Carga del dataset MNIST.

Como se ha mencionado previamente, Flower permite utilizar modelos de *Keras* y de *Pytorch*. Para mostrar una alternativa al caso de uso previo, se utilizará un modelo de *Pytorch*. En Código 6, se muestra un ejemplo para crear una CNN. Pese a que el código difiere del utilizado con TFF, la red es la misma. Para crear otras redes, se deberá seguir un proceso similar al mostrado.

```
class CNN_Net(nn.Module):
   def __init__(self) -> None:
      super(CNN_Net, self).__init__()
      self.cnn1 = nn.Conv2d(in_channels=1, out_channels=32, kernel_size=5,
      ↪  stride=1, padding=2)
      self.relu1 = nn.ReLU()
      self.maxpool1 = nn.MaxPool2d(kernel_size=2)
      self.fc1 = nn.Linear(32 * 14 * 14, 10)

   def forward(self, x: torch.Tensor) -> torch.Tensor:
      out = self.cnn1(x)
      out = self.relu1(out)
      out = self.maxpool1(out)
      out = self.fc1(out.view(out.size(0), -1))
      return out
```

Código 6: Caso de Uso 1: clasificación de imágenes usando AP en AFH con Flower. Creación del modelo CNN.

En Flower, el usuario debe definir los métodos de entrenamiento y evaluación (el mismo que para un escenario centralizado). Para adaptar estos métodos a una simulación federado, primero se

debe definir el *FlowerClient*, para simular el comportamiento de cada cliente. Para permitir que Flower cree los clientes y distribuya los datos, es necesario crear este método como se muestra en Código 7.

```
def client_fn(cid: str) -> FlowerClient:
    # Load model
    net = CNN_Net().to(DEVICE)

    # Note: each client gets a different train/test data
    trainloader = train_data[int(cid)]
    testloader = test_data[int(cid)]

    # an FLower client represents a single participant
    return FlowerClient(net, trainloader, testloader)
```

Código 7: Caso de Uso 1: clasificación de imágenes usando AP en AFH con Flower. Método para crear instancias de clientes.

Se utilizará la implementación de FedAvg de Flower para entrenar el modelo en el escenario federado. Las métricas se pueden reportar si el usuario lo define en la estrategia, como se ha mostrado aquí. Para empezar la simulación, se debe indicar el método para crear los clientes, el número de clientes que entrenarán el modelo, la configuración del servidor (incluyendo el número de rondas), y la estrategia a seguir, como se presenta en Código 8.

```
fl_sim = fl.simulation.start_simulation(
    client_fn=client_fn,
    num_clients=NCLIENTS,
    config=fl.server.ServerConfig(num_rounds=NROUNDS),
    strategy=fl.server.strategy.FedAvg(),
)
```

Código 8: Caso de Uso 1: clasificación de imágenes usando AP en AFH con Flower. Entrenamiento del modelo en el escenario federado.

Por último se muestra cómo resolver este caso de uso utilizando FATE. El Pese a que la experimentación en FATE se realice en una sóla máquina, el usuario debe preparar la simulación de los diferentes clientes. Por ello, a cada cliente se le asigna un id diferente para simular un escenario real, y se le asigna el rol de

guest o de *host*. En este caso se pueden configurar varios *hosts*, y en este ejemplo se define el primer cliente como *guest*, y el resto como *hosts*. Además de estos roles, está el rol de árbitro, el cual orquesta el proceso de aprendizaje. FATE se basa en la creación de una *pipeline*, por lo que una vez creada, se le indica el id de cada participante (Código 9).

```
guest = 10000
hosts = [10001, 10002, 10003, 10004]
arbiter = 9999

# Initialize the pipeline
pipeline = PipeLine()
# Set participants information
pipeline.set_roles(
  guest=guest,
  host=hosts,
  arbiter=arbiter
)
```

Código 9: Caso de Uso 1: clasificación de imágenes usando AP en AFH con FATE. Creación de la *pipeline* y fijación de roles.

FATE no proporciona el dataset MNIST, pero permite que se cargue de forma externa. Para ello, el usuario debe configurar el componente *Reader*, como se muestra en Código 10, en el que se indica la ruta a los datos. En este caso, el dataset MNIST se ha descargado localmente, y se han dividido en varios directorios para cada cliente, además de las particiones de entrenamiento/validación.

Hay que tener en cuenta que FATE sólo permite el uso de redes *PyTorch* con capas densas. En este caso, Código 11 demuestra la implementación de una arquitectura con capas densas en FATE, pero no incluye capas CNN. Esta red es más simple que las anteriores.

Para definir el proceso de entrenamiento, se utiliza la clase *HomoNN*, Código 12. Se incluye la definición del modelo a utilizar, la función de pérdida, optimizador, dataset y estrategia de entrenamiento. Los parámetros son los mismos que los utilizados en TFF y Flower.

```
# Create table with data for client 0
train_datas[0] = {
  "name": "mnist_train_0",
  "namespace": "experiment"
}

# Bind the data into the pipeline
pipeline.bind_table(name=train_datas[0], path='/path/to/data/c0_train')

# Create Reader for training data
reader_train = Reader(name="reader_train")

# Set the data of each client
reader_train.get_party_instance(role='guest',
↪  party_id=guest).component_param(table=train_datas[0])

# Repeat the same process for the rest of clients
```

Código 10: Caso de Uso 1: clasificación de imágenes usando AP en AFH con FATE. Creación del componente Reader.

```
model = t.nn.Sequential(
   t.nn.Linear(784, 32),
   t.nn.ReLU(),
   t.nn.Linear(32, 10),
   t.nn.Softmax(dim=1)
)
```

Código 11: Caso de Uso 1: clasificación de imágenes usando AP en AFH con FATE. Creación de la red neuronal densa.

Tras crear los componentes, se deben añadir a la *pipeline* como se muestra en Código 13. El proceso de entrenamiento empieza una vez que se compila la *pipeline* y se llama al método *fit*.

Una vez se ha descrito cómo resolver los problemas con los diferentes *frameworks*, se muestra los resultados obtenidos en la Tabla 1.3. Los resultados muestran el *loss* y el *accuracy* del entrenamiento y de la validación de los modelos, así como el tiempo de ejecución de cada uno. Además, FATE no reporta el valor del *loss*. A día de hoy, FATE no es capaz de soportar más de 4 *hosts*, por lo que se utilizaron 5 clientes en lugar de 10, como en el resto de *frameworks*.

Se puede observar que el *accuracy* entre los diferentes *frameworks* es similar, y que las diferencias pueden venir por la implementación de los modelos. Por otra parte, los resultados de TFF

```
nn_0 = HomoNN(
   name='nn_0',
   model=model,
   loss=t.nn.CrossEntropyLoss(),
   optimizer=t.optim.Adam(model.parameters(), lr=0.001),
   dataset=dataset_param,
   trainer=TrainerParam(
     trainer_name='fedavg_trainer',
     epochs=NEPOCHS * NROUNDS,
     aggregate_every_n_epoch=NEPOCHS,
     batch_size=BATCH_SIZE
  )
)
```

Código 12: Caso de Uso 1: clasificación de imágenes usando AP en AFH con FATE. Configuración del entrenamiento federado.

```
pipeline = PipeLine()

# Add all the corresponding components
pipeline.add_component(reader_train)
pipeline.add_component(
    nn_0,
    data=Data(train_data=reader_train.output.data)
)
# ...

# Compile the pipeline and train
pipeline.compile()
pipeline.fit()
```

Código 13: Caso de Uso 1: clasificación de imágenes usando AP en AFH con FATE. Configuración de la *pipeline* y entrenamiento del modelo.

y Flower muestran que la arquitectura convolucional tiene mejor rendimiento que la red de solo capas densas. De hecho, sólo se han usado 10 rondas globales de entrenamiento para obtener los resultados, por con más rondas se podría obtener un mejor rendimiento. De acuerdo a los tiempos de ejecución, no hay diferencias entre TFF y Flower. Mientras que la red densa de TFF es más rápida que la de Flower, la CNN de Flower es más rápida que la CNN de TFF. FATE es más lento que el resto de los *frameworks*. Por los resultados obtenidos, FATE no debería ser considerada para resolver esta tarea, y la elección entre TFF y Flower quedará en manos del usuario, y de si es experto ya de un *framework* o no.

Tabla 1.3: Resultados del caso de uso 1 en la tarea de clasificación de imágenes usando AP en un escenario de AFH para los tres *frameworks* y dos arquitecturas. Los valores N/A indican que o bien no se ha podido construir la CNN en FATE, o bien no se ha podido obtener el valor de *loss* en validación para FATE.

		TFF	Flower	FATE
Dense	Train loss	0.192	0.002	1.489
	Train accuracy	0.947	0.991	0.968
	Test loss	0.276	0.006	N/A
	Test accuracy	0.922	0.965	0.947
	Runtime (s)	53.6	198.5	1516.7
CNN	Train loss	0.043	0.000	N/A
	Train accuracy	0.989	1.000	N/A
	Test loss	0.090	0.002	N/A
	Test accuracy	0.973	0.987	N/A
	Runtime (s)	749.5	337.1	N/A

1.4.2. Caso de uso 2: Análisis de opiniones con Aprendizaje Profundo en AFH

Los problemas de PLN también copan los tópicos de la literatura en el AA, sobre todo gracias al crecimiento de los modelos de AP en la última década. En este caso de uso se muestra cómo resolver una tarea de AO utilizando el dataset *Sentiment140*, usando una red pre-entrenada en lugar de construir una desde cero. Los pasos de la metodología propuesta son:

- *Tipo de problema*: En este caso el problema a resolver se da en un entorno de AFH. Tras haber analizado las limitaciones de FATE, donde sólo se pueden crear redes densas, se utilizarán TFF y Flower para resolver el problema.
- *Distribución federada*: Se ha realizado una simulación de una distribución IID. Aunque el dataset sea inherentemente federado, es un dataset muy largo, y se ha utilizado un subconjunto de este para poder entrenar los modelos, por lo que la distribución de datos es manual.
- *Selección del modelo*: Se va a utilizar un modelo pre-

entrenado que está creado para tareas de clasificación de texto, y sobre el cual se aplicarán técnicas de *fine-tuning*. Esto ayudará a que el modelo converja rápidamente, reduciendo así la posible sobrecarga en la comunicación.

- *Estrategia de agregación*: Al igual que en el caso de uso anterior, se utilizará un agregador genérico.

- *Estrategias de entrenamiento y evaluación*: El modelo se evaluará utilizando los datasets privados de los clientes, en los que cada cliente reportará las métricas sobre sus propios datos.

Primero se analiza cómo resolver el caso de uso utilizando TFF, destacando las diferencias con el caso de uso previo. En este caso, los datos se cargan de *tensorflow datasets*, al igual que en Código 2, y el se cargará solo un porcentaje de los datos, como se muestra en Código 14. Las características a utilizar son el texto del *tweet*, así como la polaridad. Los *tweets* se han preprocesado para eliminar los signos de puntuación, o convertir el texto a minúsculas, entre otros. Para realizar las particiones IID, se ha creado una lista de ids aleatorios, y se han distribuido las instancias entre los clientes de forma aleatoria. De esta forma, los clientes no comparten instancias entre sí.

```
# Load train and test partitions of sentiment140 from tfds
sent140 = tfds.load('sentiment140', split=['train[:1%]', 'test[:10%]'])
sent140_train, sent140_test = sent140[0], sent140[1]

# Transform train to dataframe and keep desired columns
sent140_train_df = tfds.as_dataframe(sent140_train)[['text', 'polarity']]
sent140_train_df['text'] = sent140_train_df['text'].apply(lambda x:
↪  text_processing(x))

# Create a random list of ids and set to the dataframe
ids_train = [i for i in range(NCLIENTS) for _ in
↪  range(len(sent140_train)//NCLIENTS)]
random.Random(seed).shuffle(ids_train)
sent140_train_df['id'] = ids_train

# Do the same process for the test data
```

Código 14: Caso de Uso 2: Análisis de Opiniones usando AP en AFH con TFF. Carga, preprocesamiento y distribución IID de un porcentaje de los datos del dataset Sentiment140.

El dataset *Sentiment140* tiene tres clases por defecto: 0 (negativo), 2 (neutral), y 4 (positivo). Dado que en el caso de uso anterior se trató un problema multiclase, en este caso se convertirá en uno binario. Para ello se han eliminado las instancias neutras, y se han asignado las clases positivas al valor 1, como se muestra en Código 15.

```
def delete_neutral_ops(df):
    df = df.loc[df['polarity']!=2]
    df['polarity'] = df['polarity'].replace(4, 1)
    return df

sent140_train_df = delete_neutral_ops(sent140_train_df)
sent140_test_df = delete_neutral_ops(sent140_test_df)
```

Código 15: Caso de Uso 2: Análisis de opiniones usando AP en AFH con TFF. Binarización del problema multiclase.

La definición del modelo se muestra en Código 16. En este caso el método carga un modelo pre-entrenado, con los pesos del modelo congelados durante el entrenamiento. Si el usuario quiere hacer *fine-tuning*, podría hacerlo descongelando los pesos al cargar el modelo, pero esto tomará más tiempo de ejecución. Además de añadir las capas pre-entrenadas, se ha añadido una capa densa y la capa final de predicción. A pesar del modelo elegido, otro modelo de *TensorFlow* podría ser utilizado. En este caso se utiliza como función de pérdida la función *binary cross entropy*, por la naturaleza binaria del problema.

El resto del proceso, incluyendo la configuración de las rondas de entrenamiento y la ejecución de la simulación federada, es igual que la mostrada con TFF en el caso de uso previo.

En este caso de uso, además se muestra cómo resolver la tarea de AO utilizando Flower; sin embargo, apenas existen diferencias con la solución presentada. Los datos se cargan, preprocesan y binarizan al igual que en TFF, Código 14 y 15. En la Sección 1.4.1, se menciona que Flower puede trabajar con *Keras* y con *TensorFlow*, por lo que en este caso se puede utilizar el mismo modelo que para TFF, Código 16. La estrategia de entrenamiento se define de la misma forma que en Código 8.

Una vez se ha descrito cómo afrontar una tarea de AO con TFF y Flower, se presentan los resultados de la experimentación

```
def create_keras_model():
  # Load pretrained model from tfhub
  hub_layer = hub.KerasLayer(
    "https://tfhub.dev/google/...",
    input_shape=[],
    dtype=tf.string,
    trainable=False
  )

  # Create model with fixed pretrained model
  # Also add some dense layers
  model = tf.keras.Sequential()
  model.add(hub_layer)
  model.add(tf.keras.layers.Dense(16, activation='relu'))
  model.add(tf.keras.layers.Dense(1))
  return model


def model_fn():
  keras_model = create_keras_model()

  return tff.learning.models.from_keras_model(
    keras_model,
    input_spec=train_data[0].element_spec,
    loss=losses.BinaryCrossentropy(),
    metrics=[metrics.BinaryAccuracy()]
  )
```

Código 16: Caso de Uso 2: Análisis de opiniones usando AP en AFH con TFF. Creación de un modelo pre-entrenado.

realizada. La Tabla 1.4 presenta los resultados de entrenamiento y validación de *accuracy* y *loss*. Pese a que los datos provengan de la misma fuente, y a que el modelo sea el mismo, y además de que realicen la misma estrategia de entrenamiento, los resultados difieren. Flower sólo necesitó el 40 % del tiempo requerido por TFF para entrenar el modelo, siendo una opción mucho más rápida en este caso. Además, el *accuracy* de validación es mejor en Flower, haciéndolo una mejor opción para resolver AO en un entorno federado.

1.4.3. Caso de uso 3: Privacidad diferencial en AFH

En los casos anteriores no se aumentó la privacidad durante el proceso de entrenamiento. En este caso se muestra como emplear técnicas de PD durante el entrenamiento, aumentando la privacidad de los datos de los clientes. Dado que TFF es el único

Tabla 1.4: Resultados del caso de uso 2: Análisis de opiniones con AP en AFH con TFF y Flower.

	TFF	Flower
Train loss	0.563	0.002
Train accuracy	0.723	0.992
Test loss	0.494	0.006
Test accuracy	0.889	0.966
Runtime (s)	470.2	186.5

framework que incluye técnicas de PD, es el que se usará para mostrar un ejemplo de cómo aplicarla, aprovechando la misma metodología que en el caso de uso 1, Sección 1.4.1. La PD se puede aplicar durante la selección del modelo y durante el despliegue a través de los nodos, o durante el diseño de la estrategia de entrenamiento. La primera opción se refiere a modelos que protegen la privacidad de los clientes y sus datos por su diseño, mientras que la segunda se refiere a mecanismos adicionales para proteger dicha privacidad, siendo independiente del modelo en uso. En la literatura, los mecanismos independientes del modelo son más comunes, ya que pueden aplicarse a una gama más amplia de modelos. TFF sólo dispone de un método de PD disponible, y es el método de PD con recorte adaptativo [ATMR21].

Las diferencias con el caso de uso 1 son escasas, dado que sólo se añade el mecanismo de PD en el proceso de aprendizaje. Para añadir el mecanismo de PD sólo hay que definir el agregador, Código 17, sustituyendo al FedAvg del caso de uso 1. Este agregador recibe como parámetro un multiplicador de ruido gaussiano, así como el número de clientes que se espera que participen en cada ronda. Después el agregador se fija al definir el proceso de entrenamiento federado. En cada ronda, los clientes envían los parámetros de su modelo, añadiendo a estos cierta cantidad de ruido en los valores enviados, para así proteger su información local.

En la Figura 1.6a se muestra la variación del rendimiento en las métricas de validación, tanto en *accuracy* como en *loss*. Por ese motivo, se han hecho pruebas con particiones IID y non-IID, y los resultados son la media de 10 ejecuciones. Los resultados son

```
# Aggregator with Differential Privacy
aggregation_factory = dp_aggregator(noise_mult, NCLIENTS)

# Include aggregator in the training process
training_process = build_unweighted_fed_avg(
   model_fn,
   client_optimizer_fn=lambda: optimizers.Adam(learning_rate=0.001),
   server_optimizer_fn=lambda: optimizers.Adam(learning_rate=0.01),
   model_aggregator=aggregation_factory
)
```

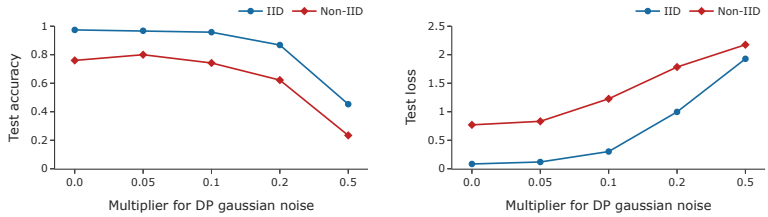
Código 17: Caso de Uso 3: Privacidad diferencial usando AP en AFH con TFF. Proceso de entrenamiento incluyendo mecanismos de PD.

los esperados, a mayor ruido se introduce en las comunicaciones, el rendimiento decrece. Un valor de 0.1 para el ruido gaussiano mantiene el rendimiento del modelo a nivel de *accuracy*, tanto en las distribuciones IID como en las non-IID, mientras se aumenta la privacidad local. Al aumentar el ruido, decae el rendimiento, pero se puede recuperar incrementando el número de rondas de aprendizaje. Consecuentemente, incrementar el multiplicador de ruido por el umbral 0.1, dependerá de los requisitos del problema y de la preocupación de los usuarios involucrados por la privacidad. Además, el tiempo de ejecución se mantiene constante, sin tener en cuenta el nivel de ruido añadido. Finalmente, pese a que en los casos de uso 1 y 2 Flower fuera una mejor opción, se deben tener en cuenta muchos factores a la hora de elegir un *framework*. Si la privacidad de las comunicaciones es un diseño clave a priorizar, entonces la PD es necesaria, por lo que , hasta la fecha, sólo TFF podría utilizarse al ser la única que proporciona esa solución.

1.4.4. Caso de uso 4: Clustering con k-means en AFH

Además de modelos de aprendizaje supervisado, también es posible entrenar modelos de aprendizaje no supervisado que se pueden adaptar a un entorno de AF a través de la metodología de trabajo propuesta:

- *Tipo de problema*: En este caso se trabajará en un entorno de AFH.



(a) *Accuracy* de validación. (b) Testing loss.

Figura 1.6: Resultados de validación del caso de uso 3 mientras aumenta el ruido de la PD.

- *Distribución federada*: Se trabajará con el dataset MNIST, por lo que la distrbución de los datos será IID.
- *Selección del modelo*: TFF ofrece una implementación del modelo k-means para aprendizaje modelo. Dado que el modelo de no se puede expresar como un vector de parámetros, es necesario un agregador específico, y TFF incluye uno para entornos de AF.
- *Estrategia de agregación*: En cada ronda se aplicará el agregador de TFF para k-means, el cual recibe los centroides calculados por cada cliente, así como el número de instancias asociadas a cada centroide. Después, el servidor calculará los nuevos centroides como una combinación con pesos de los centroides de los clientes, donde los pesos se refieren al número de instancias para cada centroide del cliente.
- *Estrategias de entrenamiento y evaluación*: En este caso, para evaluar un modelo de *clustering*, no se utilizad de ningún dato de test, sino que se utilizan métricas internas para asegurar el rendimiento del modelo, como el análisis de los centroides. Hasta la fecha, TFF no dispone muchas opciones para evaluar modelos de *clustering*.

La carga y el particionado de datos se realiza al igual que en Código 2. La creación del modelo se indica en Código 18, donde también se define la estrategia de entrenamiento. El número de centroides en cada cliente se fija al número de clases del dataset.

```
# Create k-means model
fed_process = build_fed_kmeans(
    num_clusters = 10,
    data_shape = (784,)
)

# Initialize and run federated process
fed_state = fed_process.initialize()
for round_num in range(1, NROUNDS+1):
   result = fed_process.next(fed_state, client_data)
```

Código 18: Caso de Uso 4: *clustering* con k-means en AFH usando TFF. Definición del modelo k-means.

Se debe subrayar que, mientras se ofrece un método para k-means, las opciones para manipularlo más allá de la fase de entrenamiento son muy limitadas. De entre las opciones disponibles, se muestra el número de instancias asignadas por cada cliente a cada centroide, o las coordinadas de los centroides, Código 19.

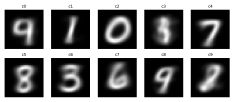
```
# Assignation of data examples to each cluster
print(result.state.finalizer)

# Centroids
for c in result.state.global_model_weights:
   print(c)
```

Código 19: Caso de Uso 4: *clustering* con k-means en AFH usando TFF. Mostranso algunos resultados.

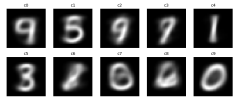
Además de mostrar por pantalla los centroides, se pueden dibujar. En la Figura 1.7 se enseñan los centroides de un k-means centralizado clásico, y se comparan con los de la versión federada que usa 10 clientes. En ambos casos, el algoritmo ha utilizado 10 iteraciones. Se puede observar que la salida es muy similar en ambos casos: la mayoría de los centroides representa un dígito claro, mientras que otros no dejan claro qué dígito representan. Estos centroides comprenden instancias de diferentes clases.

1.4.5. Caso de uso 5: Árboles de decisión en AFV

En esta sección se muestra el caso de uso 5 con uno de los *frameworks* presentados en la Sección 1.3, en el que se realizará



(a) Centroides de un k-means centralizado.



(b) Centroides de un k-means federado con 10 clientes.

Figura 1.7: Caso de Uso 4: Comparación de los centroides de k-means usando una versión centralizada o una federada.

una simulación de un experimento para entrenar un árbol en AF. La Tabla 1.2 muestra las propiedades de los *frameworks*, y los AD están catalogados dentro de las propiedades avanzadas, en concreto con la propiedad de *Interpretabilidad/Explicabilidad*. De entre todos los *frameworks* sólo FATE e IBM FL permiten el entrenamiento de AD en un entorno de AF. En este caso se ha seleccionado el *framework* FATE, ya que, además de poder simular el entrenamiento de un AD de entre los *frameworks* comentados, FATE permite también visualizar el árbol generado, lo que ayuda con la interpretabilidad de los resultados.

FATE permite entrenar un AD en un escenario AFV, es decir, los datos están particionados verticalmente. Para llevar a cabo este caso de uso, se va a diseñar el entrenamiento del modelo siguiendo la metodología explicada, y que puede consultarse en la Figura 1.5:

1. *Tipo de problema*: AFV, puesto que los datos están particionados verticalmente.

2. *Distribución federada*: Al igual que en la mayoría de los casos de AFV, en este caso se trabaja con un dataset tabular, en concreto con el dataset Credit2 para la predicción del riesgo de conceder un crédito. Los datos en este escenario no son inherentes de una distribución federada, como se ha mostrado en la Tabla 1.1, por lo que es necesario hacer una simulación de la partición non-IID. En el caso de FATE, proporciona el dataset ya particionado para poder utilizarlos directamente. La distribución de datos está hecha para dos nodos de forma

artificial, dado que el dataset utilizado no es inherentemente federado.

3. *Selección del modelo*: En este caso se ha optado por elegir un modelo de AD. Los AD se han usado con éxito en problemas a lo largo de la literatura [BMR20]. FATE proporciona el modelo SecureBoost [CFJ+21a] sobre un dataset tabular particionado verticalmente, por lo que se usará este modelo.

4. *Estrategia de agregación*: Dado que los AD no pueden expresarse como vectores de parámetros, requieren de un mecanismo de agregación propio. En este caso, SecureBoost define un esquema de agregación específico en el que los clientes entrenan el modelo global conjuntamente.

5. *Estrategias de entrenamiento y evaluación*: Para evaluar el modelo, cada cliente mantiene una porción de sus datos como conjunto de evaluación; estos datos no están disponibles para el resto de clientes o para el servidor central. En este caso, aunque las instancias de evaluación son las mismas para los diferentes clientes, cada cliente conservará sus características propias de manera privada para cada instancia.

En AFV las características están distribuidas entre los clientes, sin embargo sólo uno de ellos tiene las etiquetas de las instancias. En FATE, el cliente que tiene las etiquetas es considerado como *guest*, mientras que el otro nodo es considerado como *host*. Además, para usar *SecureBoost*, no es necesario utilizar un nodo neutral adicional. En Código 20 se muestra como cargar un CSV con los datos para cada cliente. Cada archivo CSV contiene un número diferente de atributos, los propios de cada nodo, sin que se de un solapamiento de las características del problema entre los nodos. Estos datos se deben alinear antes de poder entrenar el modelo, y esto se hace añadiendo el objeto `Intersection` a la *pipeline* de trabajo (*Código 21*).

Esto permite a los nodos identificar qué instancias van a participar en el entrenamiento sin proporcionar más información que su id. El modelo y la estrategia de entrenamiento se definen en Código 22. Además, en Código 22 se definen los hiper-parámetros del modelo siguientes: el objetivo o la métrica de perdida, el tipo

```
# Path to the FATE repository in the local machine
data_base = "/path/to/data/repository/FATE/"

# Add the data to the FATE framework before reading it
pipeline.add_upload_data(
   file=os.path.join(data_base, "default_credit_hetero_guest.csv"),
   table_name=guest_train_data["name"],
   namespace=guest_train_data["namespace"],
   extend_sid=True
)
# Add also host data

# Upload data
pipeline.upload()

# Define Reader component and add to the pipeline
reader_0 = Reader(name="reader_0")
# Add components to the pipeline
pipeline.add_component(reader_0)
```

Código 20: Árboles de decisión en AF usando FATE. Carga de datos.

```
# Define DataTransform and Intersection components
data_transform_0 = DataTransform(name="data_transform_0",
↪  with_match_id=True)
intersection_0 = Intersection(name="intersection_0")

pipeline.add_component(data_transform_0,
↪  data=Data(data=reader_0.output.data))
pipeline.add_component(intersection_0,
↪  data=Data(data=data_transform_0.output.data))
```

Código 21: Árboles de decisión en AF usando FATE. Alineación de datos por el ID.

de encriptación, y el número de árboles o la profundidad máxima de estos. Entonces, el modelo se añade a la *pipeline* donde, los datos de entrada del modelo son los obtenidos de la operación de `Intersección`.

El último paso para empezar el proceso de entrenamiento consiste en compilar la *pipeline* y entrenar el modelo, como se muestra en Código 23. FATE permite visualizar los resultados de los modelos, y en el caso del modelo de AD, permite visualizar la estructura de los árboles construidos. Hay que tener en cuenta que al analizar los árboles, estos sólo pueden ver la información del *guest* o del *host*, dependiendo de quién esté accediendo al árbol,

```
hetero_secureboost_0 = HeteroSecureBoost(
   name="hetero_secureboost_0",
   objective_param={"objective": "cross_entropy"},
   encrypt_param={"method": "paillier"},
   num_trees=10,
   tree_param={"max_depth": 3}
)

# Add to the pipeline
pipeline.add_component(hetero_secureboost_0,
↪  data=Data(train_data=intersection_0.output.data))
```

Código 22: Árboles de decisión en AF usando FATE. Creación de SecureBoost.

esto es, que el *guest* no puede observar la información proporcionada por el *host* al árbol, y viceversa. La Figura 1.8 muestra la visualización del árbol desde el punto de vista del *host*; esto es, los nodos que se han particionado por sus atributos, mientras que los nodos que se han creado con la información del *guest* permanecen ocultos para el *host*.

```
# Compile the pipeline and train
pipeline.compile()
pipeline.fit()
```

Código 23: Árboles de decisión en AF usando FATE. Entrenando el modelo.

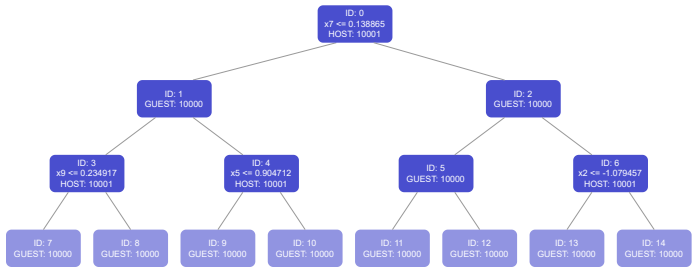


Figura 1.8: AD en AF usando FATE. Vista del *host* del árbol de decisión construido con SecureBoost.

1.5. Conclusiones

En este capítulo se ha mostrado que el AF es un campo crucial para el entrenamiento de modelos de IA en entornos distribuídos donde la privacidad es clave. El AF permite entrenar modelos de IA, aprovechando el conocimiento de todos los nodos involucrados en el entrenamiento, sin compartir ningún tipo de dato de estos nodos. El uso de AF ha aumentado a lo largo de los años debido al gran número de estudios y diferentes *frameworks* para resolver problemas de AF. Las regulaciones recientes de privacidad de datos y de IA, como la del Parlamento Europeo [Act21], enfatizan en la privacidad como un fundamento indispensable para los sistemas de IA fiable. Además de la necesidad de mantener la privacidad de los datos de los nodos al entrenar un modelo, es fundamental que el modelo resultante sea explicable para poder ser utilizado por los diferentes expertos, además de ser uno de los requisitos de un sistema de IA fiable [DRDC+23]. Sin embargo, los *frameworks* mostrados en la Sección 1.3 no facilitan la creación de modelos interpretables, sino la de modelos de caja negra que logran un mayor rendimiento. Se debe lograr un balance entre rendimiento e interpretabilidad, por lo que es necesario avanzar en la creación de modelos y métodos de agregación interpretables en AF, para que así se puedan aplicar estos métodos a sistemas en el mundo real.

2 Agregación de Árboles de Decisión en Aprendizaje Federado

Las soluciones basadas en IA fiable son esenciales en las aplicaciones de hoy en día, por lo que hay que priorizar principios como la robustez, seguridad, transparencia, explicabilidad, y privacidad, entre otras. Esto ha hecho que el AF surja como una solución para entrenar modelos de AA en entornos distribuídos, asegurando la privacidad de los nodos involucrados. Los AD son modelos auto-explicativos y, además, son ideales para entornos colaborativos entre múltiples dispositivos con capacidades de computación limitadas, como el del AF. La estructura de los AD impide que su agregación sea trivial, por lo que son necesarias técnicas que puedan agregar los caminos de decisión de los árboles, sin introducir sobre-aprendizaje o sesgo al modelo global, a la vez que mantiene el árbol global robusto y generalizable. En este capítulo se presenta una propuesta del proceso de agregación de AD para AF basado en reglas. Este capítulo se encuentra divido de la siguiente forma: (1) la Sección 2.1 introduce los AD y los algoritmos tradicionales de estos en AAC, y analiza los métodos de AD más utilizados en entornos de AF; (2) la Sección 2.2 presenta el proceso de agregación, así como sus resultados y un análisis del proceso; y, (3) la Sección 2.3 presenta unas breves conclusiones sobre el proceso propuesto.

2.1. Árboles de decisión en Aprendizaje Federado

Un AD es un modelo predictivo con estructura ramificada que se asemeja al proceso deductivo humano. Los AD son modelos del AA tradicional con un gran rendimiento resolviendo problemas de clasificación o regresión, entre otros. Los AD son modelos muy populares en el AA tradicional por su estructura simple, la cual los hace interpretables y explicables. Pese a esto, los AD presentan varios inconvenientes como el sobre-ajuste. Dos modelos algoritmicos de AD muy conocidos son: (1) Classification and Regression Trees (CART) [Bre17], que es un algoritmo versátil y adecuado para tareas de clasificación y regresión. Se construye un árbol binario que divide el dataset de forma recursiva en función de la mejor característica en cada momento; (2) Iterative Dichotomiser 3 (ID3) [Qui86], que está diseñado principalmente para problemas de clasificación. Se construye un árbol de forma recursiva, seleccionando en cada nodo la mejor característica basándose en la métrica de ganancia de información, o *Information Gain (IG)*.

Los AD individuales presentan limitaciones de rendimiento en múltiples problemas, por lo que se han empleado técnicas de *ensemble* para mejorar su rendimiento. Los modelos de *ensemble* basados en AD más destacados incluyen a Random Forest (RF) [Bre01] y a Gradient Boosting Decision Trees (GBDT), con modelos como XGBoost (XGB) [CG16, GGD+20] y *LightGBM* [KMF+17]. RF es una técnica de *bagging* para AD que potencia la generalización y reduce el sobreajuste de los AD individuales. Por su parte LightGBM y XGB utilizan técnicas de *boosting*, refinando el rendimiento del modelo mediante un proceso de optimización iterativa. Aunque los modelos basados en técnicas de *ensemble* mejoran el rendimiento de los AD individuales, estos pierden la interpretabilidad propia de los AD individuales, haciendo que estos modelos sean menos interesantes cuando la interpretabilidad es un factor clave en la tarea a resolver.

Los AD se han utilizado en AF con éxito, mostrando la versatilidad del AF para entrenar un sólo árbol o modelos de *ensemble*.

En la práctica, modelos como RF [HLM+22, LLL+22, LMY+22] y GBDT [YOW22, LHL+22b] se han utilizado tanto en AFH como en AFV, por encima de modelos en los que se construye un sólo árbol como lo es ID3 [TBA+19].

En el estado del arte se pueden encontrar modelos de AD basados en un enfoque de *bagging* como Hauschild et al. [HLM+22], FedRF, en el que se construye un RF global en un escenario de AFH sobre datos médicos. En este modelo cada cliente construye un RF que contiene k árboles locales sobre su dataset. Posteriormente se seleccionan aleatoriamente k árboles de entre todos los disponibles de entre los clientes, formando así un RF global. Yang Liu et al. [LLL+22] construye un RF, pero en este caso cada árbol se construye entre todos los clientes. Por otro lado, Yang Liu et al. [LMY+22] construye un RF de forma colaborativa, cada árbol se construye entre todos los clientes, pero si un cliente abandona el entrenamiento, entonces aquellos árboles que puedan contener información de ese cliente se deben reconstruir, eliminando así cualquier rastro de información del cliente que ha abandonado el proceso federado. Aunque estos enfoques sean atractivos, se han propuesto en entornos federados cuando hay pocos clientes disponibles para entrenar el modelo global, por lo que no se sabe cómo será su rendimiento en entornos de gran escala.

Por otra parte, se encuentran modelos de *boosting* como Kewei Chen et al. [CFJ+21b], el cual propone un *framework* de AFV, *SecureBoost*, donde alinea los datos bajo restricciones de privacidad, y luego entrena un GBDT mientras mantiene todos los datos de entrenamiento de cada cliente protegidos del resto de clientes privados. Este enfoque requiere de un elevado coste de comunicación entre los dispositivos, y por ello surgen otros enfoques que buscan reducir estos costes, mientras mantienen el rendimiento. Weijing Chen et al. [CMF+21] logra mejorar los costes de comunicación de *SecureBoost* incorporando varias optimizaciones para los cálculos del texto cifrado y varias optimizaciones en el proceso de las comunicaciones. Xiaochen Li et al. [LHL+22b] presenta otro *framework* de AFV para *boosting*, *OpBoost*, que busca mejorar la comunicación y el coste computacional, optimizando

la solución basada en la PDL basada en la distancia. Yamamoto et al. [YOW22] propone un *framework* de AFH para *boosting* que optimiza los costes computacionales locales y globales para así reducir los costes de comunicación. Gencturk et al. [GSC22] propone un *framework* de AFH donde se crea un RF en cada cliente, para posteriormente utilizar todos los árboles de cada RF local para actualizar los pesos locales de las instancias de cada cliente. Qinbin Li et al. [LWH20] propone un *framework* de AFH en el que se construye primero una tabla hash global para así alinear las instancias de los clientes sin compartir ningún dato privado. Una vez construida la tabla hash global, se construye un GBDT, en el que cada árbol del *ensemble* lo construye un único cliente a la vez. Después de que un cliente construya el árbol, se añade al modelo de *ensemble* global, y se actualizan los pesos de las instancias locales de cada cliente utilizando el último árbol construido. Tras esto, el siguiente cliente puede construir el siguiente árbol. Al igual que con los enfoques de *bagging*, estos enfoques carecen de experimentación cuando el número de clientes crece, por lo que no se puede saber su rendimiento en escenarios de AF a gran escala.

Truex et al. [TBA+19] propone el modelo Federated-ID3, el cual adapta el modelo clásico ID3 [Qui86] a un escenario de AFH. Este modelo difiere de los modelos presentados previamente, que se centran en estrategias de *bagging* y *boosting*, y es el único encontrado en la literatura que se enfoca en la construcción de un AD en un entorno de AF.

2.2. Proceso de agregación de árboles de decisión en entornos de Aprendizaje Federado

Uno de los objetivos de la tesis es la creación de un agregador de AFH para AD basados en reglas. En esta sección se presenta el proceso llamado Proceso de agregación de AD en entornos de AF (Client Decision Tree Aggregation for Federated Learning process (ICDTA4FL)) [AGZLH24], el cual se particulariza para los modelos ID3 y CART.

2.2.1. Propuesta de proceso de agregación de árboles de decisión en entornos de Aprendizaje Federado con filtrado de clientes

Se ha propuesto el proceso de agregación para AD en un entorno AF, en el que los nodos no comparten sus datos, pero aún así entrenan un modelo de AF. Un AD se puede descomponer en un conjunto de reglas, y este conjunto de reglas se puede estructurar para construir un AD [SR20]. En la Sección 2.1 se ha mostrado una revisión del estado del arte de AD construidos de manera colaborativa entre múltiples clientes en entornos de AFV y AFH. Estos enfoques suelen hacer un uso extensivo de las comunicaciones entre los clientes y el servidor para construir el árbol global, ya sea un árbol simple o un conjunto de árboles. Se propone un proceso de agregación en AFH para AD que es independiente del modelo, donde hay pocas comunicaciones cliente-servidor.

La propuesta ICDTA4FL se basa en la habilidad para unir varios AD [SR20]. El Algoritmo 1 muestra el flujo de trabajo del proceso ICDTA4FL. Este proceso sólo necesita de una ronda de entrenamiento federado para construir el modelo global, y tan sólo necesita de cuatro comunicaciones cliente-servidor, reduciendo así los costes de comunicación del entrenamiento y haciéndolo resistente a posibles errores de conexión de los clientes durante la fase de entrenamiento.

En el proceso ICDTA4FL, se consideran un conjunto de clientes $C = \{C_1, ..., C_n\}$, los cuales entrenan solamente su modelo local, y un servidor que actúa como moderador de las comunicaciones y como agregador. Cada cliente $C_i, \forall i = 1, \ldots, n$, tiene su dataset local $D_i, \forall i = 1, \ldots, n$, el cual nunca se comparte con ningún otro cliente ni con el servidor. El flujo de trabajo de ICDTA4FL se describe a continuación.

Lado de los clientes: Construir y enviar al servidor el modelo local. Existe un conjunto de clientes $C = \{C_1, \ldots, C_n\}$ y un servidor denominado $Server$. Cada cliente $C_i, \forall i = 1, \ldots, n$ tiene sus datos locales, D_i, que nunca se comparten con otros clientes

Algorithm 1 El proceso ICDTA4FL

Client's side
  for C_i; i=1, . . . , n **do**
    $C_i \leftarrow$ train a decision tree, $LocalDT_i$, with its local data D_i.
    Send $LocalDT_i$ to the Server.
Server's side
  Send the received trees to the clients.
Client's side
  for C_i; i=1 to n **do**
    $C_i \leftarrow$ evaluate the local DTs, $C_k LocalDT_i$, $k = 1, \ldots, n; i \neq k$
    Send the evaluation metrics to the server.
Server's side
  Delete the trees that do not surpass a filter selected for the metrics.
  Extract the rules for selected decision trees.
  Aggregate the rules applying the Cartesian product.
  Build a global decision tree, $GlobalDT$ with the aggregated rules.
  Send $GlobalDT$ and the aggregated rules to the clients.
Client's side
  for C_i; i=1 to n **do**
    Evaluate the $GlobalDT$ with its local data D_i

ni con el servidor. El cliente $C_i, \forall i = 1, \ldots, n$ construye un AD local $LocalDT_i$ utilizando el conjunto de datos D_i. Tras crear el AD, el cliente C_i lo envía al servidor.

Lado del servidor: Enviar los AD a los clientes. Los AD tienden a sobre-ajustarse a los datos con los que se entrenan, pudiendo afectar así al modelo global ya que podrían añadir ruido a este. Para abordar este problema, tras recoger todos los AD locales de los clientes, el servidor los reenvía a los mismos, para que cada cliente evalúe sobre sus datos locales (D_i) cada AD entrenado por el resto de clientes.

Lado de los clientes: Evaluar modelos del resto de los clientes El cliente C_i, $i = 1, \ldots, n$ recibe los AD locales originales entrenados por el cliente C_k, $k = 1, \ldots, n$; $k \neq i$. Se denota ese AD como $C_k LocalDT_i$. Cada cliente C_i evalúa los $n - 1$ AD locales $C_k LocalDT_i$ en sus datos privados D_i, y envía las métricas obtenidas al servidor. Las métricas para evaluar los modelos deben ser las mismas que las elegidas en la estrategia de evaluación del modelo global.

Lado del servidor: Construir el AD global El servidor tiene que construir el AD global y enviarlo a los clientes. Este proceso se representa en las siguientes tareas:

- **Eliminar los árboles con bajo rendimiento**
  El servidor tiene $n \times n$ métricas de evaluación asociadas con cada AD local, las cuales se denotan por $EvalC_k LocalDT_i$, $i = 1, \ldots, n$; $k = 1, \ldots, n$. Se calcula la media de las métricas para cada $LocalDT_i$, $i = 1, \ldots, n$ obteniendo $EvalLocalDT_i = \sum_{k=1}^{n} EvalC_k LocalDT_i$. Se aplica un filtro para descartar aquellos árboles cuyas $EvalLocalDT_i$ no superan el filtro fijado. Como filtro por defecto se usa la media, pero se pueden utilizar otros filtros como el percentil, la mediana, la moda, u otros que el usuario decida. Filtrar ayuda a eliminar aquellos AD que pueden añadir ruido al AD global. Este paso es esencial cuando el número de clientes crece, es decir en escenarios de gran escala, o cuando los datos están altamente distribuídos, es decir, en distribuciones non-IID.

- **Obtener las reglas de los árboles**
  El servidor obtiene las reglas de cada AD local. Cada regla se crea desde el nodo hoja hasta el nodo raíz del árbol, por lo que la regla contiene todas las restricciones asociadas. Sea RS_i el conjunto de reglas asociado con el AD $LocalDT_i$, $i = 1, \ldots, n$. Este conjunto de reglas consiste en reglas de la forma $(r_{ij}, \hat{y}_{r_{ij}})$, donde r_{ij} es la condición de la regla y $\hat{y}$ es un vector que contiene las distribuciones de probabilidad de cada una de las Y clases. $\hat{y}$ es un vector K-dimensional, donde K es el número de clases disponibles en el conjunto de datos y cada elemento del vector corresponde a la probabilidad de una clase particular.

- **Agregar las reglas**
  Cuando se agregan dos conjuntos de reglas, $(r_{1j}, \hat{y}_{r_{1j}})$ y $(r_{2j}, \hat{y}_{r_{2j}})$, se utiliza el producto cartesiano, dando lugar a la fusión $(r_{1j} \wedge r_{2j}, \hat{y}_{r_{1j}} + \hat{y}_{r_{2j}})$. La suma de dos vectores de probabilidad se hace mapeando cada elemento del vector con su elemento correspondiente. La compatibilidad de estas reglas se verifica rigurosamente para garantizar que no se contradicen unas con otras, es decir, no son incompatibles. Dependiendo del tipo de AD que se ha construido, esta validación puede cambiar. En el proceso ICDTA4FL presentamos la agregación de dos tipos de árboles que sirven como las bases para modelos más complejos: (1) ID3, dado que es un algoritmo de aprendizaje clásico de AD que ha sido adaptado a entornos federados y (2) CART, ya que es un algoritmo fundamental que sirve como base para métodos de *ensemble* como GBDT y RF. Posteriormente se presenta cómo validar la compatibilidad entre dos reglas de un ID3, dando lugar al modelo *ICDTA4FL-ID3*, y cómo se hace la agregación. Además se presenta el mismo proceso para CART, generando el modelo *ICDTA4FL-CART*.

- **Construir el AD usando las reglas asociadas**: El servidor agrega un conjunto de reglas que contiene todas las reglas agregadas y las probabilidades de las clases asociadas a estas para construir el AD global $GlobalDT$. La estructura del árbol depende del algoritmo de árboles usado por los clientes. Para inicializar el árbol global, el servidor crea un

nodo raíz que contiene todas las reglas del conjunto de reglas agregadas. Para dividir un nodo, el servidor selecciona la característica $f \in F$ que maximiza una métrica específica, la cual dependerá del árbol a construir. La métrica de la característica f se calcula usando las reglas disponibles en el nodo. El servidor entonces divide el conjunto de reglas en dos subconjuntos basándose en el valor de la característica seleccionada. Cada nodo hijo tendrá el conjunto de reglas que cumplen las restricciones, y el proceso de construcción del árbol termina cuando se cumplen los criterios de parada. En concreto, el proceso de generación de nodos termina cuando la clase con mayor probabilidad es consistente en todas las reglas del nodo, o cuando la mejor característica por la cual dividir el nodo no logra reducir las reglas en al menos un nodo hijo. Este proceso es independiente del árbol, y se especificará para los modelos ICDTA4FL-ID3 y ICDTA4FL-CART posteriormente.

- **Enviar el modelo global a los clientes**: El servidor envía el AD global, $GlobalDT$, a los clientes junto con las reglas agregadas.

Lado de los clientes: Evaluación del AD global El cliente C_i, $i = 1, \ldots, n$ evalúa el AD global $GlobalDT$ sobre su dataset local D_i. De esta forma el cliente puede comprobar que el $GlobalDT$ obtenido mejora el rendimiento de su AD local, por lo que considerará utilizar este nuevo árbol para inferir nuevos datos.

2.2.2. Proceso de agregación de reglas y construcción del árbol usando ID3 (Modelo ICDTA4FL-ID3)

Previamente se ha mostrado el proceso de agregación ICDTA4FL, con el que se pueden agregar múltiples AD sin la necesidad de intercambiar información entre los clientes, solo mandando el modelo local. A continuación se presentan los pasos específicos necesitados en el proceso ICDTA4FL cuando se utiliza ID3 como árbol base, es decir, cuando todos los clientes entrenan un ID3 a nivel local, dando lugar al modelo ICDTA4FL-ID3. El

proceso a seguir es el mismo que el explicado el Algoritmo 1, pero se tienen que especificar dos pasos que dependen del árbol utilizado: la agregación de las reglas, y la construcción del AD global usando las reglas agregadas.

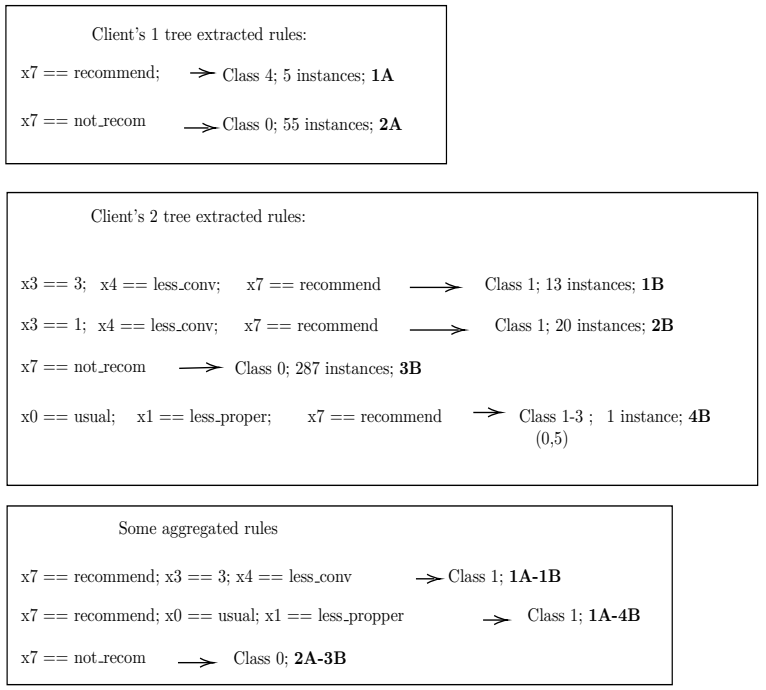


Figura 2.1: Ejemplo de reglas generadas por los árboles locales de los clientes cuando se usa como árbol base un ID3. Las reglas del Cliente 1 se obtienen descomponiendo totalmente el árbol del cliente en reglas, mientras que las reglas del Cliente 2 son solo algunas de generadas por éste. En la figura aparecen las condiciones de las reglas, la clase que predicen, el número de instancias del cliente que cumplen esa regla, y, en negrita, el nombre que se le da a la regla para un mejor entendimiento del proceso de agregación de las reglas.

Agregación de las reglas. El servidor tiene un conjunto de reglas para cada AD local. La Figura 2.1 presenta el conjunto de reglas asociado con dos clientes aleatorios C_1 y C_2. En este ejemplo simulado, el primer cliente dispone de dos reglas, mientras que el segundo tiene cuatro reglas. En este paso sólo se agregan las reglas que son compatibles, es decir, que no se contradicen entre sí. Para poder fusionar dos reglas usando el producto cartesiano en dos árboles ID3, A y B, es necesario asegurar que ninguna condición del antecedente del árbol A está presente en las condiciones del árbol B. Esto es porque si dos reglas contienen antecedentes superpuestos, están denotando lo mismo. Un ejemplo de esta condición de agregación de reglas se puede ver en la Figura 2.1 cuando se intentan fusionar las reglas del cliente C_1 con las reglas del cliente C_2. Primero, se comienza agregando la regla **1A** con **1B**, y antes de fusionarlas se comprueba la compatibilidad de estas. Dado que ambas reglas 1A y 1B comparten la condición ***x7==recommend*** en el antecedente, ambas son compatibles. El resultado de fusionar ambas reglas es la regla **1A-1B**, es decir, ***x7==recommend: x3==3: x4==less_conv*** $\rightarrow$ **Class 1**. Después, la distribución de probabilidad de las clases de cada regla es sumada.

Un ejemplo de dos reglas que se contradicen entre sí son las reglas **2A** y **4B**. La regla **2A** tiene la condición ***x7==not recommend*** en el antecedente, mientras que la regla **2B** tiene la condición ***x7==recommend*** en el antecedente, por lo que estas reglas se contradicen una a la otra.

Construcción del AD global. El servidor usa el conjunto de reglas agregadas para construir un ID3. El proceso de construcción del árbol terminar cuando se cumplen los criterios de parada, es decir, cuando se cumplen las condiciones presentadas en la Sección 2.2.1, o cuando no hay más características disponibles por las que dividir un nodo. Para dividir un nodo, se calcula la Ganancia de Información, o *Information Gain* (IG) para cada nodo del AD y se selecciona la característica $f_r \in F$ con mayor valor de la métrica IG. El valor de la métrica IG asociado a la característica f_r se especifica en la Ecuación 2.1. La entropía del nodo se calcula como la entropía ($argmax(\hat{y}_1)$, $argmax(\hat{y}_2)$, $argmax(\hat{y}_K)$) de todas las

restricciones incluidas en el nodo [SR20].

$$IG(RS_r, R) = \frac{\sum_{s=1}^{k}(|RS_{rs}| * entropy(RS_{rs}))}{\sum_{s=1}^{k}|RS_{rs}|} - entropy(RS_r) \tag{2.1}$$

2.2.3. Proceso de agregación de reglas y construcción del árbol usando CART (Modelo ICDTA4FL-CART)

En esta sección se especifican los pasos necesarios en el proceso ICDTA4FL para funcionar cuando los clientes entrenan como AD base un CART, llamado modelo ICDTA4FL-CART. A continuación se describen los dos pasos dependientes del algoritmo de AD en la construcción del AD global.

Agregación de las reglas. La condición de contradicción en un conjunto de reglas obtenidas de un CART difiere de las condiciones de un ID3, ya que las condiciones del CART representan intervalos de una separación binaria, (Véase Figura 2.2). En este caso, si dos reglas comparten la misma condición, es decir, regla **2A** y regla **4B**, se selecciona la condición menos restrictiva. Mientras que la regla **2A** tiene la condición $x0 > 32.5$, la regla **4B** tiene la condición $x0 > 35$, siendo la segunda condición más restrictiva que la primera, por lo que al fusionarlas se mantendría la condición de la regla **2A**. Un ejemplo de reglas que se contradicen entre sí son las reglas **1A** y **3B**. La regla **1A** tiene la condición $x0 <= 32.5$, mientras que la regla **3B** tiene la condición $x0 > 35$, haciendo la fusión de reglas incompatible.

Construcción del AD global. Un CART es un AD binario, por lo que la métrica para dividir un nodo es diferente a la del ID3, y se da en la Ecuación 2.2. Las condiciones de parada de construcción del árbol son las descritas en la Sección 2.2.1.

$$IG(RS_i, R) = \frac{|RS_{i1}| * entropy(RS_{i1}) + |RS_{i2}| * entropy(RS_{i2})}{|RS_{i1}| + |RS_{i2}|} - entropy(RS_i) \tag{2.2}$$

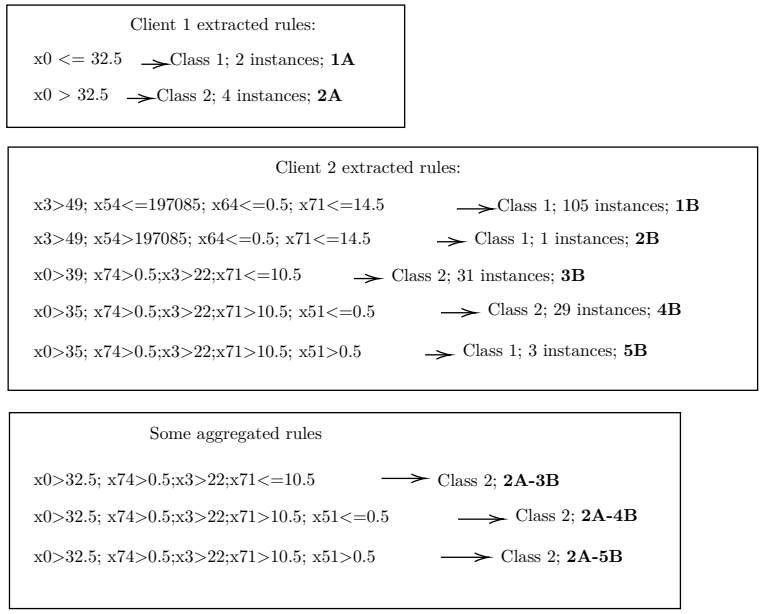


Figura 2.2: Ejemplo de reglas generadas por los árboles locales de los clientes cuando se usa como árbol base un CART. Las reglas del Cliente 1 se obtienen descomponiendo totalmente el árbol del cliente en reglas, mientras que las reglas del Cliente 2 son solo algunas reglas generadas por éste. En la figura aparecen las condiciones de las reglas, la clase que predicen, el número de instancias del cliente que cumplen esa regla, y, en negrita, el nombre que se le da a la regla para un mejor entendimiento del proceso de agregación de las reglas.

2.2.4. Marco experimental

Para validar la propuesta, se ha realizado una batería de experimentos. Estos experimentos se describen en las siguientes secciones:(1) Datasets y distribución de datos, (2) Métricas y clientes, (3) Hiperparámetros de los AD, y (4) Comparación con el estado del arte.

Datasets y distribución de datos. Se han seleccionado 4 datasets para llevar a cabo el marco experimental: Nursery[1], Adult[2], Car[3], y Credit2[4]. Estos datasets son muy utilizados por los modelos de la literatura de AD en un entorno federado. La información de estos datasets se muestra en la Tabla 2.1.

Nombre	Tipo Dataset	#Instancias	Características	Clases
Adult	Numérico y Categórico	48842	14	2
Nursery	Categórico	12960	8	5
Car	Categórico	1728	6	4
Credit	Numérico	30000	24	2

Tabla 2.1: Datasets utilizados en el marco experimental.

La distribución de datos es muy importante en los entornos federados, y es importante comprobar el rendimiento del modelo en estas distribuciones de datos. Por tanto, se plantean experimentos con distribuciones de datos IID y non-IID para evaluar el rendimiento de los modelos locales y del modelo global.

- En la distribución IID todos los clientes tienen el mismo número de instancias y el mismo número de clases.
- En la distribución non-IID los clientes podrán tener diferentes clases, y sobre todo un número de datos diferentes entre ellos. Estas distribuciones se ha hecho de forma totalmente aleatoria, pero asegurando que cada cliente tenga al menos 5 instancias.

Métricas y clientes. Las métricas consideradas para evaluar el rendimiento de los modelos obtenidos con el proceso ICDTA4FL son el *Accuracy* (Acc) y el *Macro-F1* (F1). Es esencial mantener el rendimiento del modelo cuando incrementa el número de clientes, dado que el número de clientes dependerá de la tarea que vaya a resolver, o del número de clientes disponibles para entrenar. En el marco experimental por tanto se utilizarán 2, 5, 10, 20, y 50 como

[1] `https://archive.ics.uci.edu/ml/datasets/nursery`
[2] `https://archive.ics.uci.edu/ml/datasets/adult`
[3] `https://archive.ics.uci.edu/dataset/19/car+evaluation`
[4] `https://archive.ics.uci.edu/ml/datasets/default+of+credit+card+clients`

el número de clientes en los que se analizará el rendimiento del modelo, cuando se pase de escenarios con pocos clientes a escenarios a gran escala. Además, se ha realizado una validación cruzada de 10 veces (10CV) de los que se obtendrá la media de los resultados. Cada cliente evaluará su AD local y el AD global en el dataset de validación local de cada iteración del 10CV. Para obtener los resultados, en cada cliente se calculará la media del 10CV, y luego se calculará la media de los resultados de todos los clientes para obtener las métricas finales.

Hiperparámetros de los AD. El proceso ICDTA4FL funciona con múltiples tipos de AD, y para probarlo se han realizado experimentos que usan como modelo base de los modelos locales de los clientes los algoritmos ID3 y CART. Se fijan dos parámetros al inicializar cada modelo local, la profundidad máxima y el criterio. Para CART, se fija la profundidad máxima a 5, y se utiliza el índice-gini como criterio, mientras que para ID3 se fija $\frac{|F|}{2}$ como profundidad máxima e IG como el criterio. Aquí F representa el número de características del dataset. En el caso de los modelos globales, se utilizará la misma profundidad máxima que la fijada en los modelos locales, y se utilizará como criterio de los AD globales las definidas en la Sección 2.2.1. Por tanto, para ICDTA4FL-ID3 se usará la IG cuya métrica se define en la Ecuación 2.1, mientras que para CART se usará como criterio la definida en la Ecuación 2.2.

Modelo Federated-ID3: Un modelo de AF basado en ID3 usado para la comparación con el estado del arte. En este capítulo se ha mostrado que los modelos del estado del arte basados en AD para entornos de AF son modelos de *ensemble*, utilizando estrategias de *bagging* o de *boosting*, pero no construyen un modelo global formado por un solo AD como se hace con el proceso ICDTA4FL. El modelo Federated-ID3 [TBA+19] es el único método que construye un sólo AD en lugar de un construir un *ensemble* de árboles, por lo que se usará este modelo para comparar el proceso ICDTA4FL, en concreto se comparará con el modelo ICDTA4FL-ID3. El modelo Federated-ID3 [TBA+19] selecciona la característica f que maximiza la IG. Se comienza construyendo el

nodo raíz, y para cuando se alcanza un nodo hoja, cuando no hay más características disponibles o cuando se alcanza la profundidad máxima, $\frac{|F|}{2}$, siendo F el conjunto de características total del dataset. En el modelo Federated-ID3 se construye un solo AD agregando los *counts* y los *class counts* para calcular la IG, mientras que el modelo ICDTA4FL-ID3 construye un AD global usando los AD locales de los clientes.

2.2.5. Resultados experimentales y análisis

En esta sección se presentan los resultados del estudio realizado para ambos modelos (ICDTA4FL-ID3 y ICDTA4FL-CART), y un análisis profundo de los mismos. Inicialmente se mostrarán los resultados obtenidos por el modelo ICDTA4FL-ID3 y se compararán estos con los obtenidos con el modelo Federated-ID3. Posteriormente se compararán los resultados del modelo ICDTA4FL-ID3 con los obtenidos por el modelo ICDTA4FL-CART. Tras esto se analizará el rendimiento de ambos modelos con diferentes números de clientes participando en el entrenamiento. También se mostrará la robustez del proceso ICDTA4FL cuando se utilizan otros filtros diferentes a la media para elegir la cantidad de árboles a agregar, usando como ejemplo el percentil. Por último, se mostrará la interpretabilidad del proceso ICDTA4FL sobre los AD locales construidos por los clientes.

Modelo ICDTA4FL-ID3: Resultados y análisis. A continuación se van a mostrar los resultados obtenidos por el proceso ICDTA4FL cuando los clientes crean localmente un ID3. Se comparan los resultados obtenidos por el modelo ICDTA4FL-ID3 con los del modelo del estado del arte, el modelo Federated-ID3 [TBA+19]. Los resultados se presentan en dos tablas, las Tablas 2.2 y 2.3, en la que se muestran los resultados sobre los datasets mostrados en la Tabla 2.1. La Tabla 2.2 presenta los resultados obtenidos cuando se da una distribución de datos IID. La Tabla 2.3 presenta los resultados obtenidos cuando se da una distribución de datos non-IID. El modelo *baseline* representa la media de los resultados de los AD locales de los clientes, evaluados sobre sus test locales. Dado que no todas las características de los datasets Adult y Credit son categóricas, es necesario realizar

un preprocesamiento para que sean compatibles con el árbol ID3. El prepocesamiento realizado ha sido un preprocesamiento básico que convierte las características de ambos dataset de numéricas a categóricas.

		Nursery		Adult		Car		Credit2	
		Acc	**F1**	**Acc**	**F1**	**Acc**	**F1**	**Acc**	**F1**
2 Cli	**Baseline (ID3)**	90.83	69.85	81.37	73.31	88.01	65.43	80.22	**66.21**
	ICDTA4FL-ID3	**90.93**	**73.75**	81.53	**73.64**	**90.08**	**74.6**	80.35	65.98
	Federated-ID3 [TBA$^+$19]	89.22	60.18	**83.64**	64.14	77.65	37.88	**83.71**	65.96
5 Cli	**Baseline (ID3)**	90.41	**78.26**	80.87	**73.13**	**82.44**	**56.19**	81.48	67.61
	ICDTA4FL-ID3	**91.75**	76.22	80.52	72.96	79.59	43.57	**82.81**	**68.39**
	Federated-ID3 [TBA$^+$19]	88.79	63.07	**81.51**	55.06	77.37	37.17	75.34	49.61
10 Cli	**Baseline (ID3)**	88.65	**70.56**	88.18	72.39	**78.33**	**49.45**	77.27	63.01
	ICDTA4FL-ID3	**89.85**	68.31	**89.97**	**72.48**	77.91	48.79	**81.1**	**67.17**
	Federated-ID3 [TBA$^+$19]	88.42	66.3	79.8	51.17	72.83	38.8	74.87	51.97
20 Cli	**Baseline (ID3)**	85.19	63.81	77.96	**68.85**	75.41	49.86	78.18	63.55
	ICDTA4FL-ID3	87.5	**74.54**	**78.57**	63.87	**77.5**	**57.54**	**80.95**	**64.13**
	Federated-ID3 [TBA$^+$19]	**88.12**	66.46	77.03	47.81	72.67	36.44	65.30	56.25
50 Cli	**Baseline (ID3)**	82.10	64.55	77.23	66.12	62.8	45.44	75.77	61.29
	ICDTA4FL-ID3	**87.42**	**74.04**	**81.8**	**70.41**	**67.6**	**53.55**	**81.31**	**64.93**
	Federated-ID3 [TBA$^+$19]	84.42	66.11	71.45	44.82	62.13	36.22	69.83	50.97

Tabla 2.2: Resultados del modelo ICDTA4FL-ID3 contra el estado del arte en una distribución de datos IID. Además, se compara el modelo ICDTA4FL-ID3 con el modelo *baseline*. El modelo *baseline* se refiere a la media de los AD locales construidos por los clientes cuando se construye un ID3 como modelo base.

Las Tablas 2.2 y 2.3 muestran los resultados del modelos ICDTA4FL-ID3 contra el modelo del estado del arte, Federated-ID3 [TBA$^+$19]. Ambas tablas muestran que el modelo ICDTA4FL-ID3 mejora los resultados del modelo Federated-ID3, además de mejorar al *baseline*, mejorando así los modelos locales de los clientes.

Modelo ICDTA4FL-CART: Resultados y análisis. A continuación se muestran los resultados obtenidos por el proceso ICDTA4FL cuando los clientes entrenan un CART localmente, dando lugar al modelo ICDTA4FL-CART. Dado que no hay ningún modelo del estado del arte que construya un solo árbol, y

		Nursery		Adult		Car		Credit2	
		Acc	**F1**	**Acc**	**F1**	**Acc**	**F1**	**Acc**	**F1**
2 Cli	**Baseline (ID3)**	91.04	70.22	80.01	71.8	80.83	45.09	80.67	67.17
	ICDTA4FL-ID3	**91.45**	**71.45**	81.66	**73.36**	**86.25**	**60.93**	80.96	**67.55**
	Federated-ID3 [TBA$^+$19]	89.22	59.57	**83.47**	64.03	77.29	37.31	**83.81**	67.02
5 Cli	**Baseline (ID3)**	90.39	73.52	79.5	71.45	79.61	**54.37**	78.81	62.96
	ICDTA4FL-ID3	**91.75**	**75.31**	**82.45**	**73.81**	**79.95**	45.76	**80.4**	**64.7**
	Federated-ID3 [TBA$^+$19]	88.86	63.96	81.74	54.69	76.32	36.81	75.31	50.63
10 Cli	**Baseline (ID3)**	86.46	70.62	77.47	**69.33**	75.41	**55.03**	73.94	60.03
	ICDTA4FL-ID3	**91.52**	**75.08**	**80.39**	68.75	**79.74**	54.02	**84.28**	**71.53**
	Federated-ID3 [TBA$^+$19]	88.16	65.15	79.69	51.08	75.95	35.45	73.94	45.01
20 Cli	**Baseline (ID3)**	85.96	65.92	76.47	66.97	66.94	42.44	75.58	**59.48**
	ICDTA4FL-ID3	**88.4**	**69.89**	**78.98**	**67.86**	**77.99**	**53.89**	**77.58**	53.66
	Federated-ID3 [TBA$^+$19]	87.47	67.69	77.09	48.18	74.19	37.2	66.31	56.85
50 Cli	**Baseline (ID3)**	82.34	64.56	76.77	**64.13**	62.48	45.56	73.56	58.16
	ICDTA4FL-ID3	**85.35**	**72.45**	**78.04**	52.43	**70.64**	**49.44**	**81.05**	**63.8**
	Federated-ID3 [TBA$^+$19]	84.31	64.21	71.6	44.97	66.28	39.1	67.83	49.69

Tabla 2.3: Resultados del modelo ICDTA4FL-ID3 contra el estado del arte en una distribución de datos non-IID. Además, se compara el modelo ICDTA4FL-ID3 con el modelo *baseline*. El modelo *baseline* se refiere a la media de los AD locales construidos por los clientes cuando se construye un ID3 como modelo base.

que este sea un AD tipo CART, no se realizará una comparación con modelos que utilicen técnicas de *bagging* o *boosting*, puesto que estos modelos se crearon para mejorar el rendimiento de los AD individuales. Por ello, en su lugar la comparación se realizará con el modelo ICDTA4FL-ID3, ya que es el otro modelo que se puede obtener utilizando el proceso ICDTA4FL. Al igual que al usar ID3, los datasets necesitan ser preprocesados para poder ser compatibles con un AD tipo CART. El preprocesamiento se ha realizado sobre los datasets Nursery y Car, dado que sus características son categóricas, y se han transformado todas sus características en numéricas. El dataset Adult también se ha preprocesado para cumplir estos requisitos, puesto que algunas de sus características eran categóricas. Las Tablas 2.4 y 2.5 presentan los resultados de esta comparativa. La Tabla 2.4 muestra los resultados cuando la distribución de datos es IID, y la Tabla 2.5 muestra los resultados cuando la distribución de datos es non-IID.

		Nursery		Adult		Car		Credit2	
		Acc	**F1**	**Acc**	**F1**	**Acc**	**F1**	**Acc**	**F1**
2 Cli	**Baseline (CART)**	87.65	65.13	84.74	76.48	85.04	55.21	81.73	66.88
	ICDTA4FL-ID3	**90.93**	**73.75**	81.53	73.64	**90.08**	**74.6**	80.35	65.98
	ICDTA4FL-CART	87.87	65.3	**84.94**	**76.96**	86.04	55.91	**81.9**	**67.45**
5 Cli	**Baseline (CART)**	87.09	65.56	84.51	76.57	83.51	56.63	81.55	67.32
	ICDTA4FL-ID3	**91.75**	**76.22**	80.52	72.96	79.59	43.57	**82.81**	**68.39**
	ICDTA4FL-CART	89.2	67.17	**85.01**	**77.32**	**87.75**	**63.79**	81.96	67.82
10 Cli	**Baseline (CART)**	86.51	65.01	83.76	75.47	82.01	54.94	80.43	65.36
	ICDTA4FL-ID3	**89.85**	**68.31**	**89.97**	72.48	77.91	48.79	**81.1**	**67.17**
	ICDTA4FL-CART	88.37	66.37	84.65	**76.81**	**87.04**	**63.24**	81.08	65.24
20 Cli	**Baseline (CART)**	85.34	65.27	83.19	74.86	68.19	36.24	79.74	64.82
	ICDTA4FL-ID3	**87.5**	**74.54**	78.57	63.87	**77.5**	**57.54**	**80.95**	64.13
	ICDTA4FL-CART	87.1	66.67	**84.39**	**76.64**	68.44	44.55	80.2	**65.18**
50 Cli	**Baseline (CART)**	82.6	69.3	81.12	71.52	62.93	43.81	77.85	62.69
	ICDTA4FL-ID3	**87.42**	**74.04**	81.8	70.41	67.6	**53.55**	**81.31**	**64.93**
	ICDTA4FL-CAR)	82.6	69.3	**83.65**	**74.56**	**71.08**	50.78	80.27	64.02

Tabla 2.4: Resultados del modelo ICDTA4FL-CART contra el estado del arte en una distribución de datos IID. Además, se compara el modelo ICDTA4FL-CART con el modelo *baseline*. El modelo *baseline* se refiere a la media de los AD locales construidos por los clientes cuando se construye un CART como modelo base.

Las Tablas 2.4 y 2.5 contienen los resultados del modelo ICDTA4FL-CART contra los del modelo ICDTA4FL-ID3. Estas tablas muestran que el modelo ICDTA4FL-ID3 funciona mejor en ambos tipos de datasets, numéricos y categóricos, cuando el número de clientes crece, mientras que el modelo ICDTA4FL-CART obtiene mejor rendimiento con los datasets numéricos en escenarios con menores clientes.

Análisis basado en el número de clientes. Uno de los aspectos claves del AF es la habilidad de mejorar el rendimiento del modelo a través de la colaboración entre diferentes nodos propietarios de datos. En esta propuesta se ha examinado el impacto del número de clientes en el rendimiento del AD global, y los resultados se muestran en las Tablas 2.2, 2.3, 2.4 y 2.5. El número de clientes va de 2 a 50, y todos los datos se han particionado entre los clientes sin que compartan ninguna instancia. Cuando el número de clientes aumenta, cada cliente tiene menos datos locales con los que entrenar el modelo local, por lo que el número de árboles

		Nursery		Adult		Car		Credit2	
		Acc	**F1**	**Acc**	**F1**	**Acc**	**F1**	**Acc**	**F1**
2 Cli	**Baseline (CART)**	87.44	63.74	84.82	76.33	83.93	55.48	**81.87**	**68.3**
	ICDTA4FL-ID3	**91.45**	**71.45**	81.66	73.36	**86.25**	**60.93**	80.96	67.55
	ICDTA4FL-CART	87.65	63.09	**85.06**	**76.78**	86.13	58.8	81.5	64.97
5 Cli	**Baseline (CART)**	87.07	65.52	84.49	76.27	85.06	56.26	81.66	67.2
	ICDTA4FL-ID3	**91.75**	**75.31**	82.45	73.81	79.95	45.76	80.4	64.7
	ICDTA4FL-CART	88.72	66.81	**85.22**	**77.53**	**87.55**	**62.89**	**82.08**	**68.05**
10 Cli	**Baseline (CART)**	86.30	64.9	84.14	75.56	82.25	54.71	80.7	66.17
	ICDTA4FL-ID3	**91.52**	**75.08**	80.39	68.75	79.74	54.02	**84.28**	**71.53**
	ICDTA4FL-CART	88.76	66.69	**85.14**	**77.06**	**87.02**	**65.46**	81.28	66.42
20 Cli	**Baseline (CART)**	85.58	65.59	83.49	**75.02**	68.13	35.67	79.68	64.50
	ICDTA4FL-ID3	**88.4**	**69.89**	78.98	67.86	**77.99**	**53.89**	77.58	53.66
	ICDTA4FL-CART	86.36	66.29	**84.26**	74.42	68.85	49.05	**79.67**	**62.8**
50 Cli	**Baseline (CART)**	83.15	69.62	81.55	72.35	64.16	45.55	77.75	61.98
	ICDTA4FL-ID3	**85.35**	**72.45**	78.04	52.43	**70.64**	49.44	**81.05**	**63.8**
	ICDTA4FL-CART	76.7	61.72	**81.94**	**72.78**	69.6	**50.63**	78.37	61.04

Tabla 2.5: Resultados del modelo ICDTA4FL-CART contra el estado del arte en una distribución de datos non-IID. Además, se compara el modelo ICDTA4FL-CART con el modelo *baseline*. El modelo *baseline* se refiere a la media de los AD locales construidos por los clientes cuando se construye un CART como modelo base.

que pueden añadir ruido al AD global aumentará. Esto afectará al rendimiento del modelo global, y el filtro de árboles ayuda a que el rendimiento no se vea afectado. El modelo ICDTA4FL-ID3 pierde poco rendimiento cuando aumenta el número de clientes, manteniendo mejor rendimiento que el modelo del estado del arte, como se muestra en las Tablas 2.2 y 2.3. El modelo ICDTA4FL-CART también mantiene el rendimiento desde 2 hasta 50 clientes gracias al filtrado de AD realizado, puesto que elimina aquellos AD que pueden añadir ruido. De esta forma se muestra la estabilidad en el rendimiento del modelo ICDTA4FL-CART cuando aumenta el número de clientes, Tablas 2.4 y 2.5.

Robustez del proceso ICDTA4FL filtrando por el percentil. El proceso ICDTA4FL realiza un filtrado de AD que se agregarán para construir el AD global, eliminando aquellos árboles que no superan el umbral para el filtro seleccionado. Por defecto se utiliza la media como umbral, pero se puede seleccionar otras métricas como umbral, como lo son el percentil, la mediana, la moda, etc.,

siendo estos filtros más restrictivos que el usado por defecto. El filtrado es esencial en el proceso ICDTA4FL, ya que ayuda a reducir el número de árboles que se agregan, sobre todo cuando el número de clientes aumenta.

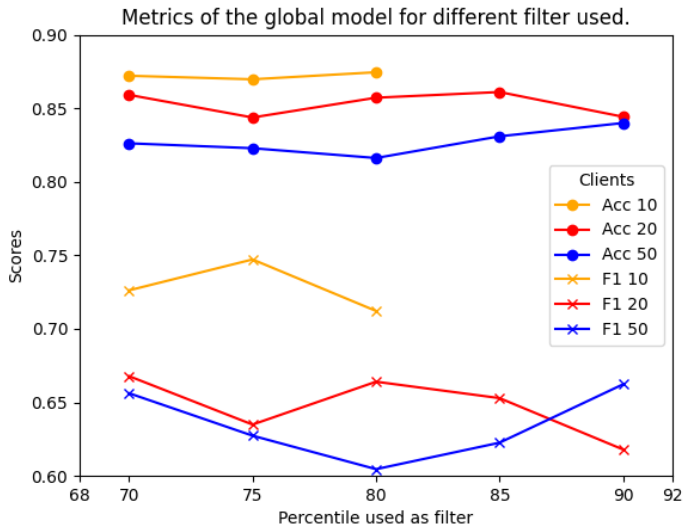


Figura 2.3: Se explora el impacto de ajustar el filtro en *Accuracy* y *Macro-F1* en una distribución non-IID en el dataset Nursery, y utilizando el modelo ICDTA4FL-ID3. El filtro está basado en el uso del percentil, eliminando aquellos árboles que no superan el percentil seleccionado cuando se construye el árbol global.

El principal motivo por el que se ha aplicado este filtrado, es que en distribuciones de datos non-IID donde los clientes tienen muy pocos datos, los árboles generados por estos clientes añaden mucho ruido al modelo global, perjudicando el rendimiento. Para demostrar esto, se ha realizado un experimento utilizando el proceso ICDTA4FL con el percentil como filtro. En la Figura 2.3 se muestra cómo el rendimiento del modelo varía con el dataset Nursery dependiendo del filtro, utilizando diferentes percentiles, siendo los resultados similares a los obtenidos cuando se utiliza la media. La ventaja de utilizar un filtro más restrictivo es que el tiempo

para generar el AD global se ve reducido, puesto que se utilizan menos árboles.

Ejemplo ilustrativo del comportamiento y la interpretabilidad de ICDTA4FL. Los AD son explicables por su arquitectura. Los árboles se pueden representar como un conjunto de reglas que manifiesta cómo las características de entrada influyen en la predicción. Esto promueve la confianza en el modelo al permitir que los clientes verifiquen el comportamiento del modelo, que entiendan la lógica detrás de la predicción del modelo, y que verifiquen si se alinea con sus expectativas. Se ha utilizado esta característica para comprobar los motivos por los que el AD global rinde mejor que el AD local con ID3. Un ejemplo donde estos modelos destacan es en casos como en el de la medicina, donde un médico utilizará un modelo interpretable y explicable antes que uno de caja negra, dado que además de tener la predicción, tiene los motivos de dicha predicción.

Instance	Class			Explanation	
	Original	$LocalDT_i$	GlobalDT	$LocalDT_i$	GlobalDT
1	1	2	1	{x7==priority, x1==improper, x0==pretentious}	{x7==priority, x1==improper, x0==pretentious, x5==convenient}
2	3	1	3	{x7==recommended, x1==very_crit, x6==slightly_prob, x3==2}	{x7==recommended, x1==very_crit, x6==slightly_prob, x4==less_conv}
3	1	3	1	{x7==recommended, x1==critical, x0==great_pret, x3==1}	{x7==recommended, x1==critical, x0==great_pret, x3==1}

Tabla 2.6: Clasificación y explicación proporcionada por el AD local del cliente C_i ($LocalDT_i$) y el AD global ($GlobalDT$) para tres instancias del dataset Nursery usando el modelo ICDTA4FL-ID3.

Se analiza el rendimiento del modelo global contra el modelo local analizando las predicciones realizadas y las reglas asociadas a estas predicciones. La Tabla 2.6 presenta tres instancias del dataset Nursery que se han clasificado de forma errónea por el modelo local y que el modelo global sí ha clasificado bien. La primera instancia está bien clasificada gracias a que el modelo global añade

una condición en la característica x_5 en el antecedente de la regla de decisión. La segunda instancia está bien clasificada por el modelo global puesto que reemplaza una condición en la característica (x_3) con la condición de otra característica (x_4). La tercera instancia pasa a estar bien clasificada por el modelo global, incluso cuando las reglas son las mismas que las del modelo local, porque el modelo global es capaz de definir mejor las fronteras de decisión.

2.3. Conclusiones

El proceso ICDTA4FL es un proceso de agregación de AD en entornos de AF independiente del algoritmo de árboles de decisión, que es capaz de agregar diferentes AD en uno solo, manteniendo la interpretabilidad y la estructura de dichos árboles. Se ha probado el rendimiento del proceso en un entorno de AFH, en las distribuciones IID y non-IID sobre 4 datasets, y cambiando el número de clientes, para verificar que se adapta a entornos de gran escala. Se han usado ID3 y CART como modelos base, obteniendo finalmente los modelos ICDTA4FL-ID3 y ICDTA4FL-CART.

Por tanto, se concluye que el proceso ICDTA4FL presenta las siguientes ventajas:

- Puede funcionar con varios tipos de AD, siendo capaz de adaptarse al problema a resolver.
- Obtiene un rendimiento excelente en distribuciones de datos IID y non-IID, mejorando los modelos locales de los clientes.
- Mantiene la estructura jerárquica de los AD originales que se usan para construir el modelo global. El proceso ICDT4FL es capaz de generalizar el conocimiento de los clientes en un sólo árbol, y permite entender el comportamiento del modelo.

3 FLEX-Trees: Una librería para árboles de decisión

En este capítulo se introduce FLEX-Trees, una librería de FLEX, el cual es un *framework* de AF que ofrece a los investigadores de AF la posibilidad de *flexibilizar* al máximo su experimentación. Con FLEX se pueden simular experimentos que posteriormente se pueden dar en el mundo real, habilitando la creación de métodos novedosos de AF de una manera más rápida a la que lo hacen otros *frameworks* presentados en esta tesis. FLEX-Trees es una librería adyacente a FLEX, que incorpora nuevas funcionalidades al *framework* original, en este caso integra modelos interpretables del estado del arte de AF. Es necesario disponer de herramientas que ayuden a la creación de métodos interpretables y/o explicables, dado que son requisitos de un sistema de IA fiable, sobre todo en ámbitos donde se trabajan con datos sensibles. Los modelos de FLEX-Trees sirven como ejemplo a los investigadores para la creación de nuevos métodos interpretables novedosos. Este capítulo se divide en varias secciones: (1) la Sección 3.1 que incluye una breve descripción del *framework* de AF FLEX; (2) la Sección 3.2 que describe el módulo de interpretabilidad, FLEX-Trees, además de un ejemplo de uso de FLEX y FLEX-Trees; y, (3) la Sección 3.4 donde se presentan unas conclusiones breves sobre la importancia de FLEX-Trees.

3.1. Descripción de la plataforma FLEX

En el Capítulo 1 se han mostrado los diferentes *frameworks* de AF con sus respectiva propiedades. Sin embargo, estos *frameworks* presentan múltiples limitaciones a la hora de realizar simulaciones de experimentos de AF. Los investigadores deberían poder diseñar un experimento federado en múltiples aspectos, como la distribución de datos, los parámetros de privacidad, las estrategias de comunicación, e incluso arquitecturas federadas personalizadas. Proporcionar esta flexibilidad a los investigadores impulsará su investigación y les permitirá explorar nuevas técnicas de AF, o adaptar el AF a casos específicos sin la carga de tener que lidiar con las complejidades del código repetitivo. Un *framework* de AF debería ser clave para acelerar la investigación de AF, proporcionando un entorno controlado que imite las complejidades del mundo real mientras permite a los investigadores desarrollar nuevas técnicas de AF.

Por ello se ha desarrollado *FLEX: a FLEXible Federated Learning Framework* [HJLAG+24], una librería de simulación de experimentos federados de código abierto con licencia Apache 2.0 y escrita en Python. FLEX ofrece a los investigadores la posibilidad de crear casi cualquier distribución de datos non-IID, además de una distribución de datos IID, así como la posibilidad de crear diferentes arquitecturas que se parezcan mucho más a las que se puedan dar en el mundo real como la explicada en [BEG+19]. FLEX es compatible con múltiples *frameworks* de AA y AP centralizados, lo que permite que al investigador una adaptación más rápida que otros *frameworks* al comenzar a usarlo. FLEX permite montar un flujo de trabajo como el mostrado en la Figura 1.5, y además viene con varias librerías, los cuales complementan su funcionalidad.

3.1.1. Módulos de FLEX

FLEX está diseñada bajo la premisa de ser una librería para la federación de modelos, y no como una librería de modelos federados. Por ello, FLEX está compuesta por diferentes módulos que permiten una fácil y rápida implementación de nuevos modelos federados, sin la necesidad de repetir las bases de un entorno

federado como lo son la distribución de los datos, o la creación de la arquitectura federada. Estos módulos se pueden ver en la Figura 3.1.

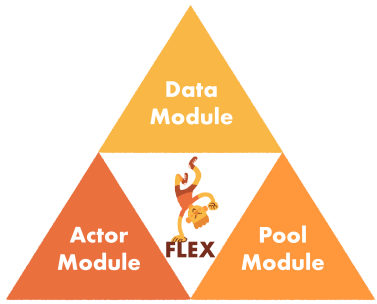


Figura 3.1: Módulos de la librería FLEX.

Módulo de datos El módulo de datos es el encargado de la lectura y federación de los datos para poder simular distribuciones de datos que se pueden dar en un entorno real. FLEX dispone de los datasets inherentemente federados de LEAF [CDW+19], y de otros datasets que no son federados per se, pero que son muy utilizados en la la literatura de AF [LRBAG+24]. FLEX no sólo dispone de estos datasets inherentemente federados, sino que además facilita el uso de datasets de otras librerías como PyTorch, TensorFlow o HuggingFace, o un dataset propio. Los datasets inherentemente federados generarán tantos clientes como la característica que identifica a los usuarios únicos. Por otro lado, los datasets propios de un entorno de AA tradicional, se pueden particionar en distribuciones IID entre múltiples cliente, y también se puede realizar casi cualquier distribución non-IID. En este último caso, el usuario puede utilizar funciones que realicen esta distribución.

También se puede realizar el particionado de datos para los diferentes escenarios del AF como son AFH, AFV, o AFT. Para mostrar cómo hacerlo usando FLEX y FLEX-Trees, se muestra

Código 24, en el que se carga el dataset Credit2, y se crean las configuraciones para los diferentes escenarios:

- **Escenario AFH**: La configuración se muestra en la línea 3 Código 24. Posteriormente se aplica la configuración para el entorno AFH en la línea 10.
- **Escenario AFV**: La configuración se muestra en la línea 5 Código 24. Para aplicar la configuración, bastaría con indicar que se usa la *config_afv* en lugar de *config_afh* en la línea 10.
- **Escenario AFT**: La configuración se muestra en la línea 7 Código 24. Para aplicar la configuración, bastaría con indicar que se usa la *config_aft* en lugar de *config_afh* en la línea 10.

```
1  train_dataset, test_dataset =
   ↪  flextrees.datasets.tabular_datasets.credit2(ret_feature_names=False,
   ↪  categorical=False)
2  # Configuración para AFH
3  config_afh = flex.data.FedDatasetConfig(seed=0, n_nodes=5,
   ↪  replacement=False)
4  # Configuración para AFV
5  config_afv = flex.data.FedDatasetConfig(seed=0, n_nodes=2,
   ↪  replacement=True, features_per_node=[1, 2],
   ↪  keep_labels=[False, True])
6  # Configuración para AFT
7  config_aft = flex.data.FedDatasetConfig(seed=0, n_nodes=2,
   ↪  replacement=True, features_per_node=4, keep_labels=[True,
   ↪  False])
8  # Una vez creada la configuración, sólo hay que aplicarla
9  # Se pone como ejemplo el uso de config_afh
10 federated_dataset_afh =
   ↪  flex.data.FedDataDistribution.from_config(train_dataset,
   ↪  config_afh)
11
```

Código 24: Ejemplo de carga del dataset Credit2 usando FLEX y FLEX-Trees, y creando los escenarios AFH, AFV y AFT. Se carga Credit2 desde la librería *flextrees*, línea 1, y luego se crean las diferentes configuraciones de AFH, AFV, y AFT. Se muestra cómo aplicar la configuración para el escenario AFH, el resto se haría de la misma forma.

Usando **FedDatasetConfig** se pueden crear diferentes distribuciones y escenarios que permiten al investigador comprobar si realmente funcionaría su propuesta en un entorno federado o no.

Módulo de actores FLEX permite la creación de diferentes arquitecturas de datos, desde la arquitectura cliente-servidor, la más típica en AF, en la que hay múltiples nodos cliente y un nodo independiente que actúa de servidor-agregador, hasta una arquitectura *peer-to-peer*, en la que todos los nodos actúan como cliente-agregador-servidor. En FLEX se definen diferentes roles que permiten la comunicación entre los diferentes actores, e implanta diferentes restricciones entre los roles. Estos roles son: (1) cliente, que sólo puede comunicarse con el agregador, cuando este le solicite el modelo entrenado; (2) agregador, que sólo puede pedirle a los clientes el modelo local entrenado y con el servidor para enviarle los modelos agregados; y (3) servidor, que sólo puede comunicarse con los clientes para darles el modelos global, o indicar a los clientes el modelo que se va a entrenar en el flujo de AF. Definidos estos roles, se pueden crear las diferentes arquitecturas propias del AF. FLEX incluye funciones que permiten crear las arquitecturas cliente-servidor y *peer-to-peer* directamente, como se muestra en Código 25.

```
# Client-server architecture
pool = flex.pool.FlexPool.client_server_pool(
     fed_dataset=federated_dataset,
     init_func=define_model,
     input_shape=train_dataset.shape[1],
     n_classes=len(set(train_dataset.y_data.to_numpy))
)
```

Código 25: Ejemplo de arquitecturas cliente-servidor y *peer-to-peer* usando FLEX.

Aunque la arquitectura cliente-servidor sea la más común, es posible que si se quiere simular una situación aún más distribuida, en la que los clientes están distribuidos en diversas partes del mundo, y es necesario que haya múltiples agregadores para poder recoger los modelos de los clientes reduciendo así los problemas

de latencia [BEG+19]. FLEX permite la creación de este tipo de arquitecturas como se muestra en Código 26.

```python
def real_scenario(node_ids: list, n_aggregators: int) -> FlexActors:
    if "server" in node_ids:
        raise ValueError("The name 'server' is reserved only for the server in a client-server architecture.")
    actors = FlexActors({node_ids[client_id]: FlexRole.client for client_id in node_ids})
    actors.update(
        FlexActors(
            {
                f"aggregator_{agg_id}": FlexRole.aggregator
                for agg_id in range(n_aggregators)
            }
        )
    )
    actors["server"] = FlexRole.server
    return actors

real_scenario_actors = real_scenario(list(federated_dataset.keys()), n_aggregators=4)
real_scenario_pool = flex.pool.FlexPool(federated_dataset, real_scenario_actors)
```

Código 26: Ejemplo de arquitectura real [BEG+19] usando FLEX.

Módulo *Pool* Tras haber federado los datos y haber creado la arquitectura y escenario de AF, se deben de crear las comunicaciones entre los diferentes nodos para poder llevar a cabo el flujo de AF, y entrenar el modelo. Para llevar a cabo las comunicaciones se debe crear una *pool*, `FlexPool`. Una *pool* es un conjunto de actores, datos y modelos, cada uno de ellos indexados por el id del actor, para poder acceder a los datos y modelos de cada actor. Para realizar estas comunicaciones se deben usar los métodos `select` y `map`. El método `select` permite seleccionar una *pool* más pequeña de otra más grande, a través de los identificadores o los roles de los actores. Un ejemplo de uso del método `select` es seleccionar el número de clientes que entrenarán el modelo en cada ronda, simulando así posibles caídas de conexión de clientes, o haciendo un filtrado por algún motivo que interesa al usuario.

El método `map` se ha diseñado para realizar las comunicaciones entre dos *pools* de actores, indicando primero la fuente y luego el destino. Gracias a este método se pueden establecer comunicaciones simuladas a bajo nivel entre los diferentes actores. Sin embargo, este proceso puede ser tedioso de manejar para el usuario, por lo que se decidió incluir múltiples *decoradores* que inhibieran al usuario de comprobar estas comunicaciones, y que se centre sólo en la creación de nuevos métodos de AF usando FLEX. Un ejemplo de realizar las comunicaciones se muestra en la siguiente subsección. Además de los *decoradores*, se han creado diferentes *primitivas* que agilizan todavía más el trabajo al usuario. Estas *primitivas* representan una funcionalidad completa, por ejemplo, FLEX incorpora el agregador FedAvg, para que el usuario no tenga que implementarlo, pero gracias a los *decoradores* se facilita mucho al usuario la adición de nuevos agregadores. Estas primitivas sirven tanto para añadir funcionalidad extra a FLEX, como de ejemplo de uso de los *decoradores* con FLEX para que los usuarios creen sus propias *primitivas*.

3.2. FLEX-Trees

FLEX es una *librería* que permite crear simulaciones para la experimentación de AF, actuando así como un núcleo central de AF, y permite la creación, adaptación y federación de modelos federados, pero no implementa modelos federados en sí. Para añadir más funcionalidades a FLEX, se han creado diversas librerías adyacentes a FLEX que complementan su funcionalidad. En concreto, el desarrollo de esta tesis ha llevado al desarrollo de la librería FLEX-Trees.

3.2.1. FLEX-Trees

Esta librería es una extensión de FLEX que contiene múltiples datasets tabulares muy utilizados en el estado del arte para árboles de decisión [LRBAG$^+$24], así como varios modelos de árboles decisión del estado del arte para AFH, como son:

1. Federated-ID3 (FedID3) [TBA$^+$19]: Implementa el ID3 clási-

co en un entorno AFH. Los clientes intercambian información acerca de las características que se están evaluando para dividir cada nodo, y se elige en cada momento la mejor característica evaluada con la métrica IG.

2. Federated Random Forest (FedRF) [HLM+22]: Implementa el RF clásico, en un entorno AFH. Cada cliente entrena un RF local, y luego el servidor seleccionará, aleatoriamente, n árboles de cada cliente k, los cuales agrupará para formar un RF global que tenga nxk árboles.

3. Federated Greadient Boosting Decision Tree (FedGBDT) [LWH20]: El algoritmo GBDT adaptado a un entorno AFH. Los clientes crean inicialmente una tabla hash global para alinear los datos que sean similares sin tener que compartir ningún tipo de datos privado. De esta forma se puede llevar a cabo la actualización de los pesos de las instancias locales teniendo en cuenta los pesos de las instancias del resto de clientes. Tras crear la tabla hash global, se entrenarán n árboles de decisión. El proceso de entrenamiento de los árboles es el siguiente: (1) Un cliente entrena un árbol con sus datos locales. (2) El cliente envía el árbol al servidor. (3) Se actualizan lo pesos de las instancias de los clientes y de la tabla hash global. Se continúa el proceso hasta tener los n árboles que conforman el *ensemble*.

Estos modelos sirven de ejemplo de implementación de cómo se pueden adaptar otros modelos de AD de AF a FLEX, en concreto a AFH, pero FLEX permite la distribución de datos para un entorno AFV, por lo que estos métodos se pueden añadir sin problema, y se añadirán en el futuro.

3.3. Ejemplo de entrenamiento de un árbol de decisión en Aprendizaje Federado usando FLEX-Trees

A continuación se va a mostrar un ejemplo de cómo usar FLEX-Trees para entrenar un modelo de AD en AFH. En el

Capítulo 1, en concreto en la Sección 1.4, se muestra un ejemplo de cómo entrenar un modelo usando el *framework* FATE, y este ejemplo servirá para hacer una comparativa de la usabilidad de ambos *frameworks*, con el objetivo de mostrar la simplicidad de FLEX-Trees de cara a entrenar modelos de árboles de decisión. El ejemplo que se va a mostrar es el FedRF [HLM+22], un RF sobre un escenario AFH. Para llevar a cabo este ejemplo, se va a seguir la metodología mostrada en el Capítulo 1:

1. *Tipo de problema*: AFH, se simula el entrenamiento de un modelo entre varios centros médicos, y todos comparten el espacio de características.
2. *Distribución federada*: [HLM+22] trabaja con múltiples datasets, en este caso se ha decidido trabajar con el dataset `ILDP`. [HLM+22] hace diferentes particiones de datos, pero en este ejemplo se va a mostrar una configuración *non-IID* básica.
3. *Selección del modelo*: En este caso se va a entrenar un RF. Para ello se creará inicialmente un RF a nivel de cliente.
4. *Estrategia de agregación*: La estrategia de agregación hecha en [HLM+22] consiste en seleccionar un número de árboles M de cada RF de cada cliente de forma aleatoria, para así quedarse con un total de N árboles en total, dando lugar al FedRF.
5. Estrategias de entrenamiento y evaluación: Cada cliente entrenará un RF con su dataset local, y lo evaluará. Tras recibir el modelo global, cada cliente lo evaluará sobre sus datos locales.

En el entorno federado AFH los clientes comparten el espacio de características del problema, pero no comparten el espacio de instancias. FLEX-Trees permite al usuario cargar múltiples datasets tabulares, los cuales no son inherentemente federados y es necesario utilizar FLEX para poder crear la distribución de datos entre los clientes. Para ello se ha optado por seguir una distribución non-IID, y sólo participarán en el entrenamiento del modelo dos clientes. La carga y distribución de datos se muestra en Código 27.

```
train_data, test_data = flextrees.datasets.tabular_datasets.ildp(
    ret_feature_names=False, categorical=False
)
config = flex.data.FedDatasetConfig(seed+0, n_nodes=2,
    replacement=False
)
weights = numpy.random.dirichlet(np.repeat(1, n_clients),1)[0]
federated_data = flex.data.FedDataDistribution.from_config(
    centralized_data=train_data, config=config, weights=weights
)
```

Código 27: Caso de uso con FLEX-Trees: Carga de datos

FLEX-Trees incorpora diferentes funciones primitivas, que implementan el funcionamiento necesario para poder simular el FedRF [HLM+22], ya que FLEX no contiene los modelos per sé, sino que contiene las primitivas que permiten simular y crear simulaciones de manera sencilla. Todas las primitivas necesarias para simular FedRF se cargan en Código 28. Además en Código 28, se crea la *pool* de actores, así como el número total de árboles que tendrá cada RF de cada cliente, y el número total de árboles del FedRF.

```
from flextrees.pool import
↪  (init_server_model_rf,deploy_server_config_rf,
    deploy_server_model_rf, aggregate_trees_from_rf,
    ↪  evaluate_global_rf_model,
    evaluate_global_rf_model_at_clients,
    ↪  evaluate_local_rf_model_at_clients,
    train_rf, collect_clients_trees_rf, set_aggregated_trees_rf,
)
pool = flex.pool.FlexPool(federated_data, init_server_model_rf)
# Number of estimators of the FedRF
total_estimators = 20
# Number of trees to select from each client
nr_estimators = total_estimators // 2
```

Código 28: Caso de uso con FLEX-Trees: Importar las primitivas

Por último sólo queda usar las primitivas, que van desde el despliegue del modelo local, pasando por el entrenamiento del mo-

delo local, agregación de los modelos locales, despliegue del FedRF a los clientes, y evaluación de FedRF sobre los datasets locales de los clientes, así como sobre un dataset de validación global. Este proceso se muestra en Código 29.

```
1  # Despliegue del modelo
2  pool.server.map(func=deploy_server_config_rf,
   ↪  dst_pool=pool.clients)
3  # Entrenamiento del modelo local
4  pool.clients.map(func=train_rf)
5  pool.clients.map(func=evaluate_local_rf_model_at_clients)
6  # Agregación de los modelos locales
7  pool.aggregator.map(func=collect_clients_trees_rf,
   ↪  dst_pool=pool.clients, nr_estimators=nr_estimators)
8  pool.aggregator.map(func=aggregate_trees_from_rf)
9  pool.aggregator.map(func=set_aggregated_trees_rf,
   ↪  dst_pool=pool.servers)
10 # Despliegue del FedRF
11 pool.server.map(func=deploy_server_model_rf,
   ↪  dst_pool=pool.clients)
12 # Evaluación del modelo
13 pool.server.map(func=evaluate_global_rf_model,
   ↪  test_data=test_data)
14 pool.clients.map(func=evaluate_global_rf_model_at_clients)
```

Código 29: Caso de uso con FLEX-Trees: Flujo de entrenamiento de FedRF usando las primitivas de FLEX-Trees

En la línea 2 el servidor indica a los clientes qué modelo se va a entrenar de manera local. En las líneas 4 y 5, los clientes entrenan y evalúan un RF sobre su dataset local. Las líneas 7-9 recogen, seleccionan y agregan los RF de los clientes, creando así el FedRF. En la línea 11, el servidor manda a los clientes el modelo FedRF. Por último, en la línea 13 el servidor evalúa FedRF sobre un dataset de validación global, y en la línea 14 cada cliente evalúa FedRF sobre su dataset local, para así comprobar que el modelo global mejora el rendimiento de los modelos locales propios.

3.4. Conclusiones finales

FLEX-Trees proporciona modelos de interpretabilidad como los árboles de decisión, los cuales son necesarios en entornos

de AF para poder aplicarlos en situaciones reales. FLEX-Trees permite la integración de modelos interpretables de forma sencilla para que así los usuarios puedan realizar un diseño de nuevos modelos en múltiples entornos de AF antes de usarlos en una situación real. Se ha mostrado la capacidad de FLEX-Trees a través de varios modelos interpretables del estado del arte.

Es importante denotar que los frameworks mostrados en el Capítulo 1, en la Sección 1.3, carecen de modelos interpretables en su gran mayoría, además de no disponer facilidades a la hora de integrar o simular la experimentación con este tipo de modelos. Esto dificulta el avance en la creación de nuevos modelos interpretables novedosos, y FLEX-Trees proporciona una solución a esta carencia de los *frameworks* actuales.

4 Comentarios finales y trabajos futuros

4.1. Conclusiones

La tesis está compuesta de tres objetivos principales que se han llevado a cabo de forma progresiva a través de los capítulos previos.

- Inicialmente se realizó una metodología de trabajo para AF que permite unificar el proceso de entrenamiento de modelos en los diferentes entornos de AF, y que ayudará a agilizar el proceso de creación de métodos novedosos para AF.

- Se ha presentado un proceso de agregación novedoso para AD en entornos de AF llamado ICDTA4FL. Este proceso supera a los métodos del estado del arte que construyen un solo árbol. El proceso ICDTA4FL es independiente del algoritmo de AD utilizado, y es capaz de agregar diferentes tipos de AD como ID3 y CART, mientras mantiene la interpretabilidad y la estructura de los árboles usados. ICDTA4FL se ha realizado sobre un entorno de AFH, en ambas distribuciones IID y non-IID, variando el número de clientes. Además se probado el rendimiento de ICDTA4FL usando los árboles ID3 y CART, obteniendo los modelos ICDTA4FL-ID3 y ICDTA4FL-CART.

El proceso ICDTA4FL presenta las siguientes ventajas:

- Funciona con múltiples tipos de AD.
- Logra un rendimiento excelente en distribuciones de datos IID y non-IID.
- Mantiene la estructura jerárquica y la interpretabilidad de los AD originales utilizados para construir el árbol global, siendo capaz de generalizar el conocimiento de los nodos.

- Por último se ha llevado a cabo la creación de un *framework* de AF para modelos interpretables, FLEX-Trees, que permite la simulación de experimentos de AF de forma rápida y sencilla, dejando al usuario que se centre en la creación de métodos novedosos, y que además pueda llevar a cabo la metodología propuesta.

4.2. Publicaciones

Esta tesis está vinculada a las publicaciones que se han presentado en los capítulos 1 y 3, y que se indican a continuación:

1. Los resultados obtenidos en el primer capítulo han sido publicados en la revista **IEEE/CAA Journal of Automatica Sinica**, situada en el primer cuartil en las categorías *Computer Science, Information Systems* y *Computer Science, Artificial Ingelligence* con ranking 13 y con factor de impacto (JCR 2022) 11.8:
   Luzón, M. V., Rodríguez-Barroso, N., Argente-Garrido, A., Jiménez-López, D., Moyano, J., Del Ser, J., Ding, Weiping, Herrera, F., "A Tutorial on Federated Learning from Theory to Practice: Foundations, Software Frameworks, Exemplary Use Cases, and Selected Trends", in IEEE/CAA Journal of Automatica Sinica, vol. 11, no. 4, pp. 824-850, April 2024, doi: 10.1109/JAS.2024.124215.

2. Los resultados obtenidos en el segundo capítulo se encuentran en proceso de revisión en la revista **Information Scien-**

ces, con el título *An Interpretable Client Decision Tree Aggregation process for Federated Learning*.

4.3. Trabajo futuro

Los estudios realizados han permitido la creación de una metodología de trabajo para el entrenamiento de modelos en AF, así como un *framework* para la simulación de experimentos en AF. Además se ha propuesto un proceso de agregación de AD para AF, ICDTA4FL, sobre el cual se plantea varias extensiones directas.

- Extender el proceso ICDTA4FL a entornos de AFV, donde ya se encuentran múltiples enfoques de AD, sólo que basados en técnicas de *bagging* y *boosting*.
- Aplicar técnicas de *boosting* en el proceso ICDTA4FL. Actualmente se ha mostrado el funcionamiento del modelo para la construcción de un solo AD, pero se plantea el uso de técnicas de *boosting* que ofrezcan al modelo final un mayor rendimiento.

Appendix A: Abreviaciones

En este apéndice se incluye la lista de abreviaciones utilizadas a lo largo de la presente memoria de tesis:

ICDTA4FL Client Decision Tree Aggregation for Federated Learning process

IA Inteligencia Artificial

FedAvg Federated Averaging

AF Aprendizaje Federado

AA Aprendizaje Automático

AAC Aprendizaje Automático Centralizado

IID Independent and Identically Distributed

RF Random Forest

XGB XGBoost

GBDT Gradient Boosting Decision Trees

AD Árbol de Decisión

AP Aprendizaje Profundo

non-IID Non Independent and Identically Distributed

AFH Aprendizaje Federado Horizontal

AFV Aprendizaje Federado Vertical

AFT Aprendizaje Federado de Transferencia

SMC Secure Multiparty Computation

HE Homomorphic Encryption

TAI Trustworthy Artificial Intelligence

PD Privacidad Diferencial

PDL Privacidad Diferencial Local

PDC Privacidad Diferencial Central

TFF TensorFlow Federated

VC Visión por Computador

CNN Convolutional Neural Networks

RNN Recurrent Neural Networks

PLN Procesamiento del Lenguaje Natural

AO Análisis de Opiniones

Bibliografía

[AASB+23] Alzubaidi L., Al-Sabaawi A., Bai J., Dukhan A., Alkenani A., al Asadi A., Al-Wzwazy H., Manoufali M., Fadhel M., Albahri A., Moreira C., Ouyang C., Zhang J., Santamaría J., Salhi A., Hollman F., Gupta A., Duan Y., Rabczuk T., and Gu Y. (10 2023) Towards risk-free trustworthy artificial intelligence: Significance and requirements. *International Journal of Intelligent Systems* 2023: 41.

[Act21] Act A. I. (2021) Proposal for a regulation of the european parliament and the council laying down harmonised rules on artificial intelligence (artificial intelligence act) and amending certain union legislative acts. *EUR-Lex-52021PC0206* .

[ADRD+20] Arrieta A. B., Díaz-Rodríguez N., Del Ser J., Bennetot A., Tabik S., Barbado A., Garcia S., Gil-Lopez S., Molina D., Benjamins R., Chatila R., and Herrera F. (2020) Explainable artificial intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI. *Information Fusion* 58: 82–115.

[AGZLH24] Argente-Garrido A., Zuheros C., Luzón M. V., and Herrera F. (2024) An interpretable client decision tree aggregation process for federated learning.

[AKL21] Agarwal N., Kairouz P., and Liu Z. (2021) The skellam mechanism for differentially private fede-

rated learning. *Advances in Neural Information Processing Systems* 34: 5052–5064.

[ATMR21] Andrew G., Thakkar O., McMahan B., and Ramaswamy S. (2021) Differentially private learning with adaptive clipping. In *Advances in Neural Information Processing Systems*, volumen 34, pp. 17455–17466.

[BBG+20] Bell J. H., Bonawitz K. A., Gascón A., Lepoint T., and Raykova M. (2020) Secure single-server aggregation with (poly) logarithmic overhead. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, pp. 1253–1269.

[BEG+19] Bonawitz K., Eichner H., Grieskamp W., Huba D., Ingerman A., Ivanov V., Kiddon C., Konečnỳ J., Mazzocchi S., McMahan B., *et al.* (2019) Towards federated learning at scale: System design. *Proceedings of machine learning and systems* 1: 374–388.

[BMR20] Bhatore S., Mohan L., and Reddy Y. R. (2020) Machine learning techniques for credit risk evaluation: a systematic literature review. *Journal of Banking and Financial Technology* 4: 111–138.

[Bre01] Breiman L. (2001) Random forests. *Machine learning* 45: 5–32.

[Bre17] Breiman L. (2017) *Classification and regression trees*. Routledge.

[BTM+20] Beutel D. J., Topal T., Mathur A., Qiu X., Fernandez-Marques J., Gao Y., Sani L., Kwing H. L., Parcollet T., Gusmão P. P. d., and Lane N. D. (2020) Flower: A friendly federated learning research framework. *arXiv preprint arXiv:2007.14390* .

[BVH+20] Bagdasaryan E., Veit A., Hua Y., Estrin D., and Shmatikov V. (2020) How to backdoor federated

learning. In *International Conference on Artificial Intelligence and Statistics*, pp. 2938–2948. PMLR.

[CCA+21] Chai Z., Chen Y., Anwar A., Zhao L., Cheng Y., and Rangwala H. (2021) Fedat: a high-performance and communication-efficient federated learning system with asynchronous tiers. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1–16.

[CCC22] Caldarola D., Caputo B., and Ciccone M. (2022) Improving generalization in federated learning by seeking flat minima. In *European Conference on Computer Vision*, pp. 654–672. Springer.

[CDW+19] Caldas S., Duddu S. M. K., Wu P., Li T., Konečný J., Mcmahan H. B., Smith V., and Talwalkar A. (December 2019) Leaf: A benchmark for federated settings. In *International Workshop on Federated Learning for User Privacy and Data Confidentiality in Conjunction with NeurIPS 2019 (FL-NeurIPS'19)*. Vancouver, BC, Canada.

[CFJ+21a] Cheng K., Fan T., Jin Y., Liu Y., Chen T., Papadopoulos D., and Yang Q. (2021) Secureboost: A lossless federated learning framework. *IEEE Intelligent Systems* 36(6): 87–98.

[CFJ+21b] Cheng K., Fan T., Jin Y., Liu Y., Chen T., Papadopoulos D., and Yang Q. (2021) SecureBoost: A lossless federated learning framework. *IEEE Intelligent Systems* 36(6): 87–98.

[CG16] Chen T. and Guestrin C. (2016) Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16. ACM.

[CGH+21] Charles Z., Garrett Z., Huo Z., Shmulyian S., and Smith V. (2021) On large-cohort training for fe-

derated learning. *Advances in neural information processing systems* 34: 20461–20475.

[CHMS21] Collins L., Hassani H., Mokhtari A., and Shakkottai S. (2021) Exploiting shared representations for personalized federated learning. In *International Conference on Machine Learning*, pp. 2089–2099. PMLR.

[CLB+21] Chen L., Li S., Bai Q., Yang J., Jiang S., and Miao Y. (2021) Review of image classification algorithms based on convolutional neural networks. *Remote Sensing* 13(22): 4712.

[CMF+21] Chen W., Ma G., Fan T., Kang Y., Xu Q., and Yang Q. (2021) Secureboost+: A high performance gradient boosting tree framework for large scale vertical federated learning. *arXiv preprint arXiv:2110.10927* .

[CSP+21] Cha D., Sung M., Park Y.-R., *et al.* (2021) Implementing vertical federated learning using autoencoders: Practical application, generalizability, and utility study. *JMIR medical informatics* 9(6): 26598.

[CZG+21] Chen X., Zhou S., Guan B., Yang K., Fao H., Wang H., and Wang Y. (2021) Fed-eini: An efficient and interpretable inference framework for decision tree ensembles in vertical federated learning. In *2021 IEEE International Conference on Big Data (Big Data)*, pp. 1242–1248. IEEE.

[DMNS06] Dwork C., McSherry F., Nissim K., and Smith A. (2006) Calibrating noise to sensitivity in private data analysis. In *Theory of Cryptography*, pp. 265–284.

[DRDC+23] Díaz-Rodríguez N., Del Ser J., Coeckelbergh M., López de Prado M., Herrera-Viedma E., and Herrera F. (2023) Connecting the dots in trustworthy artificial intelligence: From AI principles, ethics,

and key requirements to responsible AI systems and regulation. *Information Fusion* 99: 101896.

[Eur19] European Commission, High-level expert group on artificial intelligence (2019) *Ethics Guidelines for Trustworthy AI*. European Union.

[GGD+20] González S., García S., Del Ser J., Rokach L., and Herrera F. (2020) A practical tutorial on bagging and boosting based ensembles for machine learning: Algorithms, software tools, performance study, practical perspectives and opportunities. *Information Fusion* 64: 205–237.

[GSC22] Gencturk M., Sinaci A. A., and Cicekli N. K. (2022) Bofrf: A novel boosting-based federated random forest algorithm on horizontally partitioned data. *IEEE Access* 10: 89835–89851.

[GSK+22] Gong X., Sharma A., Karanam S., Wu Z., Chen T., Doermann D., and Innanje A. (2022) Preserving privacy in federated learning with ensemble cross-domain knowledge distillation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volumen 36, pp. 11891–11899.

[GXP+20] Gong M., Xie Y., Pan K., Feng K., and Qin A. (2020) A survey on differentially private machine learning [review article]. *IEEE Computational Intelligence Magazine* 15(2): 49–64.

[HAA20] He C., Annavaram M., and Avestimehr S. (2020) Group knowledge transfer: Federated learning of large cnns at the edge. *Advances in Neural Information Processing Systems* 33: 14068–14080.

[HCB+21] Hilmkil A., Callh S., Barbieri M., Sütfeld L. R., Zec E. L., and Mogren O. (2021) Scaling federated learning for fine-tuning of large language models. In *International Conference on Applications of Natural Language to Information Systems*, pp. 15–23. Springer.

[HJLAG+24] Herrera F., Jiménez-López D., Argente-Garrido A., Rodríguez-Barroso N., Zuheros C., Aguilera-Martos I., Bello B., García-Márquez M., and Luzón M. V. (2024) Flex: Flexible federated learning framework.

[HLM+22] Hauschild A.-C., Lemanczyk M., Matschinske J., Frisch T., Zolotareva O., Holzinger A., Baumbach J., and Heider D. (2022) Federated Random Forests can improve local performance of predictive models for various healthcare applications. *Bioinformatics* 38(8): 2278–2286.

[HQB20] Hsu T.-M. H., Qi H., and Brown M. (2020) Federated visual classification with real-world data distribution. In *European Conference on Computer Vision*, pp. 76–92. Springer.

[KMF+17] Ke G., Meng Q., Finley T., Wang T., Chen W., Ma W., Ye Q., and Liu T.-Y. (2017) LightGBM: A highly efficient gradient boosting decision tree. In *Advances in Neural Information Processing Systems*, volumen 30. Curran Associates, Inc.

[KMY+16] Konečnỳ J., McMahan H. B., Yu F. X., Richtárik P., Suresh A. T., and Bacon D. (2016) Federated learning: Strategies for improving communication efficiency. *arXiv preprint arXiv:1610.05492* .

[KS22] Kesanapalli S. A. and Simmhan Y. (2022) Characterizing the performance of accelerated jetson edge devices for training deep learning models. *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 6(3): 1–26.

[KT21] Kasyap H. and Tripathy S. (2021) Privacy-preserving decentralized learning framework for healthcare system. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)* 17(2s): 1–24.

[KZPP+21] Kaissis G., Ziller A., Passerat-Palmbach J., Ryffel T., Usynin D., Trask A., Lima I., Mancuso J., Jungmann F., Steinborn M.-M., *et al.* (2021) End-to-end privacy preserving deep learning on multi-institutional medical imaging. *Nature Machine Intelligence* 3(6): 473–484.

[LDCH22] Li Q., Diao Y., Chen Q., and He B. (2022) Federated learning on non-iid data silos: An experimental study. In *2022 IEEE 38th International Conference on Data Engineering (ICDE)*, pp. 965–978.

[LDS+22] Lai F., Dai Y., Singapuram S., Liu J., Zhu X., Madhyastha H., and Chowdhury M. (2022) Fedscale: Benchmarking model and system performance of federated learning at scale. In *International Conference on Machine Learning*, pp. 11814–11827. PMLR.

[LHL+22a] Li X., Hu Y., Liu W., Feng H., Peng L., Hong Y., Ren K., and Qin Z. (2022) Opboost: a vertical federated tree boosting framework based on order-preserving desensitization. *Proceedings of the VLDB Endowment* 16(2): 202–215.

[LHL+22b] Li X., Hu Y., Liu W., Feng H., Peng L., Hong Y., Ren K., and Qin Z. (2022) OpBoost: a vertical federated tree boosting framework based on order-preserving desensitization. *Proc. VLDB Endow.* 16(2): 202–215.

[LKSJ20a] Lin T., Kong L., Stich S. U., and Jaggi M. (2020) Ensemble distillation for robust model fusion in federated learning. In Larochelle H., Ranzato M., Hadsell R., Balcan M., and Lin H. (Eds.) *Advances in Neural Information Processing Systems*, volumen 33, pp. 2351–2363. Curran Associates, Inc.

[LKSJ20b] Lin T., Kong L., Stich S. U., and Jaggi M. (2020) Ensemble distillation for robust model fusion in federated learning. *Advances in Neural Information Processing Systems* 33: 2351–2363.

[LKX+20] Liu Y., Kang Y., Xing C., Chen T., and Yang Q. (2020) A secure federated transfer learning framework. *IEEE Intelligent Systems* 35(4): 70–82.

[LLL+22] Liu Y., Liu Y., Liu Z., Liang Y., Meng C., Zhang J., and Zheng Y. (2022) Federated forest. *IEEE Transactions on Big Data* 8(3): 843–854.

[LM20] Liu D. and Miller T. (2020) Federated pretraining and fine tuning of bert using clinical notes from multiple silos. *arXiv preprint arXiv:2002.08562* .

[LMM+22] Li J., Meng Y., Ma L., Du S., Zhu H., Pei Q., and Shen X. (2022) A federated learning based privacy-preserving smart healthcare system. *IEEE Transactions on Industrial Informatics* 18(3): 2021–2031.

[LMY+22] Liu Y., Ma Z., Yang Y., Liu X., Ma J., and Ren K. (2022) Revfrf: Enabling cross-domain random forest training with revocable federated learning. *IEEE Transactions on Dependable and Secure Computing* 19(6): 3671–3685.

[LRBAG+24] Luzón M. V., Rodríguez-Barroso N., Argente-Garrido A., Jiménez-López D., Moyano J. M., Del Ser J., Ding W., and Herrera F. (2024) A tutorial on federated learning from theory to practice: Foundations, software frameworks, exemplary use cases, and selected trends. *IEEE/CAA Journal of Automatica Sinica* 11(4): 824–850.

[LWH20] Li Q., Wen Z., and He B. (2020) Practical federated gradient boosting decision trees. In *Proceedings of the AAAI conference on artificial intelligence*, volumen 34, pp. 4642–4649.

[LWS21] Lian Z., Wang W., and Su C. (2021) Cofel: communication-efficient and optimized federated learning with local differential privacy. In *ICC 2021-IEEE International Conference on Communications*, pp. 1–6. IEEE.

[LZS+20] Li W., Zhu L., Shi Y., Guo K., and Cambria E. (2020) User reviews: Sentiment analysis using lexicon integrated two-channel cnn–lstm family models. *Applied Soft Computing* 94: 106435.

[mel19] (2019) MELLODY: MachinE Learning Ledger Orchestration for Drug DiscoverY.

[MKT22] Manna D., Kasyap H., and Tripathy S. (2022) Milsa: Model interpretation based label sniffing attack in federated learning. In *International Conference on Information Systems Security*, pp. 139–154. Springer.

[MMR+17] McMahan B., Moore E., Ramage D., Hampson S., and Arcas B. A. y. (20–22 Apr 2017) Communication-Efficient Learning of Deep Networks from Decentralized Data. In Singh A. and Zhu J. (Eds.) *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, volumen 54 of *Proceedings of Machine Learning Research*, pp. 1273–1282. PMLR.

[MNG+17] Marjani M., Nasaruddin F., Gani A., Karim A., Hashem I. A. T., Siddiqa A., and Yaqoob I. (2017) Big iot data analytics: Architecture, opportunities, and open research challenges. *IEEE Access* 5: 5247–5261.

[MXNV20] Marfoq O., Xu C., Neglia G., and Vidal R. (2020) Throughput-optimal topology design for cross-silo federated learning. *Advances in Neural Information Processing Systems* 33: 19478–19487.

[NHDC22] Naseri M., Hayes J., and De Cristofaro E. (2022) Local and central differential privacy for robustness and privacy in federated learning. In *Proceedings of the 29th Network and Distributed System Security Symposium (NDSS)*, pp. 24–28.

[QCG+24] Qi P., Chiaro D., Guzzo A., Ianni M., Fortino G., and Piccialli F. (2024) Model aggregation techni-

ques in federated learning: A comprehensive survey. *Future Generation Computer Systems* 150: 272–293.

[Qui86] Quinlan J. R. (1986) Induction of decision trees. *Machine Learning* 1(1): 81–106.

[QWW+22] Qi T., Wu F., Wu C., Lyu L., Xu T., Liao H., Yang Z., Huang Y., and Xie X. (2022) Fairvfl: A fair vertical federated learning framework with contrastive adversarial learning. *Advances in Neural Information Processing Systems* 35: 7852–7865.

[QZL+22] Qu L., Zhou Y., Liang P. P., Xia Y., Wang F., Adeli E., Fei-Fei L., and Rubin D. (2022) Rethinking architecture design for tackling data heterogeneity in federated learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 10061–10071.

[RBJLL+23] Rodríguez-Barroso N., Jiménez-López D., Luzón M. V., Herrera F., and Martínez-Cámara E. (2023) Survey on federated learning threats: concepts, taxonomy on attacks and defences, experimental study and challenges. *Information Fusion* 90: 148–173.

[RBMCLH22a] Rodríguez-Barroso N., Martínez-Cámara E., Luzón M. V., and Herrera F. (2022) Backdoor attacks-resilient aggregation based on robust filtering of outliers in federated learning for image classification. *Knowledge-Based Systems* 245: 108588.

[RBMCLH22b] Rodríguez-Barroso N., Martínez-Cámara E., Luzón M. V., and Herrera F. (2022) Dynamic defense against byzantine poisoning attacks in federated learning. *Future Generation Computer Systems* 133: 1–9.

[RBSJL+20] Rodríguez-Barroso N., Stipcich G., Jiménez-López D., Ruiz-Millán J. A., Martínez-Cámara E.,

González-Seco G., Luzón M. V., Veganzones M. A., and Herrera F. (2020) Federated learning and differential privacy: Software tools analysis, the sherpa.ai fl framework and methodological guidelines for preserving data privacy. *Information Fusion* 64: 270–292.

[RM14] Rokach L. and Maimon O. (2014) *Data Mining With Decision Trees: Theory and Applications*. World Scientific Publishing Co., Inc., USA, 2nd edition.

[RVTM22] Radhakrishnan J. K. K., Verma A., Thomas G., and Muthusamy V. (2022) Just-in-time aggregation for federated learning. In *IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*, pp. 1–8.

[SPP21] Siomos V. and Passerat-Palmbach J. (2021) Contribution evaluation in federated learning: Examining current approaches. In *1st NeurIPS Workshop on New Frontiers in Federated Learning (NFFL 2021)*.

[SR20] Sagi O. and Rokach L. (2020) Explainable decision forest: Transforming a decision forest into an interpretable tree. *Information Fusion* 61: 124 – 138.

[SSG+21] Singhal K., Sidahmed H., Garrett Z., Wu S., Rush J., and Prakash S. (2021) Federated reconstruction: Partially local federated learning. *Advances in Neural Information Processing Systems* 34: 11220–11232.

[SZHL23] Soltani B., Zhou Y., Haghighi V., and Lui J. C. S. (8 2023) A survey of federated evaluation in federated learning. In Elkind E. (Ed.) *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence, IJCAI-23*, pp. 6769–

6777. International Joint Conferences on Artificial Intelligence Organization.

[TBA+19] Truex S., Baracaldo N., Anwar A., Steinke T., Ludwig H., Zhang R., and Zhou Y. (2019) A hybrid approach to privacy-preserving federated learning. In *Proceedings of the 12th ACM Workshop on Artificial Intelligence and Security*, page 1–11. Association for Computing Machinery.

[TG20] Tjoa E. and Guan C. (2020) A survey on explainable artificial intelligence (xai): Toward medical xai. *IEEE transactions on neural networks and learning systems* 32(11): 4793–4813.

[The18] The TensorFlow Federated Authors (Google) (2018) Tensorflow federated.

[TLS20] Thiebes S., Lins S., and Sunyaev A. (10 2020) Trustworthy artificial intelligence. *Electronic Markets* 31.

[WCP20] Wang S., Cao J., and Philip S. Y. (2020) Deep learning for spatio-temporal data mining: A survey. *IEEE transactions on knowledge and data engineering* 34(8): 3681–3700.

[WCX+20] Wu Y., Cai S., Xiao X., Chen G., and Ooi B. C. (2020) Privacy preserving vertical federated learning for tree-based models. *Proceedings of the VLDB Endowment* 13(11): 2090–2103.

[WKNL20] Wang H., Kaplan Z., Niu D., and Li B. (2020) Optimizing federated learning on non-iid data with reinforcement learning. In *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*, pp. 1698–1707. IEEE.

[WZFY20] Wang T., Zhang X., Feng J., and Yang X. (Dec 2020) A comprehensive survey on local differential privacy toward data statistics and analysis. *Sensors* 20(24): 7030.

[YLC+19] Yang Q., Liu Y., Cheng Y., Kang Y., Chen T., and Yu H. (2019) *Federated learning.* Morgan & Claypool Publishers.

[YOW22] Yamamoto F., Ozawa S., and Wang L. (2022) eFL-Boost: Efficient federated learning for gradient boosting decision trees. *IEEE Access* 10: 43954–43963.

[ZDL+21] Zhang Y., Duan M., Liu D., Li L., Ren A., Chen X., Tan Y., and Wang C. (2021) Csafl: A clustered semi-asynchronous federated learning framework. In *2021 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–10. IEEE.

[ZLL+22] Zheng Y., Lai S., Liu Y., Yuan X., Yi X., and Wang C. (2022) Aggregation service for federated learning: An efficient, secure, and more resilient realization. *IEEE Transactions on Dependable and Secure Computing* pp. 988–1001.

[ZLX+20] Zhang C., Li S., Xia J., Wang W., Yan F., and Liu Y. (2020) Batchcrypt: Efficient homomorphic encryption for cross-silo federated learning. In *2020 USENIX annual technical conference (USENIX ATC 20)*, pp. 493–506.

[ZXLJ21] Zhu H., Xu J., Liu S., and Jin Y. (2021) Federated learning on non-iid data: A survey. *Neurocomputing* 465: 371–390.