



TRABAJO FIN DE GRADO
INGENIERÍA EN INFORMÁTICA

Estrategias de selección de clientes en aprendizaje federado

Un análisis comparativo de estrategias de selección y estudio de su impacto en el entrenamiento de modelos de ML

Autor

César Alberto Mayora Suárez (alumno)

Directores

Francisco Herrera Triguero (tutor1)
María Victoria Luzón García (tutor2)



ESCUELA TÉCNICA SUPERIOR DE INGENIERÍAS INFORMÁTICA Y DE
TELECOMUNICACIÓN

—
Granada, septiembre de 2024

Estrategias de selección de clientes en aprendizaje federado: Un análisis comparativo de estrategias de selección y estudio de su impacto en el entrenamiento de modelos de ML.

César Alberto Mayora Suárez

Palabras clave: Aprendizaje Federado, Selección de Clientes, Estrategias de Selección, ML distribuido, Machine Learning, Clientes Heterogéneos, Análisis Comparativo.

Resumen

Cuando en un escenario de aprendizaje federado participan pocos clientes, es factible incorporar sus parámetros en cada ronda de entrenamiento. Sin embargo, a medida que aumenta el número de clientes, también lo hace la sobrecarga de comunicación, de modo que considerar los parámetros de todos los clientes se convierte en un reto. Al mismo tiempo, cuando el número de clientes es elevado, algunos de ellos pueden tener acceso a datos redundantes, ruidosos o menos valiosos que otros. Por lo tanto, se han introducido métodos de selección de clientes para reducir el número de participantes en cada ronda de entrenamiento. En este TFG se propone un análisis exhaustivo de algunos de estos métodos de selección de clientes en el entrenamiento de modelos de ML con aprendizaje federado que han mostrado buenos resultados en la literatura, presentando características, diseños y enfoques distintos para entrenar modelos de aprendizaje automático en entornos de aprendizaje federado. Hemos desarrollado un entorno experimental en el que evaluamos estos algoritmos de selección de clientes y los comparamos entre sí, destacando sus fortalezas y debilidades en cuanto a la convergencia del modelo y la eficiencia en el uso de los recursos de los clientes. Además, se ha propuesto una mejora sobre uno de estos algoritmos, que hemos denominado Dyn-HybridFL, que implementa la gestión dinámica de uno de los hiperparámetros del algoritmo base, HybridFL. Los resultados experimentales muestran que nuestra propuesta mejora tanto la convergencia como la eficiencia en el uso de recursos durante el entrenamiento de los modelos en aprendizaje federado.

Client Selection strategies in federated learning: A comparative analysis of selection strategies and study of their impact on ML model training.

César Alberto Mayora Suárez

Keywords: Federated Learning, Client Selection, Selection Strategies, Distributed Machine Learning, Machine Learning, Heterogeneous Clients, Comparative Analysis.

Abstract

When a federated learning scenario involves few clients, it is feasible to incorporate their parameters in each training round. However, as the number of clients increases, so does the communication overhead, so considering the parameters of all clients becomes a challenge. At the same time, when the number of clients is high, some of them may have access to redundant, noisy, or less valuable data than others. Therefore, client selection methods have been introduced to reduce the number of participants in each training round. This Bachelor's thesis proposes a comprehensive analysis of some of these client selection methods in training ML models with federated learning that have shown good results in the literature, presenting distinct features, designs, and approaches to training machine learning models in federated learning environments. We have developed an experimental setting in which we evaluate these client selection algorithms and compare them with each other, highlighting their strengths and weaknesses in terms of model convergence and efficiency in the use of clients' resources. Moreover, we have proposed an improvement on one of these algorithms, which we have named Dyn-HybridFL, which implements the dynamic management of one of the hyperparameters of the base algorithm, HybridFL. Experimental results show that our proposal improves both convergence and resource efficiency during training of models in federated learning.

Yo, **César Alberto Mayora Suárez**, alumno de la titulación Ingeniería Informática de la **Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación de la Universidad de Granada**, con DNI 35728625B, autorizo la ubicación de la siguiente copia de mi Trabajo Fin de Grado en la biblioteca del centro para que pueda ser consultada por las personas que lo deseen.

Fdo: César Alberto Mayora Suárez

Granada a 6 de septiembre de 2024.

D. **Francisco Herrera Triguero**, Profesor del Área de Ciencias de la Computación e Inteligencia Artificial del Departamento de Ciencias de la Computación e Inteligencia Artificial de la Universidad de Granada.

D. **María Victoria Luzón García**, Profesor del Área de Lenguajes y Sistemas Informáticos del Departamento de Lenguajes y Sistemas Informáticos de la Universidad de Granada.

Informan:

Que el presente trabajo, titulado *Estrategias de selección de clientes en aprendizaje federado: Un análisis comparativo de estrategias de selección y estudio de su impacto en el entrenamiento de modelos de ML*, ha sido realizado bajo su supervisión por **César Alberto Mayora Suárez**, y autorizamos la defensa de dicho trabajo ante el tribunal que corresponda.

Y para que conste, expiden y firman el presente informe en Granada a 6 de septiembre de 2024.

Los directores:

Francisco Herrera Triguero María Victoria Luzón García

Agradecimientos

A mis padres, que me apoyaron de forma incondicional en todos estos años de carrera.

Índice general

Índice general	I
Índice de figuras	III
Índice de tablas	VII
1. Introducción	1
1.1. Motivación del trabajo	2
1.2. Objetivos del estudio	4
1.3. Requisitos y Casos de Uso	5
1.4. Planificación y costo del proyecto	6
1.5. Estructura del documento	10
2. Fundamentos Teóricos	13
2.1. Aprendizaje Automático (ML)	14
2.1.1. ¿Como funciona el Aprendizaje Automático?	15
2.1.2. Tipos de aprendizaje	16
2.1.3. El problema del aprendizaje	17
2.1.4. Descenso del Gradiente	19
2.1.5. Redes Neuronales	21
2.1.6. Redes Neuronales Convolucionales (CNN)	23
2.2. Aprendizaje Federado (FL)	25
2.2.1. Notación y formulación	28
2.3. Selección de clientes en FL	29
3. Estado del Arte	33
3.1. Consideraciones sobre los tipos de selección	33
3.2. Taxonomía de métodos de selección de clientes	35
3.3. Clasificación basada en el problema a resolver	39
4. Metodología	43
4.1. Enfoque del estudio	44
4.2. Datasets utilizados	45
4.2.1. Formulación del problema de aprendizaje	47

4.2.2. Preprocesamiento y tratamiento de los datos	48
4.3. Modelos de Aprendizaje	49
4.3.1. Función de pérdida	54
4.3.2. Optimizador <i>Adam</i>	54
4.4. Herramientas utilizadas	55
4.4.1. FLEXible Framework	56
4.5. Estrategias de selección de clientes	60
4.5.1. Active Federated Learning	60
4.5.2. Greedy Shapley Client Selection	64
4.5.3. HybridFL	70
4.5.4. Dyn-HybridFL	78
4.6. Diseño de los experimentos	82
4.6.1. Experimento enfocado en el rendimiento	82
4.6.2. Experimento enfocado en los recursos	85
5. Implementación	87
5.1. Estructura del código	87
5.2. Procedimiento de implementación	89
5.3. Ejecución del entrenamiento de los modelos	90
5.3.1. Usando Google Colab (recomendado)	90
5.3.2. Ejecución en local	90
6. Experimentos y Resultados	93
6.1. Distribución de los datos	94
6.2. Configuración para tareas de ML	95
6.3. Experimentos de métodos basados en rendimiento	96
6.3.1. Análisis de los resultados	100
6.4. Experimentos de métodos basados en recursos	106
6.4.1. Análisis de los resultados	110
7. Conclusiones y Futuros Trabajos	115
8. Bibliografía	119

Índice de figuras

1.1. Esquema de una iteración en un escenario de aprendizaje federado centralizado. El servidor se encarga de distribuir el modelo global mientras que los clientes actualizan este modelo con sus datos locales; finalmente el servidor agrega estas actualizaciones del modelo para constituir el nuevo modelo global de la siguiente iteración.	2
1.2. Diagrama de Gantt de la planificación temporal teórica para la realización del TFG sobre las fases de desarrollo.	8
1.3. Diagrama de Gantt del transcurso temporal real en la realización del TFG sobre las fases de desarrollo. Se incluyen puntos importantes en el desarrollo tales como reuniones y revisiones del proyecto.	9
2.1. Diagrama de Venn que relaciona los fundamentos teóricos necesarios para la correcta comprensión y seguimiento del trabajo. Se tratarán los tres conceptos de forma incremental en el área que abarca cada campo, es decir, es necesario explicar Aprendizaje Automático (ML) antes que Aprendizaje Federado (FL) ya que el último se desarrolla sobre los conceptos del anterior.	14
2.2. Esquema básico del problema de aprendizaje. Imagen extraída del libro Learning From Data [23].	17
2.3. Función de activación sigmoide usada para asignar un valor de probabilidad dada una señal definida por un modelo w. Figura extraída de [24].	19
2.4. El Descenso del Gradiente se puede comparar análogamente a una pelota que se mueve sobre una superficie con relieves. El proceso de optimización viene a ser la tendencia de la pelota a bajar de las <i>colinas</i> para llegar una altitud más baja (<i>valles</i>).	20
2.5. Efecto del Learning Rate η en el entrenamiento de un modelo. Imagen extraída de [25].	21

2.6. Estructura de una neurona artificial. Una neurona artificial es en esencia un modelo lineal cuya salida es activada por una función de activación como la ReLU [27] o la Sigmoide 2.3. Imagen extraída de [28].	22
2.7. Ejemplo esquemático de red neuronal de dos capas ocultas. Imagen extraída de Wikimedia Commons [31].	23
2.8. Arquitectura de LeNet-5 [34] una red neuronal convolucional, aquí para reconocimiento de dígitos manuscritos.	24
2.9. Operación de convolución aplicada sobre una matriz de valores enteros de entrada y un filtro convolucional 3×3 . Imagen extraída de TowardsDataScience [35].	24
2.10. Filtro de Sobel para detectar bordes en imágenes. Imágenes extraídas de Wikimedia Commons [37]	25
2.11. Esquema de Aprendizaje Federado Centralizado con teléfonos inteligentes entrenando cooperativamente un modelo de ML. Imagen extraída de Wikimedia Commons [38].	26
 3.1. Visualización de los dos tipos de aprendizaje federado: Horizontal (a) y Vertical (b). Imagen extraída de [44].	34
3.2. Taxonomía propuesta por [7]. La taxonomía divide los algoritmos de selección sobre seis dimensiones: Policy, Termination Condition, Value Interpretation, Client Characteristics, Shared Information, y Value Generation. Que a su vez van separadas dependiendo si conciernen al servidor (marcadas en color rojo), a los clientes (marcados en verde) o a ambos (azul). Imagen extraída del mismo paper [7].	36
 4.1. Ejemplos de dígitos manuscritos de MNIST. Imagen extraída de [69] via Wikimedia Commons.	46
4.2. Ejemplos de CIFAR-10 agrupadas por las 10 clases que caracterizan el dataset. Imagen extraída de la página web del conjunto de datos [73].	47
4.3. Traza del descenso del gradiente sobre un espacio de valores normalizados (derecha) y no normalizados (izquierda). Imagen extraída de [76].	49
4.4. El flujo de aprendizaje y clasificación del método tradicional o clásico de clasificación de imágenes y clasificación de imágenes con Deep Learning. En el método clásico, las características son creadas a manos, normalmente por agentes expertos o utilizando descriptores como SIFT y HOG. En Deep Learning, estas características son <i>aprendidas</i> por la misma red neuronal. Imagen extraída de [80].	50
4.5. Diseño modular de FLEXible. Imagen extraída de [67].	57

4.6.	Esquema de los permisos asignados a cada rol de FLEX. Imagen extraída de [67].	58
4.7.	Esquema visual del flujo de trabajo de AFL en un problema de clasificación binaria. Imagen extraída del paper original [57].	61
4.8.	Función exponencial. Los autores de AFL [57] utilizan esta función para crear la distribución de probabilidad para muestrear los clientes seleccionados para la siguiente ronda en base a sus valuaciones v_k , de manera que los que tengan mayor v_k tengan más probabilidades de salir seleccionados. Imagen extraída de [98] via Wikimedia Commons.	63
4.9.	Vista general del protocolo FedCS. Las líneas continuas denotan procesos computacionales (tanto del servidor como de los clientes) y las discontinuas, procesos de telecomunicación. El término <i>Scheduled</i> se refiere a las actualizaciones y subidas del modelo que se estiman de cumplir con los <i>deadlines</i> impuestos. Imagen extraída de [8].	71
4.10.	Vista general del protocolo usado por HybridFL. Nótese las similitudes con FedCS a excepción del paso se subida de datos de los clientes al servidor (líneas rojas). Imagen extraída de [58].	73
4.11.	Áreas bajo la curva de dos curvas de aprendizaje diferentes. Una de ellas converge más rápido que la otra (azul) y en consecuencia cuando calculamos su AUC el área que ocupa esta curva es menor que la curva que converge más lentamente.	84
5.1.	Procedimiento de implementación de un método de selección de clientes.	89
6.1.	Mapa de calor de la federación de MNIST sobre 10 nodos/clientes para los dos casos de distribución IID y No-IID utilizando la federación de [102].	95
6.2.	Valores AUC de las curvas de aprendizaje resultante de la búsqueda exhaustiva de parámetros para AFL (menor valor AUC es mejor). En rojo se marcan los mejores valores AUC que marcan la mayor velocidad en convergencia sobre esos parámetros.	97
6.3.	Curvas de aprendizaje resultantes de los algoritmos basados en el rendimiento del modelo: RandomSampling, AFL y GreedyFed. La línea roja discontinua indica la ronda final de inicialización Round-Robin de valores de Shapley en GreedyFed.	99

6.4. Comparativa en la selección de los clientes por los métodos Random, AFL y GreedyFed para MNIST y CIFAR-10. Las gráficas de línea muestran la estimación de la función de densidad de los resultados de selección, o también denominado KDE (<i>Kernel Density Estimation</i>) [103].	101
6.5. Tiempos de entrenamiento por ronda de GreedyFed en MNIST y CIFAR-10. El área gris corresponde a la zona de no convergencia derivada de las rondas donde no han convergido los valores de Shapley tomando como límites el mínimo y el máximo tiempo transcurrido.	103
6.6. Proporción de rondas en los que el algoritmo GtG-Shapley ha convergido en MNIST y CIFAR-10.	104
6.7. Comparativa en la selección de RandomSampling y AFL en CIFAR-10.	105
6.8. Curvas de aprendizaje de RandomSampling de 5 ejecuciones con diferentes semillas aleatorias.	106
6.9. Comparativa en la selección de los clientes por los métodos Random, HybridFL y Dyn-Hybrid para MNIST y CIFAR-10. Las gráficas de línea muestran la estimación de la función de densidad de los resultados de selección, o también denominado KDE (<i>Kernel Density Estimation</i>) [103].	108
6.10. Diagrama de ejemplo comparativo entre RandomSampling y HybridFL en el número de actualizaciones y rondas que realizan. Este ejemplo es basado en los experimentos hechos anteriormente en 6.4 como diagrama ilustrativo.	110
6.11. Comparativa en la cantidad total de clientes seleccionados aleatoriamente (simulando el paso de selección aleatoria inicial de M clientes de 6) para un número distinto de rondas totales. El objetivo es comparar la capacidad de generalización de RandomSampling (caso (a)) y HybridFL/Dyn-HybridFL (caso (b)). Para el caso (a) se han considerado todos los $M = 100$ clientes a muestrear por cada ronda mientras que en el caso (b) se ha considerado un número mucho menor de clientes ($M = 3$), de esta manera, visualizamos de una manera más fiel al lo que observamos en los resultados de los experimentos.	112
6.12. Comparativa en el número de clientes seleccionados para cada ronda $ S_t $ de HybridFL y Dyn-HybridFL.	113

Índice de tablas

1.1. Resumen del coste del proyecto desglosado en recursos tanto humanos como de equipo técnico. El coste del equipo es un estimado de las especificaciones técnicas públicas del servicio Google Colab con el plan gratuito [17].	10
6.1. Métricas de rendimiento obtenidas de los experimentos de los métodos de selección basados en rendimiento del modelo. Se marcan en negrita los que resultan vencedores. Los resultados que muestran “N/A” en las métricas RoA@ x indican que no se ha podido llegar a cierta <i>accuracy</i> en el número de rondas establecido.	100
6.2. Métricas de rendimiento obtenidas de los experimentos de los métodos de selección basados en recursos. Se marcan en negrita los que resultan vencedores en cuanto a la métrica se refiere. Los resultados que muestran “N/A” en algunas de las métricas indican que no se ha podido obtener cierta métrica en la ejecución del algoritmo.	109

Índice de Algoritmos

1.	Protocolo estándar del entrenamiento de un modelo en FL.	27
2.	Protocolo general seguido en una estrategia de selección de clientes en FL. Las fases estilizadas en <i>italica</i> son opcionales y dependen del método de selección.	31
3.	Algoritmo de muestreo de AFL para seleccionar clientes. Extraído parcialmente del trabajo original [57].	64
4.	Algoritmo Greedy Shapley Client Selection. Extraído del paper original [60].	68
5.	Aproximación de Valores de Shapley (Server-side) GTG-Shapley.	69
6.	Protocolo HybridFL. K denota el número de clientes totales, $C \in (0, 1]$ la fracción de clientes aleatorios que reciben la petición de recursos en cada ronda. Protocolo extraído de [58].	74
7.	Algoritmo Client Selection de HybridFL. Algoritmo extraído de [58].	77
8.	Algoritmo de selección de los datos para ser subidos por los clientes.	77
9.	Algoritmo Client Selection de Dyn-HybridFL para el uso de T_{round} dinámico.	81

Capítulo 1

Introducción

El aprendizaje automático o Machine Learning (ML) es un campo dentro de la Inteligencia Artificial que permite a sistemas informáticos aprender de un conjunto de datos sin intervención humana, logrando identificar patrones que se encuentran en los mismos para así poder predecir o clasificar nuevos datos que no hayan visto anteriormente. El rendimiento y la efectividad que muestran estos modelos de Machine Learning en su entrenamiento y evaluación, depende de varios factores que podemos sintetizar en: la calidad y tipo de los datos, y la elección del mismo modelo [1]; la elección y procesamiento de estos componentes puede influir en gran medida el rendimiento final del modelo.

Progresivamente, con el avance del Machine Learning se han observado limitaciones en cuanto al escenario tradicional centralizado, en la que todos los recursos computacionales, los datos y el entrenamiento se encontraban alojados en un solo entorno. Esto ha llevado al estudio de diversas técnicas para llevar el entrenamiento de estos modelos de ML al ámbito distribuido y descentralizado [2], impulsado por la creciente tendencia hacia los sistemas distribuidos y la necesidad en varios campos de preservar los datos usados para entrenar estos modelos [3]. Este cambio hacia un escenario descentralizado y distribuido ha cambiado la forma en la que se preparaba un entorno convencional de ML como la necesidad de descentralizar los datos, distribuir la carga entre los diversos nodos o máquinas, aplicar medidas de privacidad de los datos y entrenar de forma colaborativa un modelo global de ML. Una de las técnicas que ha surgido en esta última década que resuelve estos problemas es el del Aprendizaje Federado, también conocido como aprendizaje colaborativo.

El Aprendizaje Federado o FL (*Federated Learning*) [4] es un paradigma o subcampo del aprendizaje automático o Machine Learning (ML) que comprende escenarios donde participan **múltiples entidades**, referidos usualmente como nodos o clientes, para entrenar un modelo de aprendizaje que

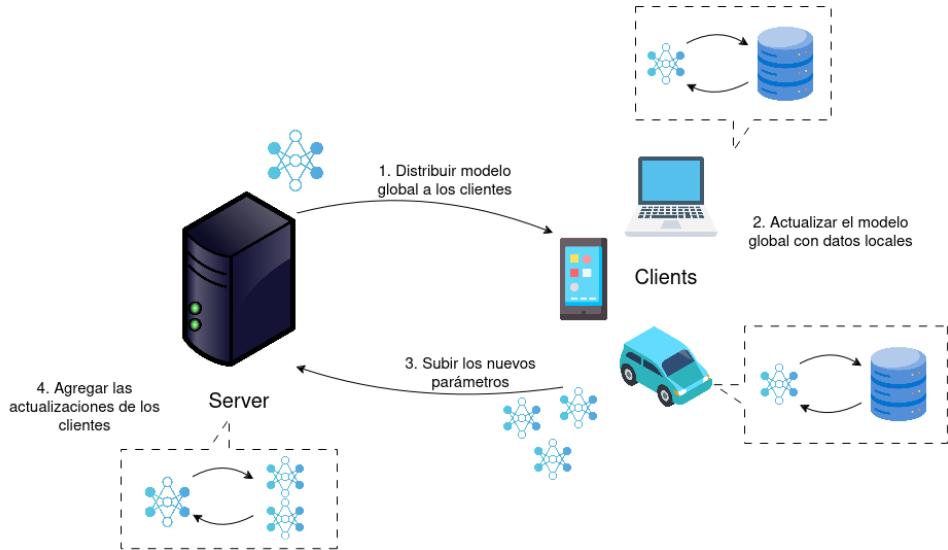


Figura 1.1: Esquema de una iteración en un escenario de aprendizaje federado centralizado. El servidor se encarga de distribuir el modelo global mientras que los clientes actualizan este modelo con sus datos locales; finalmente el servidor agrega estas actualizaciones del modelo para constituir el nuevo modelo global de la siguiente iteración.

intenta resolver un problema de forma colaborativa, manteniendo al mismo tiempo los datos *descentralizados* y en muchos casos, privados a cada cliente. En la forma más común, el aprendizaje federado se lleva a cabo teniendo una arquitectura *cliente-servidor* como podemos observar esquemáticamente en la Figura 1.1, denominado Aprendizaje Federado Centralizado (CFL), donde el entrenamiento de los modelos lo realizan los clientes con sus propios datos locales, y el servidor actúa como el agregador de los modelos entrenados por cada cliente para constituir el modelo global final a evaluar. El servidor también coordina el aprendizaje, enviando el modelo global a todos los clientes seleccionados, recuperando los parámetros de los clientes y finalmente agregarlos al modelo global. Este proceso se realiza de forma iterativa, mejorando el modelo global con cada iteración o *ronda*, agregando los modelos entrenados con diferentes conjuntos de datos locales a cada cliente, produciendo una mejora significativa en el modelo global [4].

1.1. Motivación del trabajo

En los últimos años y con el avance tanto en el campo del Machine Learning como en la creación y diseño de los sistemas distribuidos así como el surgimiento de la necesidad de protección de los datos [5], el aprendizaje fe-

derado o (FL por sus siglas en inglés) [4] ha emergido como una metodología revolucionaria en el campo del Aprendizaje Automático que permite entrenar modelos de ML de manera colaborativa sin la necesidad de centralizar los datos usados para ajustar los modelos.

Varias estrategias y métodos se han planteado para entrenar modelos distribuyendo la carga de trabajo sobre múltiples máquinas, aprovechando la computación paralela y concurrente lo cual ayuda a obtener un mejor rendimiento en el modelo final [6]; sin embargo, este concepto de Machine Learning Distribuido no contempla el escenario donde los datos distribuidos necesiten ser privados. El paradigma de FL permite el entrenamiento de modelos sobre máquinas distribuidas donde los datos de cada nodo o cliente son privados y no se comparten entre otros clientes o el mismo servidor central. Esta técnica es especialmente conveniente donde la privacidad es crucial en el ámbito y en el contexto del problema a resolver (e.g. en el ámbito de las finanzas o la salud).

En el entrenamiento de modelos en FL donde participan pocos clientes, resulta factible y conveniente actualizar los parámetros de cada cliente en cada ronda o iteración de entrenamiento, ya que de esta manera estaríamos aprovechando toda la capacidad computacional y de datos del que poseemos. Sin embargo, cuando el número de clientes crece, también lo hace la sobrecarga de comunicación producto de la subida y bajada de los modelos entre los clientes y el servidor central [7]. La sobrecarga en la comunicación ha sido demostrada ser el principal cuello de botella o *bottleneck* en cuestión de tiempo efectivo en el entrenamiento de modelos en aprendizaje federado ([4], [8]), de modo que considerar todos los parámetros de cada cliente en cada ronda se vuelve un reto. Del mismo modo, cuando el número de clientes es elevado también lo hace la heterogeneidad de los mismos; es vital considerar la naturaleza heterogénea de cada cliente en cuanto a sus datos, capacidad computacional, ancho de banda en red, etc. dado que estos recursos varían considerablemente, especialmente en escenarios donde se utilizan teléfonos móviles o dispositivos IoT ([9], [10]); esto hace que la selección de los clientes por ronda no sea una tarea trivial a la hora de obtener el mejor rendimiento del modelo final, ya que algunos clientes serán más valiosos que otros y la elección equivocada de estos clientes puede afectar de manera significativa la convergencia hacia el mejor modelo, así como la eficiencia en comunicación y el consumo de los recursos. Por tanto, es importante estudiar y analizar sobre diferentes enfoques la selección de estos clientes, y así identificar las mejores técnicas y estrategias de selección que permitan obtener un modelo de calidad al mismo tiempo que minimicen los costes asociados

En este trabajo de fin de grado se propone estudiar y analizar estrategias de selección de clientes considerando diferentes enfoques en aprendizaje federado. Los objetivos incluyen el **análisis e implementación** de méto-

dos de selección de clientes que hayan demostrado buenos resultados en la literatura, el estudio experimental de sus fortalezas y debilidades y la identificación de las mejores estrategias y/o prácticas que optimicen el proceso del aprendizaje. Se espera que este estudio contribuya al desarrollo de técnicas más eficientes y efectivas en el área del aprendizaje federado, las cuales beneficiarán a una serie de aplicaciones y sectores dependientes de esta tecnología.

1.2. Objetivos del estudio

El objetivo principal de este TFG se centra en el **análisis y comparación de distintas técnicas y estrategias de selección de clientes en aprendizaje federado**, con objetivo de poder mejorar el rendimiento y eficiencia del modelo global. Como consecuencia de este análisis y comparativa, se pretende identificar o reconocer las técnicas que hayan mostrado una mejora significativa de rendimiento que pueda permitir proponer mejoras para contribuir en la investigación de nuevas estrategias o vías de selección de clientes en aprendizaje federado.

Para poder conseguir este objetivo, desgranamos éste en objetivos parciales o específicos que nos ayudarán a conseguirlo:

- **Revisión de la literatura sobre la selección de clientes en aprendizaje federado.** Esto nos ayudará a clasificar las distintas estrategias existentes en cuanto a su enfoque y características, para así elegir un subconjunto de técnicas a estudiar que tengan características diferentes.
- **Implementación de las técnicas de selección elegidas en aprendizaje federado.** El objetivo consiste en el desarrollo de estas técnicas elegidas y su posterior adaptación o integración en un entorno o marco de aprendizaje federado.
- **Diseño y ejecución de experimentos que nos permitan medir el rendimiento de estas estrategias.** Debemos de llevar a cabo experimentos donde especifiquemos las métricas que nos permitan medir tanto el rendimiento del modelo global, el consumo de los recursos y la eficiencia en la comunicación.
- **Ánálisis y comparación de los resultados obtenidos en la experimentación de cada estrategia.** Con el resultado obtenido en los experimentos, comparamos y analizamos las fortalezas y debilidades de cada método y su impacto en el rendimiento del modelo, consumo y eficiencia.

- **Identificar las mejores estrategias y técnicas para la selección de clientes en FL.** Con esto pretendemos identificar qué decisiones o métodos de estas estrategias han mostrado un impacto positivo en el aprendizaje y a partir de ellas proponer mejoras, soluciones o vías de investigación con el propósito de seguir desarrollando en la selección de clientes.

1.3. Requisitos y Casos de Uso

Aunque éste TFG sea un estudio de carácter de investigación y no necesariamente tendría que incluir requisitos funcionales y no funcionales; vemos necesario definir algunos de estos requisitos y casos de uso sobre la aplicabilidad de los mismo métodos de selección que vamos a estudiar. Esto no solo proporcionará una visión práctica a los métodos que usaremos, sino que dotará de relevancia el estudio y análisis que propone este TFG.

En cuanto a los requisitos, éstos se centran en la necesidades técnicas y funcionales de la implementación de estos algoritmos así como el mismo entorno de aprendizaje federado que vamos a desarrollar. Se dividen de la siguiente manera:

- **Requisitos Funcionales:**

1. El sistema de selección debe seleccionar clientes que maximice la convergencia del modelo.
2. El sistema de selección de clientes debe de poder realizar una selección que atiende a los recursos heterogéneos de los clientes tanto en datos como en recursos.
3. El sistema de selección debe minimizar el número total de rondas de entrenamiento para mejorar la eficiencia en las comunicaciones.
4. El entorno de Aprendizaje Federado debe poder integrar diferentes esquemas de selección de clientes, para de esta manera comparar diferentes estrategias entre sí sobre un mismo entorno experimental.

- **Requisitos No Funcionales:**

1. El sistema de selección debe de escalar en número de clientes sin dejar que su entrenamiento sea viable.
2. El sistema de selección debe optimizar el uso de los recursos hardware que se poseen (CPU, GPU, energía, ...).

3. El sistema debe ser fácil de mantener y modificar en el tiempo, esto es, que el entorno de aprendizaje federado sea lo suficientemente flexible para la integración y modificación de nuevas características.
4. El sistema debe de completar el entrenamiento de un modelo de ML en un tiempo tratable, lo que implica aprovechar los recursos hardware y software para lograr entrenar modelos en el menor tiempo posible.

En cuanto a los casos de uso del entrenamiento de modelos con aprendizaje federado podemos incluir los siguientes casos usados en situaciones reales:

- Entrenamiento de modelos sobre un entorno de dispositivos IoT. Este caso es el más comúnmente aplicado en aprendizaje federado ya que justamente atiende en primer lugar a un entorno distribuido y en segundo lugar a un entorno de clientes heterogéneos en recursos. Los algoritmos de selección de clientes (en especial los basados en recursos) tratan de maximizar la convergencia del modelo sobre restricciones de tiempo, algo esencialmente útil en IoT donde los dispositivos interactúan con el entorno en tiempo real.
- Otro caso de uso del aprendizaje federado es en el área de salud. En estos casos, el entrenamiento de los modelos son muy sensibles al rendimiento del modelo, y por tanto, los algoritmos de selección de clientes pueden seleccionar los datos de hospitales que tengan no solo una mayor cantidad de datos sino mejor calidad de estos (e.g. mayor balanceo de clases).

1.4. Planificación y costo del proyecto

Este trabajo forma parte de la evaluación de la asignatura TFG (Trabajo de Fin de Grado) una asignatura que cuenta con 12 créditos ECTS [11], donde cada crédito equivale a 25 horas de trabajo (dado que no hay carga lectiva). Se estima por tanto un total de 300 horas para completar este trabajo y que se deben de repartir en el tiempo que tanto el alumno como los tutores disponen para la realización de este. La asignación de los tutores ya estaba realizada y pactada desde septiembre del año 2023, sin embargo, no se realizó trabajo efectivo en la realización del proyecto hasta principios del mes de marzo del año 2024. La presentación, y por tanto al finalización, también estaba pactada a llevarse a cabo en el mes de septiembre del 2024 (convocatoria extraordinaria) dado que por temas de gestión de tiempo no iba a ser posible presentarlo en la convocatoria ordinaria (junio 2024). Por

tanto, la distribución del tiempo desde el mes de marzo hasta el mes de septiembre del 2024 se establece de una forma homogénea en los 6 meses que se disponen. Teniendo en cuenta que cada mes del año tiene aproximadamente $\frac{365\text{días}}{12\text{meses} \times 7\text{semanas}} = 4.345$ semanas, entonces contamos con:

$$6 \text{ meses} \times 4.345 \text{ semanas/mes} = 26.07 \text{ semanas}$$

Ahora, repartimos estas 300 horas por las 26.07 semanas que disponemos para obtener las horas a dedicar por semana y por día:

$$\begin{aligned}\frac{300 \text{ horas}}{26.07 \text{ semanas}} &= 11.51 \text{ horas / semana} \\ &= 1.64 \text{ horas / dia}\end{aligned}$$

Con estos cálculos, tenemos un límite inferior de horas a dedicar por día y por semana. Teniendo en cuenta imprevistos en el tiempo, optamos por un total de 13 horas por semana y ≈ 2 horas y media al día (sin contar fin de semana como días de descanso).

El trabajo a continuación consiste en un TFG dirigido a la investigación con carácter científico en el campo del aprendizaje automático (ML) y a la inteligencia artificial (IA). Es por esto que este trabajo requiere de unos **requisitos orientados más hacia el análisis, experimentación y toma de resultados**; a diferencia de otros trabajos orientados a la experiencia de usuario, calidad de la implementación, etc. Es por estos factores que se optará por una metodología de desarrollo en cascada [12] para la planificación temporal del trabajo. El desarrollo en cascada es una metodología en la ingeniería del software por la que las fases del desarrollo se realizan de una forma lineal *sin retroceder* a fases previas, lo cual consideramos una metodología suficiente en este trabajo de investigación dado los requisitos que posee. Definimos entonces, las distintas etapas del desarrollo del proyecto basadas en las fases típicas en otros trabajos científicos del campo así como en el modelo del desarrollo en cascada:

- **Análisis de los requisitos.** En esta etapa se plantea el contexto del trabajo así como los requisitos para realizar el trabajo y los objetivos a cumplir. En esta etapa se comprende un periodo de *revisión de la literatura*; en nuestro caso acerca de los diferentes métodos de selección y sus distintas clasificaciones.
- **Diseño.** En esta fase se describen los distintos métodos y herramientas a utilizar para crear el entorno experimental adecuado para llevar a cabo la ejecución de los experimentos y la toma de los resultados de una forma eficiente y sencilla. También se realizan primeras implementaciones para comprobar la viabilidad del trabajo diseñado.

Etapa	Marzo				Abril				Mayo				Junio				Julio				Agosto				
	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4	
Análisis de requisitos																									
Diseño																									
Implementación																									
Experimentación																									

Figura 1.2: Diagrama de Gantt de la planificación temporal teórica para la realización del TFG sobre las fases de desarrollo.

- **Implementación.** En esta etapa se materializa lo descrito en la fase de **Diseño**. Se comprenden tareas de programación y depuración del entorno del que se apoyará el cuerpo experimental del trabajo. Esta fase es la más propensa a cambios en fases posteriores debido a la naturaleza del desarrollo de software (e.g. *bugs*, refactorización del código, ...).
- **Experimentación y toma de resultados.** En esta etapa se comprende la realización de los experimentos y el registro de los resultados obtenidos en la ejecución de estos. En esta fase se especifican los parámetros usados en las pruebas realizadas y se discuten los resultados y hallazgos que se hayan encontrado.

Teniendo definidas las fases o etapas del ciclo de desarrollo del trabajo, se describe de forma visual la planificación propuesta del trabajo en la línea temporal desde el inicio hasta el final de la realización del mismo mediante un **diagrama de Gantt** [13] mostrada en la Figura 1.2. Nótese que la planificación muestra de forma ideal y homogénea el transcurso del desarrollo del TFG, y no comprende revisiones, tomas de decisiones, reuniones, etc. Tampoco toma en cuenta el tiempo real en la finalización de cada etapa que pueden no ser iguales debido a que unas etapas toman más o menos tiempo que otras; sin embargo, se estimaba desde el comienzo que habrán ciertas fases que tomen más tiempo como la fase de **Implementación** o **Experimentación**, el cual es el motivo que en esta fase se ha optado por un tiempo un poco mayor de trabajo, aunque de forma estimada. En la Figura 1.3 se muestra el diagrama de Gantt que representa el transcurso del desarrollo real del proyecto, que incluye además puntos adicionales que fueron importantes o que marcaron un avance en la realización del TFG.

En esta sección también vamos a incluir el costo estimado para llevar a cabo este trabajo. Este TFG posee dos recursos necesarios para su realización: recursos humanos y recursos de equipo técnico o de implementación. En cuanto a los recursos humanos se toman en cuenta los integrantes que formaron parte en el proyecto, es decir, el alumno y los tutores (en este caso se cuenta con 2 tutores). Se calcula el coste de cada participante de la

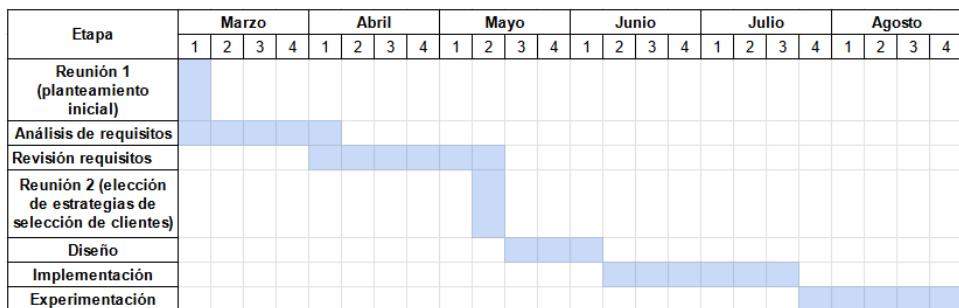


Figura 1.3: Diagrama de Gantt del transcurso temporal real en la realización del TFG sobre las fases de desarrollo. Se incluyen puntos importantes en el desarrollo tales como reuniones y revisiones del proyecto.

siguiente forma:

$$\text{Coste}_{\text{RR}} = \text{Total horas trabajadas} \times \text{Salario por hora} \times \# \text{ Trabajadores}$$

Teniendo en cuenta que el salario anual habitual para un trabajador en el sector de investigación en el área tecnológica y más concretamente en el ámbito de la Ciencia de Datos e Inteligencia Artificial con una experiencia de entre 2 y 3 años, es de una media de €20 – 30K al año [14], lo que se traduce en aproximadamente €14 – 19 por hora. Teniendo en cuenta además que el total de horas trabajadas es de al menos 300 horas, podemos computar el coste total en recursos humanos como:

$$\text{Coste}_{\text{RR}} = 300 \text{ horas} \times 19 \frac{\text{€}}{\text{hora}} \times 3 \text{ trabajadores} = 17100 \text{ €}$$

Por otro lado, el coste del equipo necesario para llevar a cabo la implementación y la experimentación se calcula teniendo en cuenta los componentes del ordenador de desarrollo. Sin embargo, tanto la experimentación e implementación se han realizado usando el servicio de Google Colab [15], el cual es un servicio web gratuito perteneciente a Google que permite la ejecución de cuadernos de Python (también conocido como Jupyter Notebooks) [16] utilizando recursos computacionales en la nube. El uso de este servicio basado en la nube fue de especial utilidad en el desarrollo del código que soporta el trabajo debido a la facilidad de su uso en cuanto a la instalación de librerías y de almacenamiento, así como el aprovechamiento de los recursos computacionales como CPU, RAM y GPUs a demanda. Aunque se ha utilizado en todo momento el plan gratuito de Google Colab, podemos estimar el coste de los recursos utilizados [17]; en el Cuadro 1.1 podemos ver un resumen del costo total del proyecto desglosado en los distintos componentes usados.

Recurso	Coste (€)
Coste _{RR}	17100
NVIDIA Tesla T4 16GB GDDR6	1500
RAM 16GB DDR4	50.00-80.00
CPU Intel(R) Xeon(R) 2.20GHz	119.99
Almacenamiento 1TB SSD	60.00-80.00
Periféricos	122.00
Total	18951.99-19001.99

Tabla 1.1: Resumen del coste del proyecto desglosado en recursos tanto humanos como de equipo técnico. El coste del equipo es un estimado de las especificaciones técnicas públicas del servicio Google Colab con el plan gratuito [17].

1.5. Estructura del documento

Este TFG esta estructurado en los siguientes capítulos:

1. **Introducción.** Este capítulo comprenderá una introducción comprensiva al lector para situarse en el ámbito que trata este trabajo así como información relevante acerca del trabajo que se ha realizado, los objetivos, motivación del estudio, planificación, así como la misma estructura del documento.
2. **Fundamentos Teóricos.** Este capítulo tiene el objetivo de profundizar en el campo que se centra este TFG, y de ayudar al lector en comprender y entender los conceptos y conocimientos fundamentales para entender el trabajo que se ha realizado. Conceptos de Machine Learning, Aprendizaje Federado y Selección de Clientes en FL son tratados con más profundidad en este capítulo y que son necesarios para la correcta comprensión del lector con los temas que se van a tratar en capítulos posteriores.
3. **Estado del Arte.** Luego de haber visto los fundamentos necesarios para leer este TFG en *Fundamentos Teóricos*, en este capítulo se situará al lector en las estrategias de selección de clientes que hay en la literatura. Para ello, se expondrán distintas taxonomías y clasificaciones de las distintas estrategias de selección de clientes y que han mostrado buenos resultados. Al final, de todas estas estrategias se elegirán un subconjunto de éstas que vamos analizar, implementar y ejecutar en los capítulos posteriores para luego hacer un análisis comparativo entre ellos.
4. **Metodología.** En este capítulo se detallarán los métodos y materiales

utilizados para llevar a cabo el estudio de este TFG. En éste se detallan los conjuntos de datos a utilizar, los modelos de ML para entrenar estos datos, se detallan también el diseño de los métodos de selección que hemos elegido en *Estado del Arte* y describiremos los experimentos que vamos a realizar para la obtención y análisis de los resultados.

5. **Implementación.** En este capítulo se dará una descripción general de la implementación que se ha realizado para poder ejecutar las estrategias de selección de clientes elegidos. Detalles de la implementación se pueden encontrar en el código de este TFG (Véase capítulo 5 para más información).
6. **Experimentos y Resultados.** Este capítulo consistirá en la configuración experimental para la obtención de los resultados de cada método de selección así como la misma exposición de éstos resultados. Posteriormente, se realizará un análisis comparativo de estos resultados y comentarios de los hallazgos encontrados.
7. **Conclusiones y Trabajos Futuros.** Finalmente, este último capítulo dará un resumen final de lo que ha consistido este trabajo así como conclusiones finales del estudiante sobre el estudio llevado a cabo.

Capítulo 2

Fundamentos Teóricos

Para el buen seguimiento por parte del lector de este trabajo, es necesario establecer ciertos fundamentos y conceptos que son imprescindibles en el entendimiento del estudio que se desarrollará en adelante. Este capítulo proporciona una vista general de conceptos teóricos que servirá en algunos casos como introducción al tema concreto que se discutirá y analizará en capítulos posteriores. Cada sección será un campo que contenga al siguiente, e.g. es conveniente instruir al lector en Aprendizaje Automático para luego instruirle en Aprendizaje Profundo. En nuestro caso, trataremos tres conceptos fundamentales para poder comprender este TFG: *Aprendizaje Automático*, *Aprendizaje Federado* y *Selección de clientes en FL*; en la Figura 2.1 se muestra el **diagrama de Venn** [18] que relaciona estos tres conceptos e iremos discutiéndolos de mayor a menor orden de escala.

Entre los fundamentos teóricos a tratar tenemos por un lado el concepto de Machine Learning o Aprendizaje Automático, que será de utilidad para situar al lector en el campo o ámbito en el que se encuentra el estudio de este trabajo. Conceptos como *modelos de ML*, *precisión del modelo*, *entrenamiento y evaluación de un modelo*; serán vistos con un poco más de detalle, el suficiente como para que el lector sea capaz de utilizarlos en su perspectiva acerca del análisis, resultados y conclusiones de este trabajo. Por otro lado, tenemos aprendizaje federado (*Federated Learning*), que se desarrolla sobre los conceptos aprendidos en Aprendizaje Automático para entrenar modelos de forma distribuida y con datos descentralizados, donde también veremos los desafíos y grandes problemas que se enfrenta tal paradigma en contraste al aprendizaje centralizado y local convencional, así como su notación estándar que utilizaremos más adelante en el trabajo. Finalmente, veremos una vista general de la selección de clientes en aprendizaje federado, donde veremos el esquema general de lo que implica la selección de los nodos sobre cada ronda de entrenamiento así como los problemas que tratan de resolver las distintas estrategias de selección de clientes. Este último tema servirá de

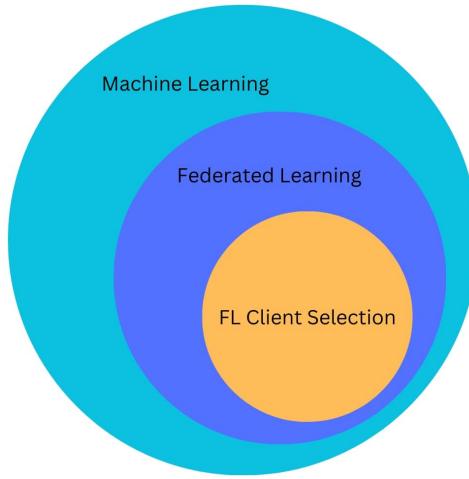


Figura 2.1: Diagrama de Venn que relaciona los fundamentos teóricos necesarios para la correcta comprensión y seguimiento del trabajo. Se tratarán los tres conceptos de forma incremental en el área que abarca cada campo, es decir, es necesario explicar Aprendizaje Automático (ML) antes que Aprendizaje Federado (FL) ya que el último se desarrolla sobre los conceptos del anterior.

conocimiento previo para el capítulo 3 donde se detallarán en las distintas estrategias y enfoques que existen, así como sus distintas clasificaciones y problemas que intentan resolver.

2.1. Aprendizaje Automático (ML)

Sería imposible entender lo que nos ofrece el Aprendizaje Federado sin saber primero de la disciplina donde reside, el Aprendizaje Automático o también conocido como *Machine Learning* o simplemente *ML*. El Machine Learning es un subcampo o rama de la Inteligencia Artificial y en general una rama de la informática que se enfoca en el uso de algoritmos *entrenados* sobre un conjunto de datos para poder crear modelos entrenados de forma autónoma, que son capaces de predecir un resultado o clasificar información sin la **supervisión humana** [19]. El aprendizaje automático entonces solventa problemas en los que se necesita capturar patrones subyacentes que son difícil para los humanos captar y programar; es por esta razón que el ML se usa en una variedad muy amplia de tareas como ([20]):

- Optimización en ventas: de entre los que podemos encontrar sistemas de recomendación para productos comerciales, multimedia, etc.
- Servicios al consumidor: el ML permite la automatización de rutinas

de servicio al cliente para los que se pueden crear *chatbots* que imiten la interacción humana.

- Prevención de fraude: el ML es ampliamente usado en grandes bancos para captar características de transacciones sospechosas que pueden inducir un fraude.

2.1.1. ¿Como funciona el Aprendizaje Automático?

Cuando nos referimos a algoritmos de aprendizaje o *algoritmos que aprenden sobre un conjunto de datos*, nos referimos concretamente a métodos estadísticos y de optimización que permiten a una máquina capturar patrones subyacentes en los datos para usarlos en favor de predecir o clasificar nuevos ejemplos de datos.

UC Berkeley ([21]) divide el sistema de aprendizaje en tres componentes:

1. Un proceso de decisión: que son una serie de cálculos y pasos en el que dado un dato de entrada, que puede estar etiquetado o no, el algoritmo de aprendizaje producirá un valor estimado sobre un patrón encontrado en el dato.
2. Una función de error o pérdida: que es una manera de medir la predicción de un modelo de ML. Se le conoce también como *criterion* y es usado en la evaluación de una predicción del modelo sobre un conjunto de datos. El objetivo en todo problema de aprendizaje es poder **minimizar** la pérdida o el error del modelo, siendo este error definido por la función de pérdida.
3. Un proceso de optimización: un proceso por el cual un modelo activamente intenta *corregir* el error cometido en la predicción de los datos de entrenamiento de forma iterativa en busca de mejorar su precisión. Es el proceso de minimización de la función de pérdida.

Por tanto, podemos resumir el funcionamiento del Machine Learning como un proceso de decisión, para el cual dado un dato de entrada, el modelo de Machine Learning entrenado bajo un conjunto de datos, produce como salida un valor que estima el resultado de tal valor sobre un patrón captado sobre el dato. Estos modelos de Machine Learning entonces, son esencialmente algoritmos que mediante métodos estadísticos y de optimización, tratan de minimizar el error o pérdida sobre el conjunto de datos de entrenamiento definido por una función de error que evalúa la precisión del modelo en la predicción del conjunto de datos. Este proceso de optimización se realiza de forma *iterativa* hasta asegurar una pérdida lo suficientemente baja para ocupar una precisión y rendimiento aceptable bajo nuevos datos.

2.1.2. Tipos de aprendizaje

Para muchos problemas en los que podemos aplicar Machine Learning, encontramos diferentes tipos de aprendizaje que podemos clasificar en tres:

- **Aprendizaje Supervisado.** Este tipo de aprendizaje, los algoritmos de ML se entrena sobre datos que están *etiquetados* que describen el valor de salida real que indican como debe ser interpretado el dato de entrada. Por ejemplo, en un problema de clasificación binaria donde queremos clasificar una imagen de un perro o un gato, el conjunto de datos sería una secuencia de imágenes etiquetadas con *perro* en caso de ser una imagen de un perro, o *gato* en caso de ser una imagen de un gato. Este tipo de aprendizaje es el más comúnmente usado y tiene una gran variedad de aplicaciones, en especial para propósitos de clasificación o predicción.
- **Aprendizaje no supervisado.** A diferencia del aprendizaje supervisado, este tipo de aprendizaje entrena modelos de ML sobre datos *no etiquetados*, lo que implica que el algoritmo de aprendizaje debe capturar patrones en los datos sin ninguna *supervisión*. En este tipo de aprendizaje, se busca, en efecto, de etiquetar los mismo datos de forma automática, por lo que es muy usado en problemas de minería de datos donde los datos no necesariamente están etiquetados pero se necesita agrupar tales datos para distintos fines. Un uso común de este aprendizaje no supervisado es utilizarlo como paso previo al aprendizaje supervisado, para el que primero agrupamos los datos en distintas clases utilizando aprendizaje no supervisado, para luego entrenar un modelo con aprendizaje supervisado con datos etiquetados previamente.
- **Aprendizaje por refuerzo.** Este tipo de aprendizaje comprende el aprendizaje de un agente inteligente a tomar decisiones interactuando con el ambiente [22]. El agente es recompensado o penalizado dependiendo de las consecuencias de sus acciones (y por tanto de las decisiones que ha tomado). El objetivo por tanto, es maximizar la recompensa total. A diferencia de los otros tipos de aprendizaje, los datos en aprendizaje por refuerzo vienen de forma secuencial en donde cada paso depende de los anteriores tomados así como los datos actuales afectarán a los futuros. Usos comunes de este tipo de aprendizaje son robótica, juegos interactivos, gestión de recursos, etc.

En lo que resta de este TFG nos centraremos más en el tipo de aprendizaje supervisado, concretamente para problemas de clasificación con tipo de datos de imágenes o como se suele denominar, Visión por Computador. Sin

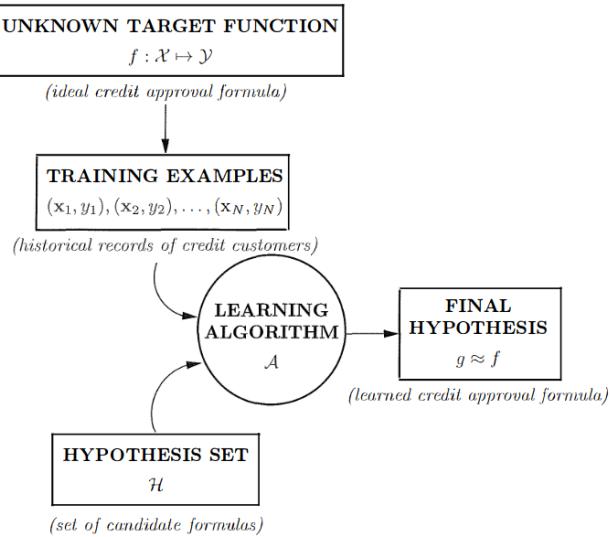


Figura 2.2: Esquema básico del problema de aprendizaje. Imagen extraída del libro Learning From Data [23].

embargo, nótese que los temas que se tratan en este trabajo relacionados con aprendizaje federado, son extrapolables a los distintos tipos de aprendizaje, así como a distintos algoritmos de aprendizaje, modelos y tipos de datos. Después de todo, el Aprendizaje Federado (y por tanto, la selección de clientes en FL) es una serie de prácticas para llevar a cabo el aprendizaje o ajuste de algoritmos de ML sobre un conjunto de datos de forma distribuída y con datos descentralizados, independientemente del tipo de problema que se intenta resolver o del formato de los datos.

2.1.3. El problema del aprendizaje

Para poder hacer uso del Machine Learning en cualquier problema de decisión, clasificación o predicción, se debe de formalizar el problema en cuestión en un problema de aprendizaje. En [23] definen formalmente el problema de aprendizaje teniendo en cuenta cinco componentes que podemos ver en la Figura 2.2:

1. **Función objetivo.** Como hemos dicho anteriormente, el aprendizaje automático intenta capturar patrones subyacentes en los datos que nos permiten predecir resultados sobre estos con la mejor precisión posible. Formalmente este concepto introduce la existencia de una función $f : \mathcal{X} \rightarrow \mathcal{Y}$ que es la función que predice de forma **correcta** todos los ejemplos de \mathcal{X} y que siempre será desconocida para nosotros. El

objetivo del ML es poder *acercarnos* a esta función.

2. **Conjunto de entrenamiento.** Se define en cualquier problema de aprendizaje el conjunto de datos de entrenamiento como una secuencia $\mathcal{D} = (\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_N, y_N)$, donde $\mathbf{x}_n \in \mathcal{X}$ son un conjunto de valores de cierta dimensión que representan un ejemplo concreto n del conjunto de datos, a lo que denominamos *características* o *features*. Mientras que $y_n \in \mathcal{Y}$, corresponde a la clase o etiqueta del ejemplo n que representa el valor o la clasificación real del ejemplo; esto en el caso de aprendizaje supervisado, para el caso de aprendizaje no supervisado los ejemplos de entrenamiento no cuentan con etiquetas, y son estas las predicciones de salida del modelo. Este conjunto de entrenamiento será usado por el algoritmo de aprendizaje \mathcal{A} para aprender la función f .
 3. **Conjunto de hipótesis.** El conjunto de hipótesis \mathcal{H} es un concepto abstracto en ML que representa el conjunto de todos los posibles modelos $h \in \mathcal{H}$ que acerquen a f . El algoritmo de aprendizaje \mathcal{A} elige la mejor función $g \in \mathcal{H}$ que aproxime a f (i.e. nuestro modelo final entrenado). Por ejemplo, para el caso de un modelo lineal, \mathcal{H} es el conjunto de todas las funciones candidatas $h \in \mathcal{H}$ tal y como se muestra en la Ecuación 2.1.
- $$h(\mathbf{x}) = \sum_{i=0}^d w_i x_i = \mathbf{w}^T \mathbf{x} \quad (2.1)$$
4. **Algoritmo de aprendizaje.** En esencia un algoritmo de aprendizaje optimiza una función hipótesis dentro del conjunto de hipótesis \mathcal{H} que aproxime de la mejor manera a la función objetivo. Existen una gran variedad de algoritmos de aprendizaje como el Regresor Lineal, Regresión Logística o Redes Neuronales, cada uno con una función de pérdida l que tratan de minimizar consiguiendo el mejor modelo que prediga el conjunto de entrenamiento.
 5. **Hipótesis final.** Finalmente tenemos la hipótesis final que vendría a ser aquella función $g \in \mathcal{H}$, resultado de un algoritmo \mathcal{A} sobre un conjunto de datos de entrenamiento \mathcal{D} , que mejor aproxima a la función objetivo f . Como ejemplo, en el caso de un modelo lineal, g es un vector de parámetros o pesos w que definen una función como suma ponderada que da como resultado la predicción sobre un dato de entrada (g sería una función tal y como se muestra en la Ecuación 2.1).

De esta manera, cualquier problema de decisión puede ser reducido a un problema de aprendizaje siguiendo el esquema propuesto por [23]. Para poner un ejemplo, consideremos el problema de clasificación de dígitos manuscritos MNIST (imágenes 24×24 en escala de grises) para aprendizaje

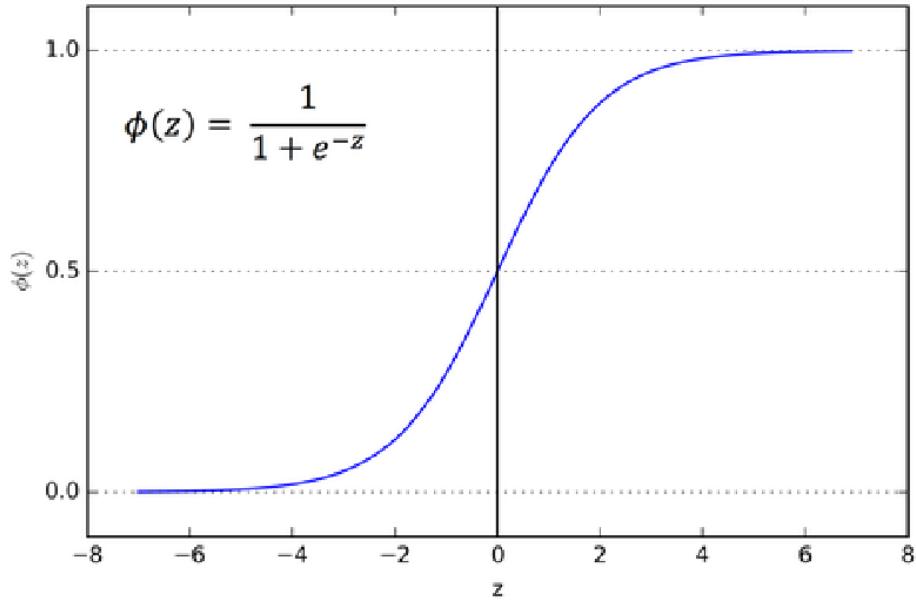


Figura 2.3: Función de activación sigmoide usada para asignar un valor de probabilidad dada una señal definida por un modelo w . Figura extraída de [24].

supervisado. En este caso el espacio de entrada \mathcal{X} sería una secuencia de valores $x \in [0, 255]$ que representan cada píxel de la imagen, mientras que el espacio de salida o etiquetas \mathcal{Y} serían valores $y \in \{0, 1, 2, \dots, 9\}$ que representan cada dígito del 0 al 9. De esta forma se conforma el conjunto de entrenamiento \mathcal{D} como una secuencia de datos $(x_n, y_n) : x_n \in \mathcal{X}, y_n \in \mathcal{Y}$. Luego, la función objetivo se define como una función $f : \mathcal{X} \rightarrow \mathcal{Y}$ que asigna a cada imagen $x \in \mathcal{X}$ una etiqueta $y \in \mathcal{Y}$. Dependiendo del algoritmo de aprendizaje, se utilizará un conjunto de hipótesis distinto, por ejemplo, podríamos utilizar regresión logística en el que las funciones hipótesis toman la forma de 2.2, donde θ es una función de activación como la *sigmoide* mostrada en la Figura 2.3, que asigna un valor de probabilidad entre $[0, 1]$ a una señal $w^T x$.

$$h(w) = \theta(w^T x) \quad (2.2)$$

2.1.4. Descenso del Gradiente

Como hemos comentado antes, el Aprendizaje Automático es un proceso de optimización, en el que tratamos de minimizar el error cometido por el

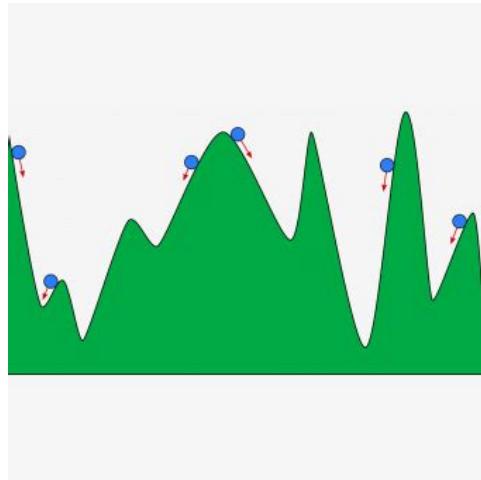


Figura 2.4: El Descenso del Gradiente se puede comparar análogamente a una pelota que se mueve sobre una superficie con relieves. El proceso de optimización viene a ser la tendencia de la pelota a bajar de las *colinas* para llegar una altitud más baja (*vallés*).

algoritmo de aprendizaje o modelo de ML sobre los datos de entrenamiento. El Descenso del Gradiente (GD) es una técnica general de optimización para minimizar una función doblemente derivable, denotada como $E_{in}(w)$. En [23], para entender el descenso del gradiente de manera intuitiva, se utiliza la analogía de una pelota que baja sobre una superficie con relieves o *colinas* (véase Figura 2.4); en la que la pelota, si esta en una colina, bajará hasta posarse en el fondo de un *valle*. Se aplica el mismo principio en el Descenso del Gradiente, solo que reemplazamos la superficie con la función de error de nuestro modelo $E_{in}(w)$ en un espacio de alta dimensionalidad. El objetivo es minimizar el error, es decir, buscar un modelo w tal que $E_{in}(w)$ sea el mínimo (en nuestra analogía, que lleguemos a tocar en un valle). El GD consigue esta optimización utilizando la Ecuación General de Aprendizaje 2.3, en el que actualizamos a un modelo $w^{(t+1)}$ restándole el gradiente de la función de error con el modelo actual $\nabla E_{in}(w^{(t)})$. El gradiente de una función es en esencia un vector de dos o más dimensiones que indica la dirección y magnitud de la tasa de inclinación de la función, es decir, es una generalización multivariable de la derivada de una función. Es por esto que, para *descender* la función de error, debemos de *restarle* su gradiente (que nos dará lo contrario de *ascender*). Este descenso se multiplica por un hiperparámetro η , denominado la tasa de aprendizaje o *Learning Rate* (LR), y que representa la "fuerza" en la que la función de pérdida se minimiza. Con esta ecuación, es posible minimizar el error de un modelo de ML de forma iterativa, y por tanto, asegurar un aprendizaje en el ajuste de la máquina con un conjunto de datos.

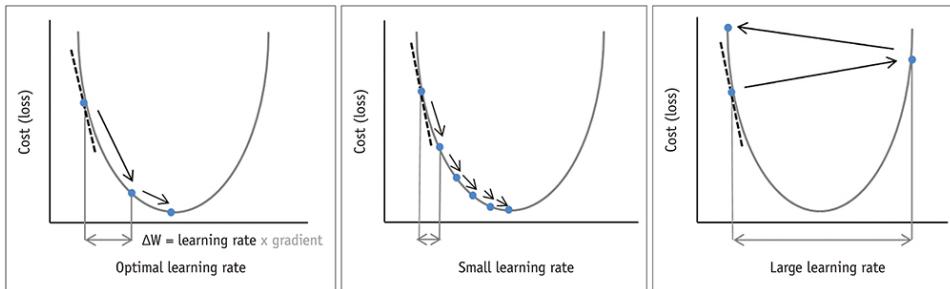


Figura 2.5: Efecto del Learning Rate η en el entrenamiento de un modelo. Imagen extraída de [25].

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta \cdot \nabla E_{\text{in}}(\mathbf{w}^{(t)}) \quad (2.3)$$

No obstante, la función de pérdida $E_{\text{in}}(\mathbf{w})$, así como lo es la superficie vista en la Figura 2.4, no tiene un único punto de valle o *saddle point*, sino que existen múltiples *mínimos locales* en los que la función de error pueda atascarse. Lo cierto es que, el llegar al punto de menor error, denominado *mínimo global*, depende de muchos factores, entre ellos el punto de partida o también dicho el modelo inicial $\mathbf{w}(0)$ del que partimos el proceso de optimización, ya que si nos encontramos en un punto del espacio muy cercano al mínimo global de $E_{\text{in}}(\mathbf{w})$, el gradiente descenderá hacia ese punto con mayor posibilidad. Otro factor que influye en gran medida es el Learning Rate η (2.3), ya que dependiendo de la magnitud en la tasa de aprendizaje podemos atascarnos en un mínimo local o nunca llegar a finalizar en uno de estos puntos. En la Figura 2.5 podemos visualizar el comportamiento del proceso de optimización del GD para diferentes magnitudes de η ; lo ideal es tener un LR que sea variable, existen muchos optimizadores basados en GD que utilizan conceptos como el *momento* de primer y segundo orden como el optimizador Adam [26], que escalan la tasa de aprendizaje de forma dinámica teniendo en cuenta la bajada del gradiente. El estudio de la optimización de estos hiperparámetros queda fuera del enfoque de este trabajo, sin embargo es importante el conocimiento de estos para detallar los resultados obtenidos posteriormente.

2.1.5. Redes Neuronales

Un concepto importante que usaremos en este trabajo son las Redes Neuronales o NN por sus siglas en inglés. Las Redes Neuronales son modelos de aprendizaje que inspirados en la estructura biológica de neuronas interconectadas del cerebro humano. En esencia, las Redes Neuronales consisten en unidades de cómputo llamadas neuronas, estas neuronas están compuestas

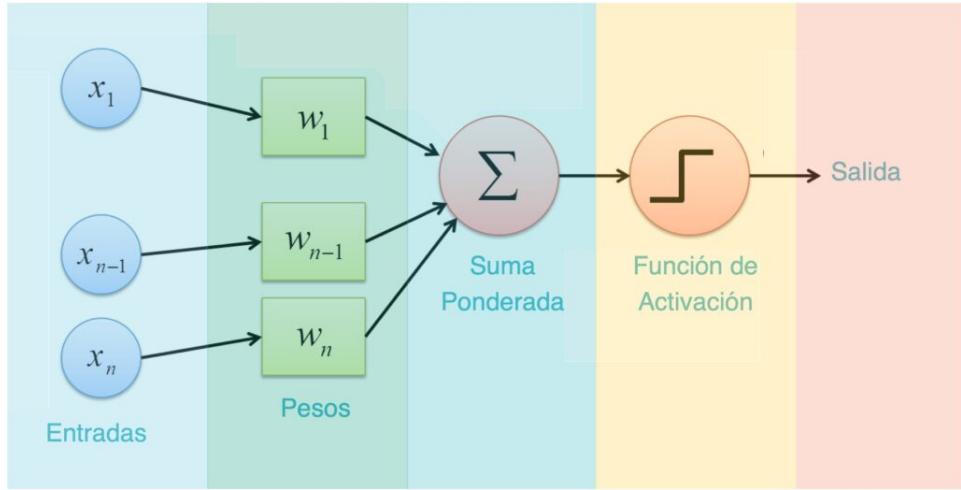


Figura 2.6: Estructura de una neurona artificial. Una neurona artificial es en esencia un modelo lineal cuya salida es activada por una función de activación como la ReLU [27] o la Sigmoide 2.3. Imagen extraída de [28].

de diferentes componentes que definen un único modelo lineal; si vemos la Figura 2.6 podemos observar como esta compuesta una neurona artificial y que en esencia es una suma ponderada activada por una función de activación no lineal. Como hemos podido apreciar, esto es muy similar al Regresor Logístico visto como ejemplo en la sección 2.1.3, y es que en efecto una neurona es un algoritmo de aprendizaje como también lo es un Regresor Lineal o una Máquina de Vectores Soporte (SVM), sin embargo, lo que diferencia una Red Neuronal de estos modelos es que una red neuronal, como su nombre lo expresa, conforma una red interconectada de estas neuronas artificiales en las que pueden ser organizadas linealmente en una capa, donde cada neurona recibe el mismo vector de entrada, y/o *apiladas* en distintas capas como la que podemos observar en la Figura 2.7; donde la salida de una capa de neuronas es la entrada de las neuronas de la capa superior. Una red neuronal que tiene muchas capas, se dice que tiene más *profundidad*; cuanta mayor profundidad tiene una red neuronal esta es capaz de capturar características más complejas de los datos, y por tanto, un red neuronal más poderosa. Este tipo de red neuronal se denomina *Multi-Layer Perceptron* o Perceptrón Multicapa (MLP), haciendo referencia al primer modelo de aprendizaje desarrollado para clasificación binaria, el Perceptrón [29].

El hecho de poder estructurar las neuronas, que como hemos dicho son modelos de aprendizaje, nos permite obtener modelos de aprendizaje mucho más poderosos en cuanto a la complejidad de problema a resolver. El Deep Learning o Aprendizaje Profundo [30] es un subcampo del Machine Learning que justamente estudia el aprendizaje de estos modelos de ML de mayor

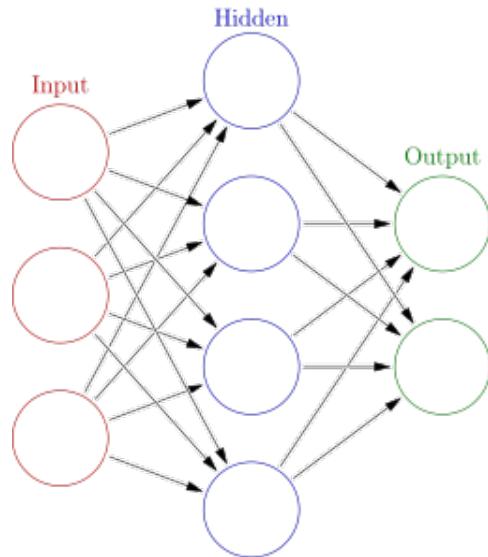


Figura 2.7: Ejemplo esquemático de red neuronal de dos capas ocultas. Imagen extraída de Wikimedia Commons [31].

profundidad, siendo de gran uso en tareas de Visión por Computador (CV) con la Redes Neuronales Convolucionales (CNN) y en el campo de Procesamiento del Lenguaje Natural (NLP) con las Redes Neuronales Recurrentes (RNN), entre otros.

2.1.6. Redes Neuronales Convolucionales (CNN)

Dado que en este TFG trabajaremos con problemas de Visión por Computador, resulta necesario al menos introducir el tipo de Redes Neuronales especializados en tareas de clasificación de imágenes, las Redes Neuronales Convolucionales o simplemente CNN, que son un tipo de red neuronal que palían las debilidades del MLP para clasificación de imágenes. El concepto de red neuronal convolucional surge de los trabajos de Yann LeCun [32] en el uso de Backpropagation [33] para poder aprender los parámetros de las capas convolucionales en una red neuronal para la clasificación de dígitos manuscritos. En la Figura 2.8 se muestra la red creada por LeCun que resuelve la tarea de dígitos manuscritos utilizando una red convolucional de cinco capas convolucionales que bautizo como LeNet-5 [34].

Las capas convolucionales son en esencia filtros que realizan operaciones matemáticas sobre la imagen para poder obtener características semánticas de ellas; en la Figura 2.9 podemos ver la operación de convolución aplicada a imágenes con un filtro 3×3 . En la Figura 2.10 podemos ver un ejemplo

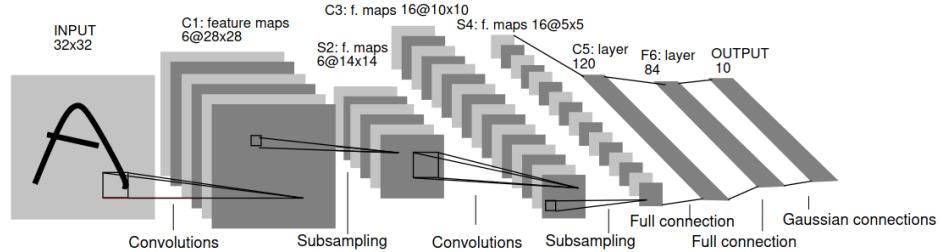


Figura 2.8: Arquitectura de LeNet-5 [34] una red neuronal convolucional, aquí para reconocimiento de dígitos manuscritos.

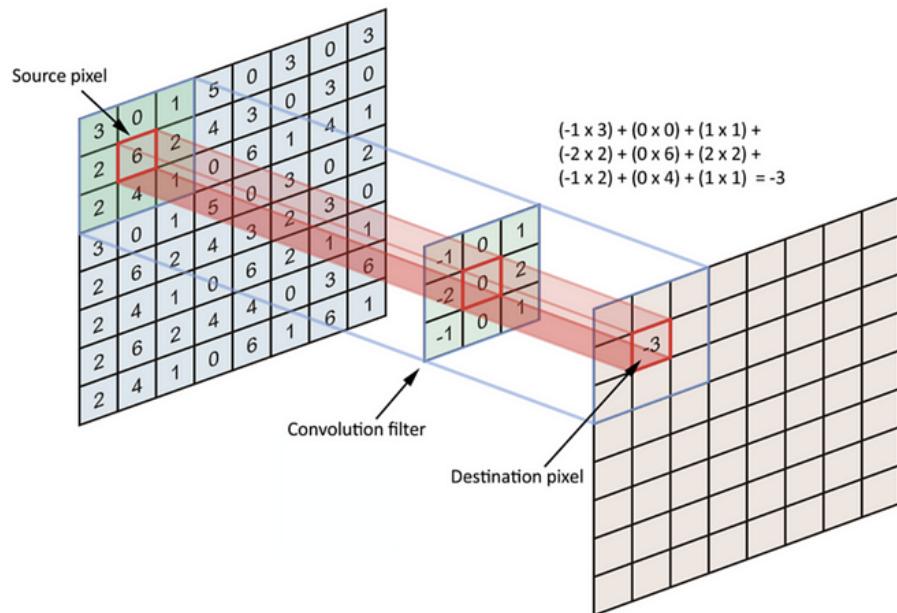


Figura 2.9: Operación de convolución aplicada sobre una matriz de valores enteros de entrada y un filtro convolucional 3×3 . Imagen extraída de TowardsDataScience [35].

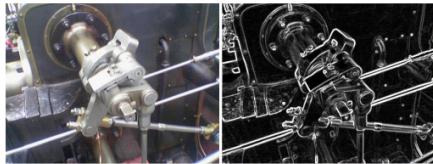
$$\mathbf{G}_x = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} \quad \mathbf{G}_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$


Figura 2.10: Filtro de Sobel para detectar bordes en imágenes. Imágenes extraídas de Wikimedia Commons [37]

de filtro o *kernel* llamado filtro de Sobel [36] que se utiliza para la detección de bordes en imágenes. Estos filtros o kernels antes se creaban manualmente en un proceso de extracción de características (*feature extraction*) y que requerían intervención humana; sin embargo, con la introducción de las redes convolucionales [32], estos filtros se aprenden automáticamente por la red, dando un proceso *end-to-end* de aprendizaje desde la imagen como entrada hasta la predicción o clasificación de salida.

2.2. Aprendizaje Federado (FL)

Dentro del campo del Machine Learning existen diferentes formas de entrenar modelos así como usando diferentes enfoques que resuelven problemas que el ML convencional, conocido como Machine Learning centralizado, carecen en ciertos escenarios. Uno de estos problemas son las situaciones donde se necesitan entrenar modelos de ML sobre cantidades muy grandes de datos, donde obviando el inherente problema de la organización de estos datos, para poder procesarlos se necesitan de suficientes recursos computacionales. Para este problema, se han llegado a proponer soluciones como el Machine Learning distribuido (*Distributed ML*), donde se distribuye la carga de trabajo sobre nodos de forma distribuida, de manera que se facilita la tarea de procesamiento de los datos, conllevando una ganancia en tiempo de entrenamiento de estos modelos. Otro de los problemas, sin embargo, que carece el Machine Learning Distribuido (y en general el Machine Learning con datos centralizados), son los escenarios donde se necesita **preservar la privacidad de los datos** donde estos son de información sensible o confidencial (e.g. hospitalares, correos electrónicos, ...).

El Aprendizaje Federado (FL) [4] es un paradigma del Machine Learning colaborativo y distribuido que resuelve estos dos problemas: distribución de la carga y descentralización de los datos. El Aprendizaje Federado es una serie de métodos y técnicas para poder entrenar modelos de Machine Learning de forma colaborativa y distribuida, y preservando la privacidad de los datos descentralizándolos entre las diferentes entidades participantes, referidos como *nodos* o *clientes*. La descentralización de los datos es un factor

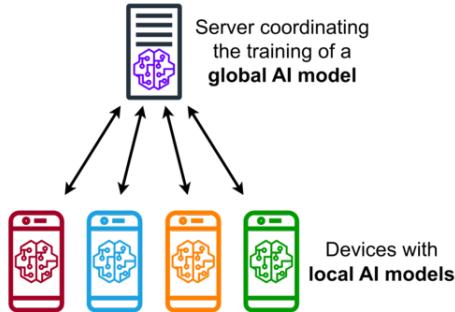


Figura 2.11: Esquema de Aprendizaje Federado Centralizado con teléfonos inteligentes entrenando cooperativamente un modelo de ML. Imagen extraída de Wikimedia Commons [38].

diferencial de esta aproximación del Aprendizaje Automático con el entrenamiento convencional de modelos en un entorno centralizado, en donde todos los datos están agrupados en un solo conjunto de datos; esto prevalece aún en ML distribuido donde, aunque los datos no estén físicamente centralizados, estos sí lo están virtualmente entre todos los nodos participantes. La diferencia entre estas dos metodologías distribuidas del ML reside en que el ML distribuido tiene como principal objetivo el paralelizar la carga computacional, mientras que el FL es el entrenamiento de modelos sobre conjuntos de datos heterogéneos [39], es decir, datos que no están generados de forma idéntica. Mientras que el aprendizaje distribuido conlleva el entrenamiento de modelos sobre múltiples nodos, mantiene la suposición de que los datos son generados de forma idéntica e independientemente distribuidos (datos IID) y se trata de mantener el mismo número de ejemplos entre los nodos. Esta suposición no se comparte con datos heterogéneos en FL, donde cada cliente tiene sus datos generados de forma diferente (no idénticamente distribuidos) y cada uno puede tener tamaños de datasets de diferentes órdenes de magnitud (no uniformemente distribuidos), esto hace que una de las principales características pero a su vez uno de sus más grandes problemas del aprendizaje federado sea que sus datos no cumplen la propiedad IID de los datos. De la misma forma que en FL tenemos datos heterogéneos, también lo son los propios clientes, donde cada uno puede poseer diferentes propiedades en cuanto a capacidad computacional (e.g. tener CPU o GPU) y de comunicación (e.g. ancho de banda) [8].

En Aprendizaje Federado, se tiene como objetivo el entrenamiento de modelos de ML sobre los conjuntos de datos locales de los nodos o clientes participantes. La idea está en que un modelo global es compartido entre todos los clientes, donde este modelo se entrenará localmente con los datasets privados de cada cliente; como lo podrían ser dispositivos de IoT o teléfonos

móviles. Posteriormente, estos modelos entrenados localmente, denominados usualmente *actualizaciones* del modelo, son intercambiados y agregados a un nuevo modelo global, llevado a cabo por una entidad central denominada *servidor*, encargada de orquestar el proceso de aprendizaje federado. Este esquema de Aprendizaje Federado con arquitectura típica de *cliente-servidor* es conocido como Aprendizaje Federado Centralizado (véase Figura 2.11, y es la forma estándar de aplicación de este paradigma de aprendizaje debido a su fácil adaptación a otras infraestructuras como el Mobile Cloud Computing o aplicaciones IoT. Aunque existen alternativas a esta arquitectura de Aprendizaje Federado como el Aprendizaje Federado descentralizado, donde no existe el concepto de servidor sino que la red de clientes se auto-orquesta, por ejemplo con la arquitectura de red *peer-to-peer*, nuestro trabajo se centrará en la configuración típica de cliente-servidor, y para la fácil lectura del trabajo, en adelante se denominará de forma indiferente el Aprendizaje Federado Centralizado como simplemente Aprendizaje Federado o FL.

El aprendizaje federado, es un proceso, fuera del entrenamiento efectivo del mismo modelo, *iterativo*. Cada iteración, o frecuentemente denominado *ronda*, de Aprendizaje Federado comprende los pasos de forma general mostrados en el Algoritmo 1 (véase Figura 1.1). Cada ronda de entrenamiento se realiza de forma síncrona, es decir, cada fase o paso en 1 se realiza una tras de otra y tras completarse todas las fases, se pasa a la siguiente ronda. El proceso de iteración se termina

Algoritmo 1 Protocolo estándar del entrenamiento de un modelo en FL.

- 1: **Inicialización:** El servidor inicializa el modelo global de partida de forma aleatoria o pre-entrenada con datos públicos.
 - 2: **Distribuir modelo global:** El servidor distribuye el modelo global de la ronda actual a los clientes participantes para poder ajustar el modelo a los datos locales de los clientes.
 - 3: **Actualización del modelo:** Cada cliente ajusta el modelo global con sus datos locales, dando como resultado un conjunto de actualizaciones del modelo entrenados con datos heterogéneos.
 - 4: **Subida de los parámetros:** Cada cliente participante sube sus actualizaciones del modelo, normalmente los gradientes calculados o directamente los nuevos parámetros del modelo, al servidor.
 - 5: **Agregación de las actualizaciones:** El servidor agrega los parámetros de cada cliente utilizando un operador de agregación, como el *FedAveraging* (FedAvg) [4], para constituir el modelo global de la siguiente ronda.
 - 6: **Terminación:** Se repiten los pasos 2-5 hasta que se cumpla un cierto criterio de parada o terminación. Por ejemplo, que se haya llegado a un máximo de iteraciones, se haya alcanzado cierto rendimiento del modelo global, etc.
-

2.2.1. Notación y formulación

Expondremos el escenario de Aprendizaje Federado formalmente, estableciendo la notación y símbolos que formulan de forma matemática el FL. Nótese que esta notación varía de forma sutil entre otros autores de otros trabajos relacionados con Aprendizaje Federado, sin embargo, muchos coinciden en la siguiente formulación. Supongamos que tenemos datos etiquetados de la forma (x, y) , donde $x \in \mathcal{X}$ es el vector de entrada (*features*) e $y \in \mathcal{Y}$ la etiqueta del dato de entrada, y una predicción de esta etiqueta $f(x, w) = \hat{y}$. Supongamos también una entidad *servidor* con vastos recursos que entrena un modelo $w^{(t)} \in \mathbb{R}^d$ para una cierta ronda de entrenamiento t , sobre los datos locales de un conjunto de K clientes de recursos restringidos. Cada cliente $k \in [K] = \{1, 2, \dots, K\}$, posee un dataset privado local \mathcal{D}_k con un tamaño de n_k datos, que usarán para entrenar el modelo distribuido $w^{(t)}$ y generar una actualización de ese modelo $w_k^{(t+1)}$ minimizando la pérdida del modelo con el dataset local del cliente k , $l(y, f(x, w_k^{(t)}))$. Si $\mathcal{D}_{\text{train}}$ es la unión de todos los datasets de los clientes $\bigcup_{k \in [K]} \mathcal{D}_k$, tal que tenemos n_{train} ejemplos; entonces, obtenemos un modelo global $w^{(t+1)}$ a partir de las actualizaciones locales de los clientes¹, tal que minimicen la pérdida global de entrenamiento 2.4, donde $l(y, f(x, w))$ es el error en un único dato. El servidor agrega las actualizaciones de los modelos en la ronda t utilizando un operador de agregación, $w^{(t+1)} = F(\{w_k^{(t)} : k \in [K]\})$. Este trabajo al poner su foco en la selección de clientes, usaremos el agregador estándar **FedAvg** [4], $w^{(t+1)} = F(\{w_k^{(t)} : k \in [K]\}) = \sum_{k \in [K]} \frac{n_k}{n_{\text{train}}} w_k^{(t)}$. Este proceso se realiza de manera iterativa para un número de rondas $T \in \mathbb{N}$.

$$\begin{aligned} \mathcal{L}(w; \mathcal{D}_{\text{train}}) &= \frac{1}{n_{\text{train}}} \sum_{(x,y) \in \mathcal{D}_{\text{train}}} l(y, f(x, w)) \\ &= \sum_{k \in [K]} \frac{n_k}{n_{\text{train}}} \sum_{(x,y) \in \mathcal{D}_k} \frac{1}{n_k} l(y, f(x, w)) \\ &= \sum_{k \in [K]} q_k \mathcal{L}(w; \mathcal{D}_k) \end{aligned} \quad (2.4)$$

En [4] muestra que es posible reducir este error de pérdida 2.4 obteniendo el modelo global con **FedAvg**, si y solo si, para todas las actualizaciones, el modelo se inicializa de la misma forma (mediante una inicialización con la misma semilla), produciendo una «reducción significativa del error en el dataset de entrenamiento global $\mathcal{D}_{\text{train}}$ » ([4]).

¹Suponemos en este caso que los clientes participantes en la ronda actual t son todos los K clientes. Cuando veamos con más detalle el concepto de selección de clientes veremos que los clientes participantes pueden ser un conjunto menor $S_t \subseteq [K]$ de clientes.

2.3. Selección de clientes en FL

Para terminar este capítulo de los fundamentos y conocimientos teóricos necesarios para la correcta compresión del lector hacia este TFG, vamos a concretar el concepto de *Selección de clientes* en Aprendizaje Federado.

El aprendizaje federado a servidor como el *framework* de Machine Learning más usado en aplicaciones distribuidas como IoT o Cloud Computing, en el que la privacidad de los datos es un factor importante y crucial. En años recientes, el número de fuentes de datos distribuidas por el mundo, tales como redes sociales o IoT, y que pueden contribuir al entrenamiento de estos modelos con FL, ha incrementado significativamente [40]. Este incremento afecta en gran medida el cuadro del Aprendizaje Federado, puesto que cuando tenemos un número inmenso de clientes (lo que correspondería a fuentes de datos), ya no es viable el entrenamiento de todos estos K clientes en cada ronda; esto se debe, entre otros pero más en especial, a uno de los grandes problemas abiertos en Aprendizaje Federado: **la sobrecarga de la comunicación entre los clientes y el servidor**. Ya desde la introducción a este *framework* del Aprendizaje Federado [4] se ha estudiado el comportamiento en el entrenamiento de estos modelos cuando el número de clientes crecía, mostrando un gran deterioro debido a la sobrecarga que involucra la distribución y subida de los modelos entre el servidor y el cliente. En [8] se ha estudiado que esta sobrecarga se agrava ante escenarios donde los recursos de los clientes tienen un nivel alto de heterogeneidad.

Para solucionar este cuello de botella que induce la comunicación entre los clientes y el servidor, se han propuesto dos formas de hacer frente a este problema. Una de las primeras soluciones era directamente proponer estrategias de optimización en los canales de comunicación. En [41] se exploran técnicas de compresión para mejorar la eficiencia en las comunicaciones teniendo una compensación con el rendimiento del modelo. Otra aproximación sobre este enfoque es [42], donde realizaban una optimización de hiper-parámetros de FL como el número de épocas de entrenamiento en cada fase de actualización de modelos o el número total de épocas. Sin embargo, como apunta [8], estas propuestas «no consideran particularmente escenarios con recursos de comunicación y computación y/o datos en clientes heterogéneos».

El segundo enfoque para tratar el problema de la sobrecarga en las comunicaciones es sencillamente reducir el número de veces en las que el servidor tiene que comunicarse con los clientes. Ya desde la creación de FedAvg (y por tanto de FL) en [4], se había diseñado el protocolo de *Federated Averaging* con el objetivo de reducir el número de rondas necesarias para llegar a cierto rendimiento del modelo, lo que reducía directamente las comunicaciones entre servidor y clientes, esto aumentando la computación en los clientes

(i.e. aprovechar las capacidades computacionales de los clientes entrenando de forma local un número más elevado de veces para reducir las rondas totales). Sin embargo, esto seguía sin ser viable con grandes magnitudes de clientes y especialmente con clientes y datos heterogéneos. Esto dio paso a una gran variedad de propuestas para poder reducir el número de rondas totales (i.e. aumentar la convergencia del modelo en FL) comprendiendo un número de clientes menores, un subconjunto $S_t \subseteq [K]$ para cada ronda de entrenamiento, lo que reduciría el número de comunicaciones entre los clientes a la vez que se consigue un buen rendimiento del modelo final; a lo que hoy en día conocemos como estrategias de Selección de Clientes.

La selección de clientes, en general sigue el esquema del Algoritmo 2 que es una modificación del protocolo estándar de Aprendizaje Federado 1, en el que se introduce una fase de **Selección de Clientes** previa a la actualización del modelo localmente en los clientes, de manera que toda carga de computación (entrenamiento del modelo) y la comunicación se realice siempre sobre un subconjunto de los clientes totales $S_t \subseteq [K]$, siguiendo un cierto criterio de selección. Es de recalcar que el protocolo seguido en 2 puede variar significativamente en las diferentes estrategias que trataremos en este trabajo. Este esquema solo es una vista general del flujo que se suele seguir en este tipo de métodos y nos sirve simplemente como esquema inicial para comprender el rol de estos algoritmos en Aprendizaje Federado.

Algoritmo 2 Protocolo general seguido en una estrategia de selección de clientes en FL. Las fases estilizadas en *italica* son opcionales y dependen del método de selección.

- 1: **Inicialización:** El servidor inicializa el modelo global de partida de forma aleatoria o pre-entrenada con datos públicos.
 - 2: *Inicialización de los clientes:* Dependiendo del método de selección de clientes, se inicializan variables que toman parte en la selección de los clientes (paso siguiente).
 - 3: **Selección de clientes:** El servidor selecciona un subconjunto $S_t \subseteq [K]$ de clientes que participarán en la ronda actual t para entrenar localmente el modelo global.
 - 4: **Distribuir modelo global:** El servidor distribuye el modelo global de la ronda actual a los clientes participantes para poder ajustar el modelo a los datos locales de los clientes.
 - 5: **Actualización del modelo:** Cada cliente ajusta el modelo global con sus datos locales, dando como resultado un conjunto de actualizaciones del modelo entrenados con datos heterogéneos.
 - 6: **Subida de los parámetros:** Cada cliente participante sube sus actualizaciones del modelo, normalmente los gradientes calculados o directamente los nuevos parámetros del modelo, al servidor.
 - 7: **Agregación de las actualizaciones:** El servidor agrega los parámetros de cada cliente utilizando un operador de agregación, como el *FedAveraging* (FedAvg) [4], para constituir el modelo global de la siguiente ronda.
 - 8: *Post-selección:* En esta fase al igual que el paso 2, depende del algoritmo de selección de clientes; para el que se actualizan las variables que toman parte en la selección de los clientes de la ronda siguiente $t + 1$.
 - 9: **Terminación:** Se repiten los pasos 2-5 hasta que se cumpla un cierto criterio de parada o terminación. Por ejemplo, que se haya llegado a un máximo de iteraciones, se haya alcanzado cierto rendimiento del modelo global, etc.
-

Capítulo 3

Estado del Arte

Desde su introducción alrededor de 2017 ([4]), el Aprendizaje Federado ya prevenía los problemas que hemos comentado en la sección anterior 2 sobre el deterioro rápido de los modelos entrenados cuando el número de clientes crece, debido a la sobrecarga de comunicación. Esto ha dado pie a una gran cantidad de propuestas de lo que conocemos como métodos de Selección de Clientes.

Este campo o vía de estudio en aprendizaje federado es relativamente nuevo y por tanto emergente. Para el año 2021 se han podido registrar más de 500 métodos de selección de clientes, recopilados en el trabajo de [43], donde muestra además similitudes y diferencias. Sin embargo, hasta ahora no se ha llegado proponer una taxonomía estándar o reconocida a nivel general para la gran variedad de métodos existentes. De hecho, este trabajo solo recopila unos pocos métodos que hemos considerado y han demostrado tener buenos resultados en la literatura, sin embargo, el estudio exhaustivo de cada tipo de método de selección queda fuera del alcance de este TFG. Sin embargo, es apropiado dar una vista general de los distintos tipos de selección de clientes, sus diferentes enfoques y características que los diferencian, de manera que podamos introducir los métodos que hemos seleccionado para realizar este trabajo.

3.1. Consideraciones sobre los tipos de selección

El estudio exhaustivo de métodos de selección en [43] recopila una gran variedad de estos algoritmos. Sin embargo, muchos de estos algoritmos son aplicados a *distintos* tipos de Aprendizaje Federado como los que hemos visto en la introducción de este TFG 1. Es necesario entonces, filtrar aquellos algoritmos que no aplican al entorno o tipo de Aprendizaje Federado que nos interesa y bajo que suposiciones éstos se desarrollan.

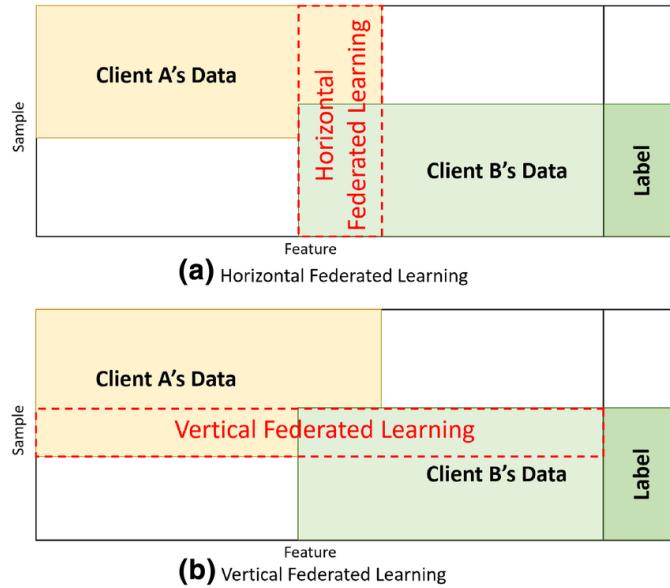


Figura 3.1: Visualización de los dos tipos de aprendizaje federado: Horizontal (a) y Vertical (b). Imagen extraída de [44].

Para empezar, podemos diferenciar dos tipos de aprendizaje federado y que se han desarrollado por separado en la literatura:

- **Aprendizaje Federado Vertical.** Este tipo de aprendizaje federado se caracteriza por establecer que los clientes puedan compartir un solo conjunto de datos sobre todos los clientes (comparten los ejemplos), pero cada cliente posee un espacio de características *distinto*. Este tipo de aprendizaje federado entrena modelos de Machine Learning sobre conjunto de datos limitado al solapamiento de los clientes, pero aprovechando todo el espacio de características de los clientes (véase Figura 3.1 (b)).
- **Aprendizaje Federado Horizontal.** Este tipo de aprendizaje federado, en contra parte con el vertical, considera un solo espacio de características (en concreto la intersección del espacio de todos los clientes) pero también considera *todos* los ejemplos de los clientes (véase Figura 3.1 (a)). De este modo, este tipo de FL entrena modelos sobre la unión de todos los datasets locales de los clientes pero sobre un solo espacio de características. Este tipo de aprendizaje es el más usado a nivel general dado que considera todos los datos de los clientes, mientras que preserva una baja complejidad al no tomar todas las características de los clientes (método regularizador). Además, este tipo de aprendizaje funciona en todos los casos donde no hayan solapamiento de los datos, y por tanto, tener una privacidad total de los datos en los clientes.

Ahora tomando en consideración la *naturaleza* de los clientes, en [43] se diferencian dos tipos de arquitecturas de FL distintos entre sí: el primero de ellos es *cross-silo*, en donde los clientes se esperan que estén disponibles a la participación en cada fase del aprendizaje. Mientras que el segundo, *cross-device*, los clientes son más individuales e independientes, y que pueden o no participar en el aprendizaje por diversas razones (e.g. perdida de conectividad). Dado estos dos tipos podemos observar que en el caso de *cross-silo*, se suele utilizar en organizaciones con un número de clientes limitado donde se pueda dar garantía de estos clientes en su disponibilidad; mientras que *cross-device* es más flexible en su aplicación ya que puede considerar cualquier tipo de cliente de distinta índole como es en el caso de clientes heterogéneos. De aquí también podemos observar que en el caso del aprendizaje federado vertical, solo se puede aplicar *cross-silo*, debido a que el espacio de características se comparte con *todos* los clientes. Mientras que en aprendizaje federado horizontal, si un cliente no está disponible, simplemente no se aprovechan sus datos, pero el aprendizaje es aún posible [45].

Dado los distintos tipos de aprendizaje federado que existen¹, en este trabajo solo consideraremos los métodos de selección basados en aprendizaje federado horizontal, y teniendo como suposición clientes *cross-device*. Esta decisión viene a raíz de la utilidad real de estos tipos de aprendizaje, aunque reconocemos que el aprendizaje federado vertical es un tipo que está emergiendo muy rápido [45], el aprendizaje federado horizontal es en gran medida utilizado por su fácil adecuación a la mayoría de problemas, y por tanto, las contribuciones que se den a tal ámbito se esperan tener un gran impacto. La decisión por la suposición de clientes *cross-device*, es de igual manera, debido a su aplicación real y en la importancia del estudio sobre entornos de clientes heterogéneos, que están presentes en muchas de los ámbitos donde se aplica aprendizaje federado [8].

3.2. Taxonomía de métodos de selección de clientes

En años recientes, han habido muchas propuestas de selección de clientes, muchos con un amplio espectro de objetivos, requerimientos, protocolos, etc. Unas enfocándose en la selección *justa* de los clientes (*fairness*) ([46]), otras basadas en la valuación, bajo un cierto criterio, de los clientes ([47], [48]); y algunas bajo ciertas condiciones y restricciones de tiempo ([8]). Este auge tan repentino ha resultado en que no se haya formalizado una taxonomía o al menos una forma común y aceptada por todos los autores de una clasificación

¹Cabe recalcar que esto es para el caso centralizado, con la participación de un servidor central.

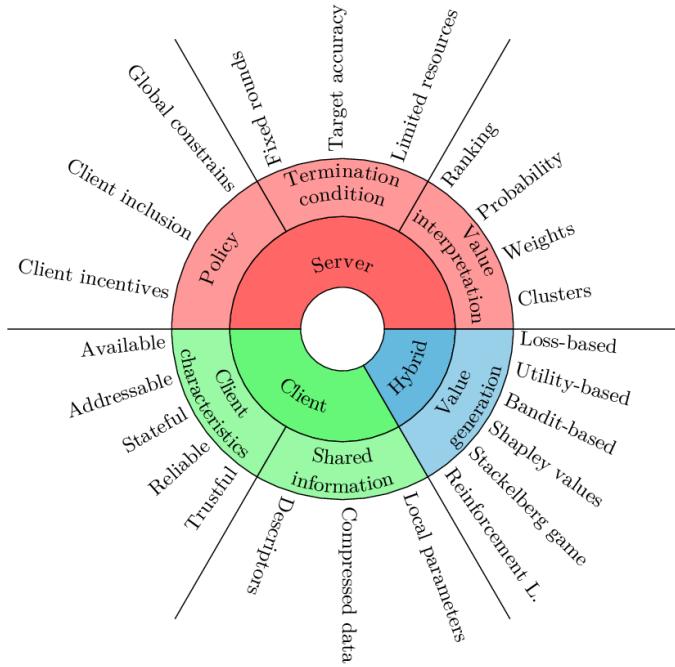


Figura 3.2: Taxonomía propuesta por [7]. La taxonomía divide los algoritmos de selección sobre seis dimensiones: Policy, Termination Condition, Value Interpretation, Client Characteristics, Shared Information, y Value Generation. Que a su vez van separadas dependiendo si conciernen al servidor (marcadas en color rojo), a los clientes (marcados en verde) o a ambos (azul). Imagen extraída del mismo paper [7].

de estos algoritmos y de los siguientes que vendrán. Esto incluso limita a los desarrolladores que usan este tipo de técnicas en sus entornos de aprendizaje federado en ubicarse entre las mejores propuestas de selección con resultados del estado del arte, y más importante aún, saber cuáles de los distintos métodos son los más adecuados para su aplicación concreta.

Últimamente han habido muchas propuestas de taxonomías para las estrategias de selección de clientes en aprendizaje federado. Una muy completa basada en el trabajo de [43], es la taxonomía de Németh et al. [7]. En esta taxonomía se divide la clasificación sobre seis dimensiones como se pueden observar en la Figura 3.2. Cada dimensión refiere a una o varias características que pueden tener las estrategias de selección, teniendo estas características de distintas dimensiones. Al mismo tiempo, estas dimensiones están categorizadas en cuanto a que son concernientes al servidor, a los clientes o a ambos (híbrido). Esta taxonomía permite enfocar el interés en ciertos algoritmos que cumplen con las características buscadas, por ejemplo, un practicante de aprendizaje federado puede tener interés en cuanto a

métodos de selección sobre un entorno de recursos limitados; con esta taxonomía es fácil ver que métodos en la literatura cumplen este requisito bajo la dimensión de **Termination Condition**.

Bajo la taxonomía vista en la Figura 3.2, podemos resumir cada dimensión en lo siguiente:

- **Policy.** La selección de clientes sobre esta dimensión se caracteriza por definir ciertas reglas o *políticas* que deben de cumplir los clientes para ser seleccionados o políticas sobre el mismo entrenamiento; por tanto, es lógico que esto sea tarea del servidor. Entre las distintas vías que se pueden tomar se distinguen:
 - Restricciones globales: como en la eficiencia en el entrenamiento, reduciendo el número de clientes seleccionados mientras se mantiene la misma tasa de convergencia ([49]) o imponiendo restricciones de tiempo de entrenamiento ([8]).
 - Políticas de inclusión de clientes: este tipo de política es muy común en selección de clientes, dado que intentan promover la participación de clientes que no son tan relevantes como los demás (e.g. datos infra-representados). Son notorios los trabajos sobre la inclusión de clientes con canales de comunicación pobres, impulsado por el problema de la sobrecarga en la comunicación discutido en el capítulo anterior 2. En ellos, los clientes con peores recursos son involucrados a entrenar sobre modelos más *fáciles*, de manera que el entrenamiento sea eficiente (Federated Dropout, [50]); o imponiendo una tolerancia en la pérdida de paquetes de comunicación (LT-FL, [51]).
 - Políticas de incentivos a los clientes: este tipo de política introduce *mecanismos de incentivos* a los clientes a participar en la federación. Esto es especialmente útil sobre aprendizaje federado *cross-device*, donde los clientes pueden o no participar por razones desconocidas (e.g. desconexión de dispositivos). Existen distintas formas y mecanismos de incentivos que van desde dar recompensas a los clientes por contribución de sus datos (datos de muy buena calidad), como son la gran gama de selección basados en valores de Shapley ([52]), hasta la capacidad de recursos que poseen los clientes, como dar recompensa a los clientes basado en su poder o uso computacional ([53]), uso energético, disponibilidad de recursos, etc.
- **Termination Condition.** Esta segunda dimensión es definida por el servidor y determina el criterio o condición de parada de todo el proceso de entrenamiento en Aprendizaje Federado. Muchas propuestas implementan sus experimentos, y por tanto sus conclusiones en base a

los resultados obtenidos, con distintas aproximaciones en base a esta condición de parada, por ser un factor importante en la configuración y diseño de estos experimentos. Se distinguen tres grandes categorías:

- En la primera se eligen un número **fijo de rondas** T . Esto es especialmente útil cuando se quieren comparar distintos métodos sobre una misma condición de parada, y determinar cual de ellos obtiene un mejor modelo. En efecto, es útil para medir y comparar la tasa de convergencia entre métodos. Sin embargo, no es el la condición preferida en situaciones reales, dado que los modelos no suelen llegar a su mejor rendimiento dado un T fijo de rondas, además de depender de varios factores.
 - La segunda consiste en prefijar una **mínima precisión del modelo** (*accuracy*) para el que se observa en cuanto tiempo (por rondas o por tiempo de reloj) un método de selección es capaz de llegar. Al igual que el anterior, es útil para comparar distintos algoritmos entre sí, más no es recomendable usarlo en situaciones reales. Se puede usar un método *baseline* (como entrenamiento centralizado o `RandomSampling`) para establecer esta métrica mínima como lo han hecho en [8] y [54].
 - La última consiste la **limitación en la participación** de los clientes. Se trata de una alternativa a las dos tradicionales anteriores en la que se da terminación llegado a una participación mínima de clientes, por ejemplo, agotando una cantidad de recompensas limitadas hacia los clientes ([55]).
- **Client Characteristics.** Esta dimensión clasifica a los métodos de selección por una serie de características predefinidas que cumplen los clientes según especifica [43]. En ella tenemos las características de: Disponibilidad, Direccionabilidad (i.e. como acceder a ellos, por ejemplo, con identificadores únicos), si mantienen o no estado (i.e. permitir o no que los clientes participen en múltiples rondas manteniendo el estado de la ronda anterior en el modelo), Fiabilidad (si el cliente aún cumpliendo con toda la *criteria* no es capaz de comunicarse con el servidor o de actualizar el modelo [7]). En [7] agregan una característica extra con respecto a la **Confianza** del cliente, para el que se suponen que existen clientes maliciosos que puedan perjudicar el ajuste del modelo (clientes bizantinos [56]).
 - **Shared Information.** En esta dimensión, clasifica los métodos de selección con respecto a qué información comparten los clientes con el servidor o entre ellos mismos. La información más común compartida por los clientes son sus parámetros actualizados del modelo global, sin embargo, hay métodos de selección que comparten otro tipo de

información como: la pérdida del modelo local ([57]), recursos computacionales y de comunicación ([8]), o incluso los datos locales del propio cliente ([58]).

- **Value Generation.** Esta dimensión comprende un subconjunto de métodos de selección de clientes que enfocan su estrategia de selección en la valuación (creada por el servidor) de los clientes. La función que asigna la valuación de cada cliente puede variar significativamente y depende de los objetivos que se quieran conseguir. Por ejemplo, [57] enfoca la valuación de los clientes directamente con la pérdida en entrenamiento del modelo local (además de un término de penalización en base a sus datos); otros autores toman la pérdida como un factor de entre más métricas para medir la *utilidad* de un cliente ([54];[48]); y otros más recientes valúan los clientes en base a la contribución marginal de cada uno con el modelo global utilizando valores de Shapley [52] ([47];[59];[60]). Hay varias propuestas que toman conceptos de otras ciencias como la economía con el *juego de Stackelberg* [61] ([53]) u otros tipos del aprendizaje automático como el *aprendizaje por refuerzo* ([62]).
- **Value Interpretation.** Esta dimensión puede considerarse como la continuación del anterior **Value Generation**, para el que luego de que se hayan calculado (por parte del servidor) las valuaciones de los clientes dado un/os término/s, estas se deben de interpretar para poder realizar la selección de los clientes. La forma más intuitiva de interpretación es realizando un **ranking**, a partir del cual se eligen los mejores n clientes de ese ranking en base a sus valuaciones utilizando un algoritmo *greedy* ([8]). También se pueden interpretar como la **probabilidad** de ser seleccionados, muestreando de una distribución de probabilidad construida a partir de estas valuaciones ([57];[59]). Otros autores interpretan estas valuaciones como ponderaciones sobre los clientes en la fase de agregación del modelo, dando como resultado una suma ponderada en lugar de una media uniforme como en FedAvg [4].

3.3. Clasificación basada en el problema a resolver

La taxonomía propuesta por [7] es muy útil para explorar distintas estrategias de selección de clientes bajo distintas *dimensiones*. Sin embargo, este trabajo aunque no se centra en explorar todos los algoritmos de selección (ya que sería una tarea ardua) si que se pretende explorar métodos de selección que sean *diferentes* entre sí y que tengan características propias distintivas para una mejor exploración general en la selección de clientes. Esto, con la

taxonomía de [7] no parece ser posible, esto debido a que las dimensiones solo categorizan métodos bajo ciertas características y particularidades; asemejándose a ver todos los algoritmos pero con distintos lentes (dimensiones). Esto da pie a que un método de selección pueda estar presente en más de una dimensión (o incluso en todas). Por ejemplo, FedCS [8], es un algoritmo que se caracteriza por imponer una política de restricción global (concretamente imponer *deadlines* de tiempo de entrenamiento y subida de modelos), pero también en cuanto a su criterio de parada (llegar a un mínimo de *accuracy* mediante un *baseline*).

Dado que nuestro objetivo es poder elegir métodos de selección que ofrezcan soluciones distintivas, nos basaremos en el estudio de [63] para elegir los algoritmos que analizaremos e implementaremos en este trabajo. En este artículo se realiza un estudio exhaustivo de métodos de selección de clientes en aprendizaje federado que han mostrado tener resultados del estado del arte en el campo, incluyendo sus fortalezas y debilidades. Esto lo realiza primero separando su estudio en métodos basados en el problema que estos intentan resolver, o en otras palabras, en el *enfoque* de interés en el que estos se desarrollan. Concretamente, este estudio se distinguen tres enfoques que nos centraremos en este TFG:

- **Selección Convencional.** En este primer tipo se consideran algoritmos que dependen totalmente o casi totalmente de información global y compartida de los clientes para hacer frente a los problemas de coste de comunicación y datos no-IID, sin embargo, la selección propia de los clientes participantes se realiza siguiendo un esquema aleatorio, selección conocida como *Random Sampling*. El muestreo aleatorio de clientes tiene como objetivo tener una participación representativa de todos los clientes de manera igualitaria (se seleccionan de forma uniforme). El algoritmo más importante utilizando Random Sampling es el mismo FedAvg propuesto por el trabajo pionero en el campo de Federated Averaging ([4]), donde demuestra convergencia en aprendizaje federado promoviendo la computación en los clientes por encima de la comunicación entre el servidor y los clientes (i.e. aumentando el entrenamiento local en los clientes). Sin embargo, este tipo de algoritmos no toman en cuenta la heterogeneidad tanto de los clientes como de los mismos datos que están presentes en la gran mayoría de problemas reales con aprendizaje federado ([60]), haciéndolo subóptimo para estas casuísticas.
- **Métodos basados en el rendimiento.** Estos métodos de selección se caracterizan por enfocarse enteramente en converger el mejor modelo global en el **menor** número de rondas, es decir, algoritmos que buscan la velocidad de convergencia. Para ello, realizan una selección de clientes para el que estiman que el modelo resultante de esa ronda

da de entrenamiento sea de mejor rendimiento. Estos métodos suelen premiar a aquellos clientes que mejor contribuyan al aprendizaje del modelo, y por tanto, tienen una preferencia o sesgo hacia ellos. Active Federated Learning [57], valía cada cliente basándose en que tan valioso es sus actualizaciones del modelo, por ejemplo, tener un modelo entrenado con datos infra-representados o una pérdida mayor mejorará en la generalización del modelo y por tanto, convergerá más rápido seleccionando estos clientes. Otros algoritmos como los basados en valores de Shapley como UCB-CS [47], S-FedAvg [59] o GreedyFed [60]; seleccionan clientes mediante un ranking basado en sus valores de Shapley cumulativos [52], los cuales miden la contribución marginal o *importancia* de un cliente al modelo agregado. Estos algoritmos tienen la ventaja de ser especialmente eficientes en entornos con un *alto índice* de datos no-IID.

- **Métodos basados en los recursos de los clientes.** Estos métodos de selección implementan técnicas en las que basan en los recursos computaciones y de comunicación de los clientes para poder seleccionarlos. En este enfoque los trabajos de Nishio y Yonetani con FedCS [8] han marcado los primeros pasos en la selección de clientes en cuanto a restricciones en el tiempo de entrenamiento. Para ello, se seleccionan clientes estimando el tiempo que tardarán en completar una ronda, para luego seleccionar la mayor cantidad de clientes participantes sujetos a esta restricción de tiempo. En base a FedCS, han salido otros métodos como HybridFL [58], que además de implementar FedCS, trata de construir un dataset IID a partir de los datos de los clientes, induciendo la propiedad de datos independientemente e idénticamente distribuidos para solventar o al menos sanitizar el problema de datos no-IID. Además de estos mecanismos, existen otros algoritmos que implementan mecanismos para incentivar a clientes, con pocos recursos pero que poseen datos valiosos, en la federación; en esta línea son notorios los trabajos relacionados con el juego de Stackelberg [61] como [64], para la optimización en el uso de los recursos.

Finalmente, habiendo visto esta clasificación más disjunta entre los distintos algoritmos de selección de clientes en la literatura tomaremos los siguientes algoritmos por enfoque:

- **Selección Convencional:** tomaremos el algoritmo de **Random Sampling** [4] el cual será utilizado como método *baseline* en la comparación de los métodos de selección utilizados.
- **Métodos basados en el rendimiento:** para este enfoque tomaremos dos algoritmos: **Active Federated Learning** [57] y **GreedyFed** [60],

los cuales representan por un lado un método de selección basado en la pérdida del modelo por cliente y por el otro un algoritmo basado en los valores de Shapley (SV) que ha mostrado mejorar otros algoritmos basados en este tipo de valuación.

- **Métodos basados en los recursos de los clientes:** por último, en este enfoque consideraremos el algoritmo de **HybridFL** [58] por estar basado en FedCS [8], un algoritmo muy importante sobre este enfoque. Y un segundo algoritmo propio basado en HybridFL que intenta mejorar una debilidad de este último; este método se ha bautizado como **Dyn-HybridFL** y al igual que los demás algoritmos seleccionados, se explicarán más a detalle en la implementación de cada uno de estos (Capítulo 5).

En el estudio de [40] se denota otro enfoque relacionado con la seguridad del modelo, especialmente en la tolerancia sobre clientes bizantinos [56]. Sin embargo, no lo consideraremos en la implementación de este TFG por cuestión de los métodos utilizados los cuáles explicamos con detalle en el capítulo 4.

Capítulo 4

Metodología

En la realización de un trabajo de investigación con cierto nivel de rigurosidad es importante y crucial poder describir de forma detallada la metodología utilizada para llevar a cabo el estudio del que se basa el trabajo de fin de grado. Este capítulo detallará todos los componentes en los que se constituye este trabajo así como la metodología seguida para poder llevarlo a cabo con el máximo nivel de rigurosidad de una manera clara y concisa para el mejor entendimiento del lector. Este capítulo, a diferencia de los anteriores y posteriores, tiene una carga teórica más alta que es necesaria para detallar con la mínima ambigüedad posible los métodos y materiales relacionados al aprendizaje automático y federado que se utilizarán, así como las estrategias de selección elegidas. También se describirá el entorno software en el que se montan tanto los algoritmos de selección como los experimentos realizados. Por último, se describirá el diseño experimental usado, así como las métricas de rendimiento e hipótesis sobre las estrategias de selección en base a estas métricas que esperamos observar en los resultados de los experimentos.

Para poder explicar todos estos métodos de una manera organizada, este capítulo se organizará en estas secciones:

1. **Enfoque del estudio.** En esta sección daremos una descripción general del enfoque utilizado para llevar a cabo este trabajo de investigación.
2. **Datasets utilizados.** Se describirán los conjuntos de datos utilizados en el entrenamiento de los modelos utilizando aprendizaje federado así como su preprocesamiento y su distribución.
3. **Modelos de aprendizaje.** Como continuación de la descripción de los datos, describiremos los modelos de Machine Learning utilizados para ajustar estos datasets.

4. **Herramientas y entorno de desarrollo.** Explicaremos las herramientas software que hemos utilizado para llevar a cabo la implementación del entorno de aprendizaje, las estrategias de selección y el entorno de experimentación.
5. **Estrategias de selección de clientes.** Aquí detallaremos las estrategias de selección de clientes elegidos en el capítulo 3¹, así como exponer sus fortalezas, debilidades, características y parámetros de configuración.
6. **Diseño de los experimentos.** Finalmente, en esta sección se expondrán todos los detalles relacionados al plan de experimentación de los algoritmos de selección: métricas de evaluación, selección de hiperparámetros, gráficas usadas, etc.

4.1. Enfoque del estudio

Como hemos podido ver en el capítulo 1, uno de nuestros objetivos primordiales es no solo el análisis de los métodos de selección de clientes que hemos elegido en el capítulo 3, sino en la comparativa entre ellos. Es por esto, que el enfoque de este trabajo de investigación es en el **experimental comparativo** de los algoritmos de selección elegidos. Este enfoque nos servirá para poder evaluar de forma experimental el rendimiento de los métodos de selección y en base a los resultados obtenidos poder determinar el mejor de ellos.

Es importante notar, sin embargo, que no vamos a comparar los algoritmos entre todos, esto debido a que como hemos comentado en el capítulo 3, hemos seleccionado un par de estrategias enfocadas en objetivos diferentes y que tienen características que no podemos comparar entre sí por ser disjuntas. Dicho esto, nuestro enfoque se centrará en la comparativa de métodos de selección en base al objetivo o problema que tratan de resolver, concretamente compararemos por un lado los métodos enfocados en el *rendimiento del modelo*, y por otro los enfocados en los *recursos de los clientes*. En la sección 4.5 veremos con detalle las diferencias entre los dos enfoques para de esta forma responder a la cuestión de la comparación separada y disjuntas de estos dos subconjuntos de estrategias.

Además de ser un estudio experimental de carácter comparativo, también nos centraremos en menor medida en el **análisis del diseño** del diseño de cada estrategia. Esto nos dará una mejor comprensión de las decisiones de

¹Nótese que no pretendemos explicar los detalles de implementación, que vendrán explicados en el capítulo 5, sino que explicaremos los detalles teóricos y de diseño de estas estrategias.

diseño del algoritmo así como entender el impacto visto en los resultados experimentales. Este análisis lo veremos en más detalle en la sección 4.5 de este capítulo, donde daremos una descripción detallada de cada estrategia de selección que estudiaremos.

Aunque este trabajo tiene un carácter más científico y de investigación, consideramos apropiado las decisiones y el proceso tomado en el desarrollo e implementación de los métodos de selección propuestos. Actualmente, en el campo del aprendizaje federado están surgiendo propuestas de *frameworks* o marcos de trabajo para desarrollar entornos de aprendizaje federado ([65]; [66]). Esta tarea no es trivial dado que muchos de los algoritmos vistos en este trabajo necesitan un nivel de flexibilidad mayor a la hora de establecer el flujo de entrenamiento de un modelo de ML; además, dado que el aprendizaje federado está diseñado para ser desplegado en entornos distribuidos, esto lo hace más complicado el proponer librerías y *frameworks* que ofrezcan todos los requisitos necesarios para suplir las demandas de los usuarios y desarrolladores que implementen este tipo de aprendizaje. Este trabajo *no* se enfocará en el diseño e implementación de un *framework* de FL, y se optará por el *uso* de uno de ellos, FLEXible [67], desarrollado por el Instituto Andaluz Interuniversitario en Data Science and Computational Intelligence (Dasci) [68], que describiremos más en detalle en este capítulo. Este TFG se enfocará por tanto en los detalles técnicos de implementación de estos algoritmos de selección de clientes para estudiar una de las muchas propuestas de *frameworks* para construir entornos de FL.

Finalmente, en este trabajo se ha optado por un tipo de estudio **cuantitativo**, debido al nivel de rigurosidad y objetividad en los resultados para la justa comparación del rendimiento de las estrategias de selección que estudiaremos.

4.2. Datasets utilizados

El estudio que propone este TFG evalúa el rendimiento de los modelos de Machine Learning en un entorno de Aprendizaje Federado. Estos modelos se ajustan, o “aprenden”, con un conjunto de datos de cierto tamaño y tipo de dato para evaluar su rendimiento en la predicción o clasificación de estos. En este TFG utilizaremos dos de los conjuntos de datos más usados, importantes y de gran escala en el campo de Visión por Computador que proponen un problema de clasificación lo suficientemente realista pero también viable en complejidad para llevar a cabo el estudio: MNIST y CIFAR-10.

El primero de ellos, MNIST [70] (*Modified National Institute of Standards and Technology*), consiste en una colección de imágenes de 28 píxeles de ancho y alto en escala de grises (valores unidimensionales en el rango [0, 255])



Figura 4.1: Ejemplos de dígitos manuscritos de MNIST. Imagen extraída de [69] via Wikimedia Commons.

de dígitos manuscritos del 0 al 9 (Véase Figura 4.1). El dataset de MNIST original referenciado en el trabajo de Yann LeCun sobre reconocimiento de dígitos manuscritos [70], puede ser descargado desde la página oficial en [71]; sin embargo, en para este estudio hemos optado por una versión extendida de MNIST, denominada *Extended MNIST* o simplemente EMNIST [72], desarrollado y mantenido por el mismo Instituto Nacional de Estándares y Tecnología (NIST). Este dataset comprende una versión extendida del concepto original del original MNIST para proporcionar dígitos manuscritos (MNIST) y letras manuscritas en el mismo formato (Imágenes 28×28 de ancho y alto en escala de grises). También ofrece una cantidad más grande de datos en comparación a MNIST y diferentes opciones de separación de los datos (solo dígitos, letras, por clase, ...). En nuestro caso utilizaremos la separación por dígitos, es decir, comprenderemos todos los dígitos y no las letras, simulando el dataset de MNIST. Esta versión contiene un total de $N_T = 280,000$ imágenes de dígitos manuscritos separados en imágenes de entrenamiento ($N = 240,000$) y de test ($N_{\text{test}} = 40,000$).

El segundo conjunto de datos que utilizaremos es CIFAR-10 [74], el cual es una colección de 60,000 imágenes de 32 píxeles de ancho a color (3 canales RGB con valores en el rango $[0, 255]$) agrupadas en 10 clases de objetos reales (Véase Figura 4.2). Este dataset representa objetos reales a una resolución muy baja (32×32) lo que lo hace un conjunto de datos muy adecuado entre los trabajos de investigación en Visión por Computador ya que permite a los investigadores probar distintos algoritmos para sacar conclusiones de forma rápida, lo que lo hace también apropiado para este trabajo. Concretamente, el dataset contiene 50,000 imágenes de entrenamiento contenidas en cinco *batches* de 5,000 imágenes por clase, mientras que para test tenemos 10,000 imágenes con un *batch* con 1,000 imágenes por clase. Además, las clase son “completamente excluyentes entre sí”, lo que quiere decir que no hay

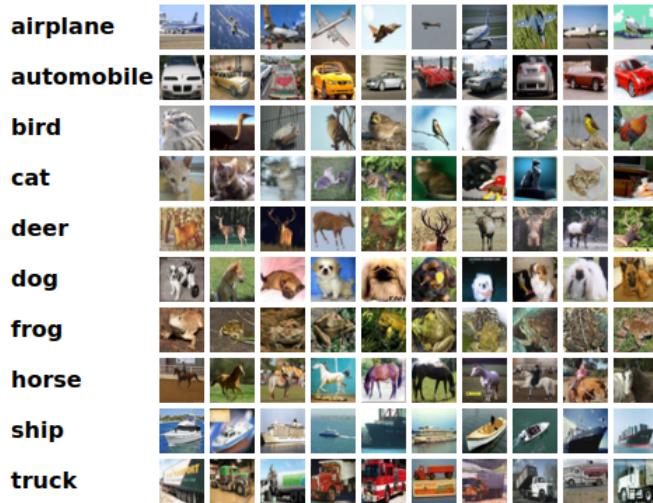


Figura 4.2: Ejemplos de CIFAR-10 agrupadas por las 10 clases que caracterizan el dataset. Imagen extraída de la página web del conjunto de datos [73].

solapamiento de las clases (e.g. un *automóvil* no será clasificado también como *camión* [74]). Podemos ver que CIFAR-10 es un dataset mucho más pequeño en tamaño de ejemplos que MNIST (casi 5 veces más grande), sin embargo, CIFAR-10 es mucho más complejo en dimensión por cada ejemplo, es decir, tenemos más información que procesar por imagen que en MNIST; esto es debido a que CIFAR-10 son imágenes con más resolución (32×32 frente a MNIST con 28×28) y a color (3 canales de color frente a un canal en MNIST). El uso de ambos para este trabajo nos permite diversificar los problemas a resolver por las distintas estrategias que veremos, siendo buena y usual práctica en investigación utilizar diversos conjuntos de datos con distintas características.

4.2.1. Formulación del problema de aprendizaje

Como hemos comentado en el capítulo 2, para cada dataset le corresponde un problema de aprendizaje que formaliza el escenario de Machine Learning. En el caso de MNIST, definimos formalmente su problema de aprendizaje como un problema de clasificación multiclase, donde definimos una imagen de entrada $X_n \in \mathcal{X}$ como una matriz $X_{28 \times 28}, x_i \in \{1, \dots, 255\}$, donde x_i es un pixel de la imagen X . Cada imagen de entrada $X_n \in \mathcal{X}$ es etiquetada con una clase $y_n \in \mathcal{Y}, y = \{0, 1, \dots, 9\}$ que representa el dígito al que corresponde la imagen. Para el caso de CIFAR-10, también será un problema de clasificación multiclase, donde definimos una imagen de entrada $X_n \in \mathcal{X}$ como una matriz $X_{32 \times 32 \times 3}, x_i \in \{1, \dots, 255\}$, donde x_i^c es un

pixel de la imagen X en el canal c . Cada imagen de entrada $X_n \in \mathcal{X}$ es etiquetada con una clase $y_n \in \mathcal{Y}, y = \{0, 1, \dots, 9\}$ tal que indexa la lista de clases $C = (c_0 = \text{plane}, c_1 = \text{automobile}, \dots, c_9 = \text{truck})$.

En cuanto a los otros componentes del problema de aprendizaje, en cuyo caso nos falta definir el modelo o de aprendizaje, lo veremos más adelante en la sección 4.3. Todos los otros componentes se aplican lo ya visto en el capítulo 2.

4.2.2. Preprocesamiento y tratamiento de los datos

Para el procesamiento y tratamiento de los datos utilizaremos una normalización de los valores de cada imagen tanto para el entrenamiento como para test. Este proceso también es llamado **normalización min-max** o *re-escalado* ya que simplemente realiza una normalización de todos los píxeles a valores en el rango $[0, 1]$ o $[-1, 1]$, utilizando la operación 4.1. Esta normalización se aplica tanto a MNIST como a CIFAR-10.

$$x' = \frac{x - \min(x)}{\max(x) - \min(x)} \quad (4.1)$$

Adicional a esto, normalizaremos los valores de cada píxel de las imágenes en base a la media y desviación típica para cada canal utilizando la operación 4.2 para un canal x determinado. En el caso de MNIST, esta normalización se realiza con $\mu = 0.5, \sigma = 0.5$ para el único canal que posee, elegimos 0.5 por una razón práctica; muchos investigadores usan esta media y desviación típica ya que los valores reales de MNIST están muy cerca de estos valores. En el caso de CIFAR-10, utilizamos los valores correctos de normalización y desviación típica [75]: $\mathbf{M} = (0.4914, 0.4822, 0.4465)$, $\mathbf{\Sigma} = (0.247, 0.243, 0.261)$.

$$C_{\text{norm}} = \frac{C_x - \mu_x}{\sigma_x} \quad (4.2)$$

El uso de la normalización de los datos así como su re-escalado es una práctica muy usada en los problemas de Machine Learning por permitir una rápida convergencia del optimizador GD (véase sección 2.1.4). Esto es debido a que cuando normalizamos restringimos el espacio de valores a un rango más cerrado y concreto, lo que a la vez disminuye el espacio de búsqueda del optimizador, como podemos ver en la Figura 4.3, en el que cuando los datos no están normalizados, el gradiente toma más pasos de optimización resultado de los grandes saltos entre los valores, mientras que cuando están normalizados y escalados, el gradiente toma pasos más uniformes. Esto permite al algoritmo de aprendizaje a ser más robusto ante *outliers* o valores atípicos, y en general a tener una rápida convergencia.

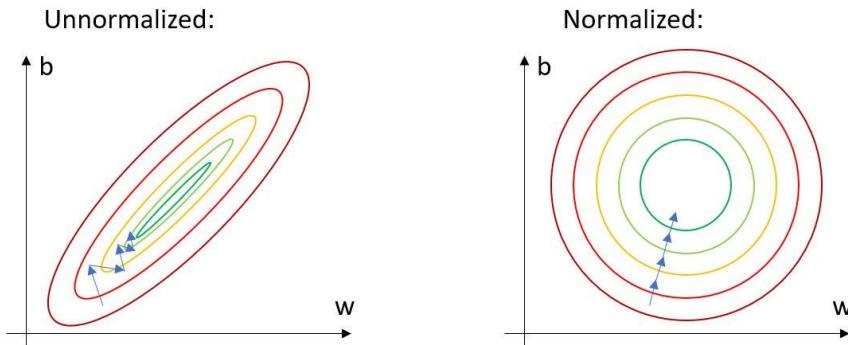


Figura 4.3: Traza del descenso del gradiente sobre un espacio de valores normalizados (derecha) y no normalizados (izquierda). Imagen extraída de [76].

4.3. Modelos de Aprendizaje

Como hemos comentado en la sección anterior, a cada problema de aprendizaje, le corresponde un modelo de aprendizaje \mathcal{A} que lo resuelva. Formalmente, se le denominan algoritmos de aprendizaje, y estos pueden ir desde Regresores Logísticos hasta Redes Neuronales. En nuestro caso, tenemos dos problemas de clasificación de Visión por Computador, es decir, problemas de clasificación de imágenes. Por tanto, necesitamos un algoritmo de aprendizaje que a partir de una sucesión de píxeles comprendidas en una matriz $X_n \in \mathcal{X}$ nos de una probabilidad de pertenecer a cada clase de \mathcal{Y} .

Lo que diferencia el campo de Visión por Computador de otros campos del Machine Learning, es el procesamiento de las imágenes para poder ser utilizada en un algoritmo de aprendizaje. Como hemos podido ver en la Figura 2.2, un algoritmo de aprendizaje le es proporcionado un vector $x_n \in \mathbb{R}^d$ **unidimensional** como representación o *embedding* de un ejemplo n ; es lo que comúnmente se denomina un vector de características o *features*. Sin embargo, cuando tratamos con imágenes, las características de cada entrada son los vectores **multidimensionales** de cada imagen. En el caso de una imagen perteneciente al dataset MNIST tendríamos que cada entrada es de la forma $\mathcal{X} = \mathbb{R}^d \times \mathbb{R}^d$, donde d es el ancho y alto de cada imagen suponiendo imágenes cuadráticas (mismo ancho y alto). Por tanto, en Visión por Computador es importante los pasos de tratamiento de las imágenes para poder tener una representación adecuada para ser pasada a un algoritmo de aprendizaje.

Una forma simple de representación de imágenes es simplemente creando un vector unidimensional a partir de la/s matriz/ces de la imagen, en un

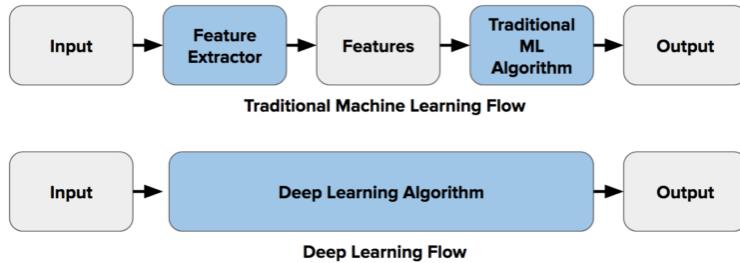


Figura 4.4: El flujo de aprendizaje y clasificación del método tradicional o clásico de clasificación de imágenes y clasificación de imágenes con Deep Learning. En el método clásico, las características son creadas a mano, normalmente por agentes expertos o utilizando descriptores como SIFT y HOG. En Deep Learning, estas características son *aprendidas* por la misma red neuronal. Imagen extraída de [80].

proceso conocido como *flatten*², para luego ser pasada a un clasificador. Esta solución funciona bien para imágenes de muy baja resolución (como por ejemplo las imágenes de MNIST), sin embargo, cuando tenemos imágenes de mayor complejidad dimensional, usar esta aproximación puede no ser eficiente. En Visión por Computador existen diversas formas para poder solucionar el problema de representación de las imágenes de forma más inteligente reduciendo la dimensionalidad del vector que vamos a pasar al clasificador, en un proceso fundamental, sobre todo en métodos clásicos (i.e. no relacionados al Deep Learning), denominado *Feature Extraction* o Extracción de Características. Muchos algoritmos de extracción de características han mostrado grandes avances en clasificación de imágenes con Machine Learning como: HOG [77], SIFT [78] y las características de Haralick usando GLMC [79]. Estos algoritmos proporcionan *descriptores* que son esencialmente un vector de valores reales que representan a una imagen, y por tanto un vector de características.

Aunque esta aproximación del paso de extracción de características funciona bien en muchos problemas de visión por computador, esto requería al fin y al cabo tener que extraer las características a mano o con descriptores como los comentados anteriormente que implican la configuración parametrizada de estos para obtener buenas representaciones (Véase Figura 4.4). Como hemos visto en el capítulo 2, los trabajos de Yann LeCun [32] con las redes convolucionales para clasificación de imágenes han mostrado superar a estos métodos “clásicos” de visión por computador, en el que las características son *aprendidas* por la misma red neuronal mediante filtros o *kernels* que crean *embeddings* de estas imágenes para luego ser pasadas a un clasificador como un Perceptrón Multicapa; estas redes con las llamadas

²Que viene de la traducción al inglés de “aplanar”.

Redes Neuronales Convolucionales que hemos podido ver en el capítulo 2 y pertenecen al campo del Aprendizaje Profundo o *Deep Learning*. Esto nos da una aproximación *end-to-end* de clasificar imágenes sin el paso intermedio de extracción de características (Véase Figura 4.4). Dicho esto, en este trabajo utilizaremos dos arquitecturas de red neuronal: para MNIST utilizaremos un Perceptrón Multicapa debido a que las imágenes son de muy baja dimensionalidad, y por tanto es viable poder usarlas sobre un clasificador universal como lo es una MLP; mientras que para CIFAR-10 utilizaremos una Red Neuronal Convolutional o CNN, ya que debido a la alta dimensionalidad con respecto a MNIST, es más apropiado utilizar una CNN para reducir la cantidad de parámetros que tiene que aprender la red.

En el caso de la MLP para MNIST, será una red neuronal de dos capas ocultas, es decir, una capa oculta intermedia y la última de salida. Para la primera capa oculta ocuparemos 128 unidades y 10 la última (correspondiente al número de clases). Usaremos además una función de activación ReLU [81] como función de activación en la primera capa oculta y SoftMax [82] para la de salida. En el Listing 4.1 se muestra un resumen de la arquitectura de la MLP para MNIST.

Listing 4.1: Arquitectura de la MLP para MNIST. Salida obtenida con la librería `torchsummary` [83].

Layer (type)	Output Shape	Param #
<hr/>		
Flatten-1	[-1, 784]	0
Linear-2	[-1, 128]	100,480
Linear-3	[-1, 10]	1,290
<hr/>		
Total params:	101,770	
Trainable params:	101,770	
Non-trainable params:	0	
<hr/>		
Input size (MB):	0.00	
Forward/backward pass size (MB):	0.01	
Params size (MB):	0.39	
Estimated Total Size (MB):	0.40	
<hr/>		

En cuanto a la CNN que ajusta CIFAR-10, utilizaremos dos arquitecturas. La primera red esta basada en la utilizada por los autores de GreedyFed en sus experimentos [60]; esta consiste en una CNN estándar que comprende dos capas convolucionales (*CONV Layers*) 4×4 con las siguientes características:

- Ambas tendrán 8 canales de salida.
- Cada uno de ellos activados por la función de activación ReLU [81].

- Al final de cada capa CONV, aplicamos una capa de MaxPooling 2×2 con un *stride* de 2. Esto nos permitirá reducir la dimensionalidad obtenida de los mapas de activación e inducir invarianza en la traslación, rotación y escala [84].

Será seguida de una capa Fully-Connected de salida (i.e. un clasificador MLP) de 10 unidades activada por SoftMax.

Listing 4.2: Arquitectura de SimpleCNN para CIFAR-10 tomada de los experimentos de GreedyFed [60]. Salida obtenida con la librería `torchsummary` [83].

1	-----	Layer (type)	Output Shape	Param #
2	=====			
3		Conv2d-1	[-1, 8, 31, 31]	392
4		MaxPool2d-2	[-1, 8, 15, 15]	0
5		Conv2d-3	[-1, 8, 14, 14]	1,032
6		MaxPool2d-4	[-1, 8, 7, 7]	0
7		Linear-5	[-1, 10]	3,930
8	=====			
9		Total params:	5,354	
10		Trainable params:	5,354	
11		Non-trainable params:	0	
12	-----			
13		Input size (MB):	0.01	
14		Forward/backward pass size (MB):	0.09	
15		Params size (MB):	0.02	
16		Estimated Total Size (MB):	0.12	
17	-----			
18				

Elegimos esta red neuronal para poder utilizarlas sobre los métodos de selección de clientes basados en el rendimiento, ya que es una red neuronal sencilla y con pocos parámetros que permite una rápida clasificación y aprendizaje de CIFAR-10 en muy poco tiempo. Esto nos era especialmente útil, y hasta necesario, en el caso GreedyFed debido a que es un algoritmo que requiere un tiempo de cómputo muy elevado, por lo que esta red reduciría gran parte del proceso en tiempo. El resumen de esta CNN se muestra en el Listing 4.2 e identificaremos esta CNN con el nombre de **SimpleCNN** para diferenciarla de la otra CNN.

Listing 4.3: Arquitectura de ComplexCNN para CIFAR-10 tomada de los experimentos de HybridFL [58]. Salida obtenida con la librería `torchsummary` [83].

1	-----	Layer (type)	Output Shape	Param #
2	=====			
3		Conv2d-1	[-1, 32, 32, 32]	896
4		BatchNorm2d-2	[-1, 32, 32, 32]	64
5		Conv2d-3	[-1, 32, 32, 32]	9,248
6		BatchNorm2d-4	[-1, 32, 32, 32]	64
7	-----			

```

8      MaxPool2d-5           [-1, 32, 16, 16]          0
9      Conv2d-6              [-1, 64, 16, 16]         18,496
10     BatchNorm2d-7         [-1, 64, 16, 16]          128
11     Conv2d-8              [-1, 64, 16, 16]         36,928
12     BatchNorm2d-9         [-1, 64, 16, 16]          128
13     MaxPool2d-10          [-1, 64, 8, 8]            0
14     Conv2d-11             [-1, 128, 8, 8]        73,856
15     BatchNorm2d-12         [-1, 128, 8, 8]          256
16     Conv2d-13             [-1, 128, 8, 8]        147,584
17     BatchNorm2d-14         [-1, 128, 8, 8]          256
18     MaxPool2d-15          [-1, 128, 4, 4]            0
19     Flatten-16             [-1, 2048]                0
20     Linear-17             [-1, 512]                 1,049,088
21     Linear-18             [-1, 10]                  5,130
22 =====
23 Total params: 1,342,122
24 Trainable params: 1,342,122
25 Non-trainable params: 0
26 -----
27 Input size (MB): 0.01
28 Forward/backward pass size (MB): 1.88
29 Params size (MB): 5.12
30 Estimated Total Size (MB): 7.01
31 -----

```

La otra CNN estará será tomada de la red usada en los experimentos de HybridFL [58], la cual a diferencia de la SimpleCNN, esta es mucho más compleja en número de parámetros. Concretamente, esta red consistirá en “*seis capas de convolución de 3×3 (con 32, 32, 64, 64, 128 y 128 canales, cada una de las cuales se activaba utilizando ReLU y se normalizaba por lotes, y cada dos de las cuales estaban seguidas de un MaxPooling de 2×2), seguidas de tres capas completamente conectadas (con 512 y 192 unidades activadas utilizando ReLU y otras 10 unidades activadas utilizando soft-max)*”. Este red la utilizaremos sobre los algoritmos de selección basados en los recursos de los clientes y la decisión de usar una red más compleja viene a partir de los requerimientos de estos algoritmos de tener un modelo con mayor tamaño en bytes; esto para poder “simular” modelos usados en situaciones reales que son en la práctica más profundos que SimpleCNN y que esto pueda tener un impacto en los resultados que veremos más adelante. Información adicional y detalles de la arquitectura se muestran en el Listing 4.3 y la identificaremos con el nombre **ComplexCNN**. Nótese que en el tamaño estimado de los parámetros de esta red en 4.3 es mayor que SimpleCNN (~ 26 veces mayor en tamaño), esto jugara un papel importante a la hora de estimar el tiempo que toman los clientes en actualizar sus modelos al servidor, así como la distribución de este último a los clientes y utilizar una red más *pesada* nos permitirá reflejar el impacto en la subida de los parámetros del servidor a los clientes y viceversa; veremos más detalles sobre esto más adelante en la sección 4.5 se este capítulo.

Nótese que en el campo de la Visión por Computador y en especial en el Deep Learning aplicado en este campo, se usan redes más profundas que

consiguen resultados del estado del arte como las redes residuales, también llamadas *ResNet* [85], y que pueden proveer un mejor rendimiento en la clasificación de CIFAR-10. Sin embargo, y como bien establece Nishio et al. ([8]) sobre la selección de clientes en FL, el estudio de estos modelos no son el foco de nuestro estudio.

4.3.1. Función de pérdida

En el capítulo 2 hemos visto que los algoritmos de aprendizaje necesitan una medida para evaluar el error o “pérdida” que obtiene el modelo cuando predice sobre una serie de ejemplos. Necesitamos por tanto definir una función, conocida como función de error o función de pérdida o en algunos casos *criterion*, que usen nuestros modelos de aprendizaje. Como se trata de una problema de clasificación para los problemas que estamos utilizando, elegiremos la función de pérdida la función de entropía cruzada o *Cross-Entropy* definida por 4.3 para un ejemplo n . La *Cross Entropy* es una función de pérdida muy utilizada en Machine Learning para problemas de clasificación multiclas [86] y viene del campo de la Teoría de la Información donde mide la cantidad de información que se necesita para identificar un evento desde una distribución de probabilidad a otra. Aplicado al Machine Learning, esta es usada para medir la “discrepancia” entre la clase verdadera (*ground truth*) y la clase predicha por el modelo. Esta función es también denominada indistintamente con el nombre de *LogLoss* o Pérdida Logarítmica.

$$l_n = -w_{y_n} \cdot \log \frac{\exp(x_n, y_n)}{\sum_{c=0}^C \exp(x_n, c)} = -w_{y_n} \cdot \log(\text{Softmax}(x_n, y_n)) \quad (4.3)$$

4.3.2. Optimizador *Adam*

Como hemos discutido en el capítulo 2, para poder realizar un aprendizaje sobre los datos, necesitamos un método de optimización de los pesos o parámetros del modelo, siendo el más utilizado el de Descenso del Gradiente o GD. Existen varias propuestas de métodos de optimización basados en GD, siendo uno de los más influyentes el *Stochastic Gradient Descent* [87], que elige de forma aleatoria un paquete de datos llamados “mini-batches” para poder realizar el paso de optimización, esto se realiza de forma iterativa hasta completar una época (procesando todos los mini-batches). Nosotros utilizaremos una variante de SGD llamada *Adam* [26], que suele tener una convergencia mucho más rápida que otros métodos debido a que adapta para cada parámetro un peso o ponderación; de manera que pueda dirigir al gradiente de forma más dinámica y adaptativa asignando un mayor peso a

aquellos parámetros que más afecten a la pérdida del modelo [88]. Adam, a pesar de no converger en todas las funciones convexas [89], sigue siendo uno de los optimizadores más utilizados por su robustez y convergencia a la tasa de aprendizaje y valores iniciales [90].

4.4. Herramientas utilizadas

Uno de los objetivos primordiales de este TFG es la **implementación de los algoritmos de selección y su ejecución**. La implementación y desarrollo de estos métodos, la preparación del entorno, desafíos técnicos encontrados y su solución, son factores importantes en el desarrollo de este trabajo y del estudio que queremos realizar, especialmente la integración de los métodos de selección de clientes en un entorno de FL. Para ello se ha utilizado una *suite* de herramientas software que nos permitirán crear este entorno experimental y llevar a cabo este TFG. Desplegamos así las herramientas utilizadas a continuación:

- **Python.** Python [91] es un lenguaje de programación de propósito general ampliamente utilizado en diferentes campos y sectores. Es especialmente utilizado en lo que concierne al campo de la Inteligencia Artificial, Machine Learning y a la ciencia de datos por su alto nivel de abstracción y su forma más declarativa de programar. Además, Python incluye una amplia variedad de librerías para diferentes propósitos de código abierto, muchas sino la mayoría de las cuales vamos a utilizar como herramientas de este trabajo.
- **Jupyter Notebooks.** En este trabajo utilizamos Python por medio de un entorno interactivo conocido como *Jupyter Notebooks* [16] o “cuadernos” de Python o simplemente *notebooks*. Estos notebooks crean un entorno de programación cómodo e ideal para poder ser usado en investigación, utilizando celdas de código que ejecutan código de Python y celdas de texto para poder redactar en Markdown [92]. Esto permite organizar el código que implementamos de una forma más experimental y dinámica.
- Herramientas para la ciencia de datos. Entre estas herramientas podemos incluir **Pandas** [93] para análisis de datos, **Numpy** [94] para funciones de álgebra lineal y **Matplotlib** [95] para visualización de datos.
- **PyTorch.** Para poder crear y entrenar modelos de Machine Learning, específicamente Redes Neuronales, utilizamos la librería de PyTorch [96] que nos permite de una forma muy flexible y modular crear redes neuronales y entrenarlas. Este librería también nos proporciona de

una gran variedad de utilidades que necesitaremos para poder crear y entrenar estos modelos (funciones de activación, capas convolucionales, funciones de pérdida, operabilidad con GPU, ...).

- **FLEXible Framework.** Finalmente, para poder crear un entorno de investigación y de pruebas con Aprendizaje Federado utilizamos el framework de *FLEXible* [67] o simplemente FLEX. Este framework nos permitirá crear una configuración y escenario de Aprendizaje Federado de una forma flexible y modular que nos será muy útil para poder integrar los algoritmos de selección de clientes. Saber como se compone FLEXible y su uso es importante para poder seguir el proceso de implementación de los algoritmos en el capítulo 5, es por eso que explicaremos con un poco más de detalle sus componentes y su uso en la sección 4.4.1 de este capítulo.

4.4.1. FLEXible Framework

Para poder entender los detalles de implementación de las estrategias de selección de clientes explicados en el capítulo 5, debemos primero de entender el funcionamiento y el uso del framework de aprendizaje federado que vamos a utilizar para montar nuestro entorno. En este caso utilizamos el framework FLEXible (FLEX) [67] creado por el DasCI [68] de código abierto. FLEX es un framework o librería de Python que permite la creación y entrenamiento de modelos de ML en un entorno de Aprendizaje Federado que pretende ser lo más *flexible*. Por tanto, FLEX va dirigido a trabajos o casos de uso relacionados con la investigación y experimentación con Aprendizaje Federado, ya que pretende ser un marco de trabajo que se adapta a las necesidades de cada caso de uso.

FLEX está diseñado sobre tres módulos principales mostradas en la Figura 4.5: El módulo *Data, Actor y Pool*.

El módulo de datos o *Data Module* es el encargado de gestionar la creación o descarga de los conjuntos de datos federados. La federación de los datos así como su distribución entre los diferentes nodos (i.e. clientes) es importante en la configuración experimental con Aprendizaje Federado. Este módulo gestiona así tanto la descarga de un conjunto de datasets predefinido (e.g. FederatedMNIST, Shakespeare, ...) como la distribución de los datos entre los clientes, un proceso al que se le denomina *federación* de un conjunto de datos.

La distribución de los datos la realiza siguiendo una configuración determinada definida en un objeto *FedDatasetConfig*. En tal configuración, el usuario puede especificar la federación de un conjunto de datos en general, estableciendo por ejemplo el número de nodos a utilizar, si los clientes

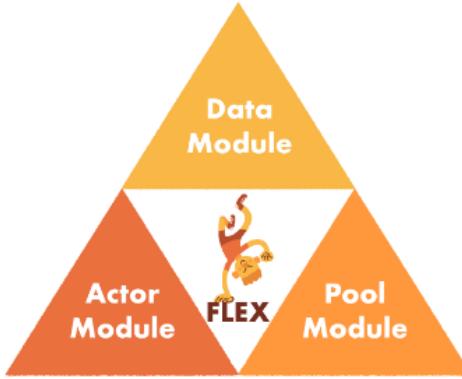


Figura 4.5: Diseño modular de FLEXible. Imagen extraída de [67].

comparten datos, etc. En el Listing 4.4 se muestra como federar una dataset utilizando FLEX.

Listing 4.4: Federación del dataset CIFAR-10 con FLEX. Código extraído de [67].

```

1 train_dataset = flex.data.Dataset.
2     from_torchvision_dataset(cifar10)
3 # Fix a seed to make our split reproducible
4 config = flex.data.FedDatasetConfig(seed = 0)
5 # it is not clear whether clients share their data or not
6 config.replacement = True
7 # 10 nodes, for greater number of nodes, change this
8 # value. As we do not provide a
9 # custom id for each node, integers starting from 0 are
10 # used as id
11 config.n_nodes = 10
12 # Assign a sample proportion for each node-class pair
13 num_classes = 10
14 alphas = numpy.random.uniform(0.4, 0.6, [config.n_nodes,
15     num_classes])
16 alphas = alphas / numpy.sum(alphas, axis=0)
17 config.weights_per_label = alphas
18 # Perform the actual data split
19 federated_dataset = flex.data.FedDataDistribution.
20     from_config(train_dataset, config)

```

En cuanto al módulo de actores o *Actor Module*, es el encargado del mecanismo de comunicación entre las entidades o *actores* en un escenario de

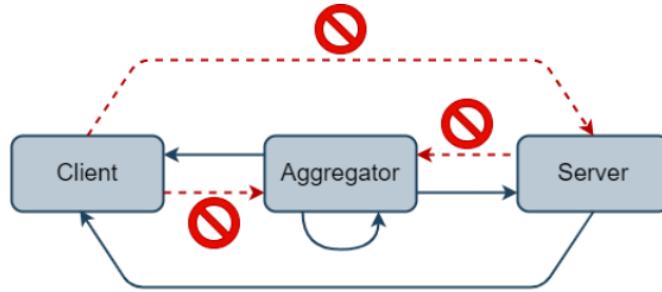


Figura 4.6: Esquema de los permisos asignados a cada rol de FLEX. Imagen extraída de [67].

Aprendizaje Federado. Para ello, cada entidad se le es asignado un **rol** que determina con que otras entidades se puede comunicar. Existen tres roles:

- *Client role*: este rol es el más restrictivo de todos ya que solo permite la comunicación con el mismo, i.e. no puede comunicarse con otros nodos.
- *Aggregator role*: este rol permite la comunicación con otros actores tanto clientes como servidores o agregadores.
- *Server role*: este rol solo permite la comunicación con otros servidores o con los clientes.

En la Figura 4.6 se muestra un resumen visual de las comunicaciones que permiten cada uno de los roles. En nuestro caso y el más común, es usar una arquitectura *cliente-servidor*, en donde todos los clientes tienen el rol de *Client* y el servidor tiene tanto los roles de *Server* como de *Aggregator*. En este trabajo se usará indistintamente los nombres de “servidor” y “agregador” para referirse al único servidor del escenario de Aprendizaje Federado.

Finalmente el módulo de *pooling* o *Pool module*, es el encargado de gestionar el flujo de trabajo en Aprendizaje Federado. Para ello, se utiliza un objeto **FlexPool** que representa una *piscina* o espacio de actores, datos y modelos indexados mediante un identificador que podemos usar para poder ejecutar el entrenamiento con Aprendizaje Federado. Las funciones más importantes y que usaremos de manera más habitual con este objeto *pool* son la función de **select** y la de **map**.

Listing 4.5: Selección de actores utilizando la función **select**. Código extraído de [67].

```
1 | # Pool with two random actors
```

```

2 subpool = pool.select(2)
3 # Pool with actors with server role
4 servers = pool.servers
5 # alternatively
6 servers = pool.select(lambda actor_id, set_of_roles:
7                         FlexRoleManager.is_server(
8                             set_of_roles))
9 # Pool with actors with ids 0 and 2
9 selected_clients = pool.select(lambda actor_id,
10                                 set_of_roles: actor_id in ["0", "2"])

```

La función `select` se utiliza para seleccionar un subconjunto de actores de un *pool*. En el Listing 4.5 se muestra como podemos seleccionar un subconjunto de servidores bajo varios criterios. Esto nos será útil para seleccionar el subconjunto de clientes seleccionados para entrenar en una ronda concreta (i.e. realizar la selección de clientes).

Listing 4.6: Distribución del modelo global utilizando la función `map`.

```

1 def distribute_model(server_model: FlexModel,
2                      client_flex_models: List[FlexModel]):
3     for k in client_flex_models:
4         # Make a Deep Copy of global model to a client's
5         # model
5         client_flex_models[k]["model"] = deepcopy(
6             server_model["model"])
6
7 pool.servers.map(distribute_model, pool.clients)

```

La función `map` es la encargada de realizar la comunicación entre actores. Su funcionamiento consiste en que una *pool* determinada llama a la función pasándole una función como argumento, esta función recibe a su vez como argumentos los modelos de FLEX (i.e. las estructuras de cada actor donde almacenan información) del que llama y los modelos a los que envía el mensaje. En el Listing 4.6 se muestra un ejemplo del caso en el que un servidor distribuye su modelo global a todos los clientes. FLEX además de la función `map`, posee una serie de *decoradores* que simplifican el proceso de Aprendizaje Federado para las diferentes fases del protocolo en cuestión, e.g. la distribución del modelo, la recolección de las actualizaciones del modelo, la evaluación en test del modelo global, etc. Los veremos en uso en la implementación del entorno de aprendizaje federado en el capítulo 5.

Un concepto importante que hemos mencionado antes son los modelos de FLEX (*FlexModel*), que son esencialmente una estructura de diccionario que almacena información en forma de *clave-valor* para un actor concreto. Un ejemplo de esto es que los actores tengan un modelo de ML almacenado, para el cual simplemente se asigna a la clave "model" el modelo en cuestión.

Esto nos será útil cuando queramos almacenar información de los actores (e.g. valuaciones de los clientes) en diferentes fases del entrenamiento.

4.5. Estrategias de selección de clientes

En esta sección vamos a ver en detalle las estrategias de selección de clientes que hemos elegido para su estudio, análisis y comparación de rendimiento. Como hemos comentado en anteriores capítulos y secciones los algoritmos que vamos a estudiar están divididos en el enfoque que tratan de resolver, concretamente los enfocados en el rendimiento del modelo: Active Federated Learning [57] y GreedyFed [60]; y los enfocados en los recursos de los clientes: HybridFL [58] y Dyn-HybridFL (propuesta propia). Vamos a ver los detalles de diseño y su flujo de trabajo en un entorno de aprendizaje federado así como sus características y objetivos que tratan de alcanzar para poder tener una mejor idea e hipótesis de cómo y en qué situaciones se comportan. Esta sección está dedicada solamente a ver con detalle los fundamentos y el diseño de cada algoritmo que estudiaremos, de manera que el lector pueda no solo entender el análisis de los resultados obtenidos a partir de estos métodos, sino de poder sacar sus propias conclusiones a partir de los tales. Esta sección *no* intenta realizar un análisis profundo de cada algoritmo, sino de proporcionar una vista general necesaria para la mejor comprensión de los experimentos realizados en capítulos posteriores.

4.5.1. Active Federated Learning

Active Federated Learning (AFL)[57], se define como un *esquema de muestreo* de clientes, simple e intuitivo cuyo objetivo es incrementar la eficiencia en el entrenamiento para, de esta manera, poder conseguir una mejor convergencia en un menor tiempo; como hemos visto en el capítulo 2, una manera de conseguir esto (y que es la manera que lo consigue AFL) es **reduciendo el número de rondas necesarias para conseguir un modelo con un buen rendimiento**. Así como muchos otros métodos basados en mejorar la convergencia del modelo, se basa en el hecho de que los datos de cada cliente son altamente variables entre sí y que por tanto, la selección del grupo de clientes en cada iteración influye en gran medida al rendimiento del modelo, por lo que este algoritmo explota esta característica de datos no-IID para elegir los clientes que mejor beneficien el modelo en la iteración actual.

AFL sigue los principios de *Active Learning*[97], un paradigma de aprendizaje automático en el que se seleccionan los datos bajo una política que depende del estado actual del modelo, dando el efecto de un aprendizaje dinámico y activo. En el caso de AFL, no seleccionamos datos sino que

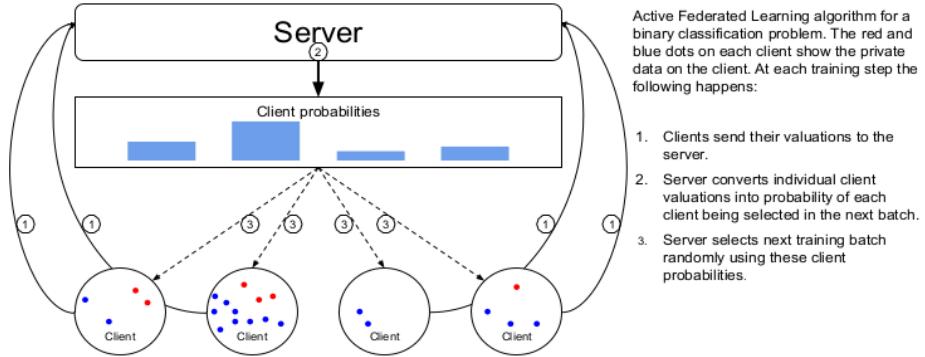


Figura 4.7: Esquema visual del flujo de trabajo de AFL en un problema de clasificación binaria. Imagen extraída del paper original [57].

seleccionamos un subgrupo de clientes basándose en sus datos locales. Concretamente, cada cliente es dado una **valuación** que mide la calidad de ese cliente (i.e. la calidad de sus datos) para ser seleccionado por el servidor central. Notase que el criterio por el cual un cliente es seleccionado utilizando este esquema depende enteramente de sus datos; esto es una característica que lo diferencia de los esquemas basados en recursos como FedCS [8], en el que toma en cuenta información más allá de sus datos.

Un esquema visual del proceso de AFL para la selección de los clientes es la que podemos ver en la Figura 4.7. En este diagrama de flujo, los clientes envían sus valuaciones al servidor (paso 1), el servidor luego convierte estas valuaciones en una **distribución de probabilidad** (paso 2) del cual muestreará los clientes que serán elegidos para participar en la siguiente ronda de entrenamiento (paso 3). Por tanto, el flujo de entrenamiento es adaptativo en cuanto a que los clientes siempre se van evaluando para que el servidor siempre elija los mejores clientes participantes de la siguiente ronda. Podemos ver entonces dos factores importantes a definir: la función de valuación y la distribución de probabilidad.

Función de valuación y política de selección

El objetivo de AFL es conseguir un subconjunto S_t para cada iteración t , que minimice el número necesario de iteraciones para conseguir un buen modelo final. Para conseguirlo, se determina un *valor* o valuación, para cada cliente que refleja la **utilidad** de cada uno para el entrenamiento del modelo en la ronda actual (en este caso la utilidad de sus datos locales). Formalmente, se define una función $\mathcal{V} : \mathcal{X}^{n_k} \times \mathcal{Y}^{n_k} \times \mathbb{R}^d \rightarrow \mathbb{R}$ que se usa para obtener las valuaciones de cada cliente $v_k \in \mathbb{R}; k \in S_t$, y que se enviarán al servidor para su interpretación y selección. La comunicación de

estas valuaciones puede ser costosa³, es por eso que AFL se diseña para que solo se volvieran a evaluar aquellos clientes que han sido seleccionados en la iteración anterior, formalmente:

$$v_k^{(t+1)} = \begin{cases} \mathcal{V}(\mathbf{x}_k, \mathbf{y}_k; \mathbf{w}^{(t)}) & , \text{ si } U_k \in S_t \\ v_k^{(t)} & , \text{ en otro caso} \end{cases}$$

No obstante, el no refrescar las valuaciones de los clientes no seleccionados no debería tener consecuencias notables, debido a que justamente los no seleccionados son los que tendrán **menos discrepancia** con el modelo actual (menor error de pérdida), lo que significa que sus datos no serán ajustados en el modelo de la iteración actual, por lo que reutilizar sus valuaciones en una posterior iteración tendría cambios imperceptibles si lo hicieramos refrescando sus valuaciones, por tanto, sus valuaciones seguirán proporcionando una buena representación de sus utilidades.

En cuanto a la función de valuación \mathcal{V} que usaremos, los autores proponen utilizar una representación de la pérdida en el entrenamiento de cada cliente con respecto al número de datos que poseen estos; ya que “es una aproximación natural” a una valuación para los clientes. En este caso, después de ajustar cada cliente con sus datos, estos envían su valor de pérdida como su valor devaluación $v_k = \frac{1}{\sqrt{n_k}} l(\mathbf{x}_k, \mathbf{y}_k; \mathbf{w}^{(t)})$; este valor crece cuanto *peor* rinda el modelo, por lo que se busca maximizar v_k para la selección. Esto tiene sentido ya que si se detecta que el modelo no ajusta bien sobre los datos de un cliente (tiene un error muy elevado) lo ideal y más sensato sería seleccionar ese cliente para ajustar el modelo con esos datos, mejorando la generalización y por tanto la convergencia del modelo. Esta función también busca seleccionar aquellos clientes con menor cantidad de datos (dividiendo entre $\sqrt{n_k}$), esto para sanar el problema del desbalanceo de clases, algo muy común en escenarios de FL puesto que en una distribución No-IID habrán clientes con una mayoría de datos minoritarios en sus datasets locales, por lo que su selección tiene que ser prioritaria; esto último combinado con que estos clientes con clases minoritarias tendrán una mayor pérdida con el modelo global, resultará en que estos clientes serán *favoritos* a la hora de la selección de la siguiente ronda.

Distribución de probabilidad

En este caso, la elección de una distribución de probabilidad es bastante directa si queremos seleccionar aquellos clientes con mayor valuación v_k , utilizando por ejemplo una distribución exponencial e^{v_k} (véase Figura 4.8). De

³De hecho, los recursos en la comunicación son el principal cuello de botella en el paradigma de aprendizaje FL.

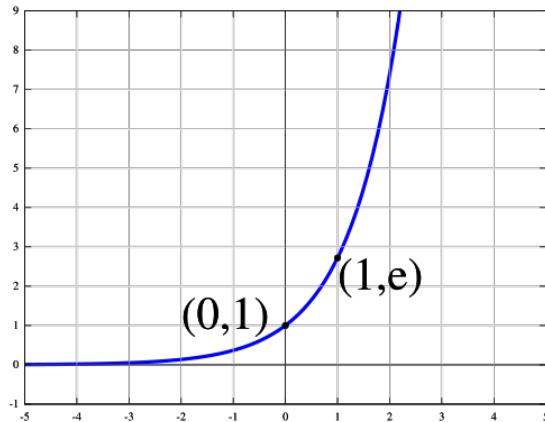


Figura 4.8: Función exponencial. Los autores de AFL [57] utilizan esta función para crear la distribución de probabilidad para muestrear los clientes seleccionados para la siguiente ronda en base a sus valuaciones v_k , de manera que los que tengan mayor v_k tengan más probabilidades de salir seleccionados. Imagen extraída de [98] via Wikimedia Commons.

manera que los clientes que tengan mayor v_k tengan una mayor probabilidad de ser muestreados (seleccionados).

Algoritmo de selección de clientes

El algoritmo propuesto por los autores que implementa esta técnica de selección de clientes es el mostrado en el Algoritmo 3. En este algoritmo se definen tres parámetros de ajuste $\alpha_1, \alpha_2, \alpha_3$. El primero α_1 , *descarta* los $\alpha_1 \cdot K$ clientes con menor v_k para luego ser sometidos a selección aleatoria, en otras palabras, este parámetro configura que proporción de los clientes se van a considerar para selección en base a sus valuaciones. El segundo, α_2 , se define como una *temperatura* de *softmax*, la cual suaviza la distribución de probabilidad de *sampling*; los autores eligen un valor de α_2 muy pequeño de 0.01 para asegurar que no hayan clientes con $p_k = 0$ por errores de desbordamiento. El tercero, α_3 , indica la proporción de clientes que serán seleccionados aleatoriamente. Nótese que los clientes descartados para ser seleccionados en base a v_k (i.e. los que se les asigna un valor $v_k = -\infty$) **pueden** ser seleccionados aleatoriamente, más no asegura que solo estos puedan ser seleccionados de esta manera; los clientes no *descartados* pero no seleccionados por v_k también pueden ser aleatoriamente elegidos. Esto se diseña así para inducir aleatoriedad en la selección de los modelos y tener un factor de *exploración* que contrarreste la explotación inducida por los clientes favoritos seleccionados y evitar inaniciones de los clientes menos valuosados.

Algoritmo 3 Algoritmo de muestreo de AFL para seleccionar clientes. Extraído parcialmente del trabajo original [57].

Input: Valuaciones de los clientes $\{v_1, \dots, v_K\}$, parámetros $\alpha_1, \dots, \alpha_3$, número de clientes por ronda M .

Output: Índices de los clientes seleccionados $\{k_1, \dots, k_M\}$

- 1: Ordenar clientes por v_k
 - 2: Para los $\alpha_1 \cdot K$ clientes con menor v_k , asignar $v_k = -\infty$
 - 3: **for** k from 1 to K **do**
 - 4: $p_k \propto e^{\alpha_2 v_k}$ ▷ Asignar probabilidades a partir de una distribución exponencial
 - 5: **end for**
 - 6: Muestrear $(1 - \alpha_3)M$ clientes en base a sus probabilidades p_k , dando un conjunto S' de clientes
 - 7: Muestrear los $\alpha_3 M$ clientes restantes de forma aleatoria y uniforme, dando un conjunto S'' de clientes
 - 8: **return** $S = S' \cup S''$
-

4.5.2. Greedy Shapley Client Selection

El siguiente algoritmo enfocado en el rendimiento del modelo es Greedy Shapley o como lo denominan sus autores, *GreedyFed* [60]. GreedyFed pertenece a una familia de métodos de selección de clientes basadas en la valuación de estos mediante los valores de Shapley o SV [52]. Esta técnica selecciona entonces a los clientes con un enfoque voraz o *greedy* y **sesgado** hacia los clientes que contribuyan al aprendizaje del modelo.

Esta técnica enfrenta a la mayoría de protocolos de selección de clientes que se hayan en la literatura en cuanto a que intentan subsanar el **sesgo producido por la heterogeneidad de los clientes** así como la inducción de selección aleatoria uniforme, que proporciona una selección más justa sobre los clientes menos contribuyentes, como justamente es el caso de AFL (véase sección 4.5.1); ambas formas de luchar contra la naturaleza de los escenarios de FL han sido probados de ser **subóptimos y con una lenta convergencia**, especialmente en escenarios con un *alto* índice de datos no-IID. GreedyFed propone un algoritmo de selección que busca la convergencia rápida en entrenamiento para así solucionar el problema de la sobrecarga en la comunicación en FL, basándose en la idea de que en la mayoría de escenarios de FL, hay un alto índice de datos no-IID, y la selección voraz de los clientes que mejor contribuyan al modelo mejoraría la convergencia de este. Así, GreedyFed, en contraste con AFL, propone una selección más directa hacia los clientes con mejores valores de Shapley para obtener un rendimiento *aceptable* del modelo en un menor tiempo de rondas.

Valuación de los clientes basada en valores Shapley (SV)

Una parte importante de esta familia de técnicas de selección de clientes es su función de valuación basada en valores SV. Los valores de Shapley son un concepto traído de la teoría de juegos, donde tratamos un escenario de Aprendizaje Federado como un juego cooperativo entre los clientes para intentar obtener el mejor modelo global. De esta manera, en cada ronda se recompensa cada cliente seleccionado con un valor de Shapley que cuantifica la **contribución marginal** de cada cliente al modelo agregado; el cálculo de estos SV pretende dar una representación justa de la contribución de todos los clientes. Con esto, el cliente puede seleccionar en cada ronda los clientes que contribuyan más al aprendizaje del modelo en base a los que tengan valores de Shapley mayores.

Entre los distintos algoritmos basados en la generación de estos valores de Shapley y su interpretación se encuentran los trabajos de UCB [47] y S-FedAvg [59]. El primero de ellos, UCB o *Upper Confidence Bound*, plantea el uso los valores de Shapley medios para poder obtener un límite superior de confianza en que el cliente en cuestión mejorará el modelo por su contribución media a este $\mu_{k,t}$, sujeto a un término de penalización que depende del número de veces que ha sido seleccionado y el número de rondas actual $\sqrt{\frac{\ln t}{N_{k,t}}}$, para evitar el sesgo hacia los mejores clientes valuados (*exploración*) y una inanición de los menos valuados (*exploración*). Formalmente la función de valuación 4.4 genera estos límites superiores de confianza en los que se basa el servidor para seleccionarlos. GreedyFed se basa en la idea de UCB para poder valuar los clientes en base a su contribución marginal media por valores de Shapley, sin embargo, no utiliza el término explícito de penalización, sino que elige vorazmente los clientes favoritos (i.e. GreedyFed promueve la explotación de los mejores clientes).

$$f_{k,t} = \underbrace{\mu_{k,t}}_{\text{contribución marginal media}} + \underbrace{\sqrt{\frac{\ln t}{N_{k,t}}}}_{\text{término de penalización}} \quad (4.4)$$

Por otro lado tenemos S-FedAvg [59], que siguiendo la interpretación de valuaciones que AFL (sección 4.5.1), construye una distribución de probabilidad en base a los valores de Shapley de los clientes. Esto permite el uso de un término *Softmax* a la distribución de probabilidad para poder suavizar las probabilidades de aquellos clientes con menores valores de Shapley, actuando como un término de *exploración*. Más adelante en esta sección veremos como GreedyFed, a diferencia de UCB y S-FedAvg, aunque no implemente un término explícito de exploración (para hacer frente a un sobreajuste del modelo), consigue hacer frente a este problema con su método de inicialización

de valores de Shapley (sección 4.5.2).

Formalmente, se modela un problema de FL con N clientes, como un juego cooperativo de N jugadores, donde se busca obtener un conjunto S_t de $M < N$ clientes seleccionados para una ronda t . Cada cliente *se valúa* con una función de valuación SV_k que da un valor de Shapley para cada cliente k . Los valores de Shapley miden la *contribución marginal media* de cada jugador sobre todos los subconjuntos S de S_t . Cada subconjunto S se asocia con un valor de utilidad dada por la función de utilidad $U : 2^{[M]} \rightarrow \mathbb{R}$ que denota un valor recompensa para cada subconjunto S . Un ejemplo natural de U es el negativo de la pérdida en validación. El SV de cada cliente k se expresa con 4.5

$$SV_k = \frac{1}{M} \sum_{S \in [M] \setminus k} \frac{U(S \cup k) - U(S)}{\binom{M-1}{|S|}} \quad (4.5)$$

Como podemos ver, el cálculo de los valores de Shapley consiste en cuantificar el cambio medio que sufre el modelo (la pérdida sobre un conjunto de validación \mathcal{D}_{val}) sobre **todas las permutaciones** de $S \cup k$, para un cliente k . Por tanto, al tratarse de un problema combinatorio, para hacerlo tratable o viable en escenarios prácticos se utiliza una aproximación de sampleo basado Monte Carlo llamado GtG-Shapley [99]; en este el cálculo se realiza con la generación aleatoria continua de permutaciones, calculando los valores de Shapley hasta que estos valores converjan o se supere un límite de iteraciones. En la práctica GTG-Shapley supera muchos otros esquemas de aproximación de SV. En el cálculo de estos SVs, se necesita comparar los clientes entre sí, esto debido a que debemos de escoger M clientes de N en total (no es obligatorio, pero en la mayoría de casos en FL se elige un subconjunto de clientes), por lo que se usarán valores de Shapley acumulativos usando la media de los valores SV_k en rondas previas 4.6⁴. En el caso de GreedyFed, al igual que en UCB y S-FedAvg, se utiliza la media de estos SV para no favorecer a clientes que fueron menos seleccionados. No se considera la valuación de los clientes k que no fueron seleccionados en una ronda t : $SV_k^{(t)} = 0$ (la contribución en la ronda actual fue nula ya que no fueron seleccionados).

$$SV_k = \frac{1}{t} \sum_t SV_k^{(t)} \quad (4.6)$$

⁴Nótese que estos valores SV medios son diferentes a los valores medios de las contribuciones marginales en GtG-Shapley; estos últimos intentan obtener una representación media de la contribución marginal de los clientes, los primeros se usan para ir acumulando estas contribuciones medias sobre las rondas de entrenamiento.

Compensación Exploración-Explotación

Como hemos comentado anteriormente, una diferencia esencial en el procedimiento de selección de **GreedyFed** en comparación a otras técnicas como UCB o S-FedAvg es que no tiene un término de exploración **explícita**, esto es, la selección de clientes con menores valuaciones, sino que se centra enteramente en la explotación de los clientes más contribuyentes. Sin embargo, GreedyFed involucra una fase de **exploración implícita** en su especificación de una manera muy sencilla: Se inicializan los SV de cada cliente siguiendo un muestreo por Round-Robin (RR) sobre M clientes (seleccionados aleatoriamente) en cada ronda. De esta manera, se permite un exploración fuerte al principio de la selección, y subsecuentemente, se va realizando una selección *greedy* de los mejores M clientes, permitiendo una rápida convergencia y minimizando la sobrecarga de comunicación.

Con RR GreedyFed consigue por una parte asegurarse que todos los clientes son valuados y explorados. También hay que notar que en RR se pretende explotar estos clientes, por lo que esta explotación temprana lleva a un descubrimiento del espacio de búsqueda más longevo en clientes no antes vistos de alta calidad. Por otra parte, GreedyFed consigue explorar otros clientes por el decrecimiento subsecuente de las valuaciones de los clientes, esto debido a la propiedad aditiva de los SV. Esto tiene sentido ya que cuando el modelo ajusta sobre los mejores M clientes en cada ronda, la contribución decrece ya que la diferencia entre las pérdidas decrece $\sum_{k \in S_t} SV_k^{(t)} = \mathcal{L}(w^{(t)}; \mathcal{D}_{val}) - \mathcal{L}(w^{(t+1)}; \mathcal{D}_{val})$. Por lo que, GreedyFed actúa eligiendo **otros clientes valiosos**; seleccionando los siguientes mejores M clientes, a diferencia de los menos explorados como en UCB. Sin embargo, dependiendo de la configuración establecida de GreedyFed, el efecto de esta exploración implícita puede atrasarse, siendo necesario un número de rondas mucho mayor, o un presupuesto de clientes M mayor.

Algoritmo GreedyFed

La implementación de **GreedyFed** se separa por un lado en el bucle de entrenamiento, donde se detalla la selección de los clientes (RR o *greedy*), el entrenamiento y la acumulación de los valores de Shapley. Y por otro lado, la implementación del algoritmo GTG-Shapley en el lado del servidor. En los Algoritmos 4 y 5 se muestran las dos implementaciones respectivamente. En estos algoritmos se utilizan las siguientes funciones: `ModelAverage(n_k, w_k)` calcula una suma ponderada de los modelos locales w_k de cada cliente k con pesos proporcionales a n_k , en nuestro caso utilizaremos FedAvg (mismas ponderaciones). `ClientUpdate` realiza el ajuste del modelo sobre los datos de un cliente \mathcal{D}_k partiendo del modelo descargado del servidor.

Algoritmo 4 Algoritmo Greedy Shapley Client Selection. Extraído del paper original [60].

Input: N clientes con dataset $\{\mathcal{D}_k\}_{k=0}^K$, servidor con dataset de validación \mathcal{D}_{val} , modelo global inicial $w^{(0)}$, número de rondas de entrenamiento T , presupuesto de selección de clientes M

Hyperparameters: Épocas de entrenamiento E , tamaño de mini-batches por época B , learning rate η , momentum γ

Output: Modelo entrenado $w^{(T)}$

Initialize: Número de selecciones de los clientes $N_k = 0, \forall k \in [K]$

```

1: for  $t = 0, 1, 2, \dots, T - 1$  do
2:   if  $t < \lceil \frac{N}{M} \rceil$  then
3:      $S_t = \{t, t + 1, \dots, t + M - 1\}$      $\triangleright$  Round-Robin (orden aleatorio)
4:   else
5:      $S_t = M$  clientes con mayor SV            $\triangleright$  Selección Greedy
6:   end if
7:   for client  $k$  in  $S_t$  do                 $\triangleright$  Actualización del modelo
8:      $w_k^{(t+1)} = \text{ClientUpdate}(\mathcal{D}_k, w^{(t)}; E, B, \eta, \gamma)$ 
9:      $N_k \leftarrow N_k + 1$ 
10:  end for
11:   $w^{(t+1)} = \text{ModelAverage}(n_k, w_k^{(t+1)} : k \in S_t)$ 
12:   $\{SV_k^{(t)}\}_{k \in S_t} = \text{GTG-Shapley}(w^{(t)}, \{w_k^{(t+1)}\}, \mathcal{D}_{\text{val}})$ 
13:  for client  $k$  in  $S_t$  do
14:     $SV_k \leftarrow \frac{(N_k - 1)SV_k + SV_k^{(t)}}{N_k}$             $\triangleright$  Media
15:  end for
16: end for
17: return  $w^{(T)}$ 

```

Algoritmo 5 Aproximación de Valores de Shapley (Server-side) GTG-Shapley.

Input: Modelo global actual $w^{(t)}$, actualizaciones de los clientes $\{w_k^{(t+1)}\}$, dataset de validación del servidor \mathcal{D}_{val} .

Hyperparameters: Umbral de error ϵ , iteraciones máximas T

Output: Valores de Shapley $\{SV_k\}_{k \in S_t}$

Initialize: $SV_k = 0, \forall k \in S_t$

```

1: Calcular  $w^{(t+1)} = \text{ModelAverage}(n_k, w_k^{(t+1)} : k \in S_t)$ 
2:  $v_0 = \mathcal{U}(w^{(t)})$ ,  $v_M = \mathcal{U}(w^{(t+1)})$ ,  $\mathcal{U}(w) := -\mathcal{L}(w; \mathcal{D}_{\text{val}})$ 
3: if  $|v_M - v_0| < \epsilon$  then
4:   return  $SV_k^{(0)} = 0, \forall k$ 
5: else
6:   for  $\tau = 0, 1, \dots, T - 1$  do
7:     for client  $k \in S_t$  do
8:       permutar  $S_t \setminus \{k\} : \pi_\tau[0] = k, \pi_\tau[1 : M]$ 
9:        $v_j = v_0$ 
10:      for  $j = 1, 2, \dots, M$  do
11:        if  $|v_M - v_j| < \epsilon$  then
12:           $v_{j+1} = v_j$ 
13:        else
14:          Subconjunto de clientes  $S^* = \pi_\tau[: j]$ 
15:           $w' = \text{ModelAverage}(n_k, w_k^{(t+1)}, S^*)$ 
16:           $v_{j+1} = \mathcal{U}(w')$ 
17:        end if
18:         $SV_{\pi_\tau[j]}^{(\tau)} = \frac{\tau-1}{\tau} SV_{\pi_\tau[j]}^{(\tau-1)} + \frac{1}{\tau} (v_{j+1} - v_j)$ 
19:         $v_j = v_{j+1}$ 
20:      end for
21:    end for
22:    break si hay convergencia
23:  end for
24: end if
25: return  $SV_k^{(\tau)}, k \in S_t$ 

```

4.5.3. HybridFL

Este método de selección es el primero de los dos que veremos que pertenecen a la familia de técnicas de selección de clientes enfocados en los recursos de los clientes. En concreto, ambos HybridFL y Dyn-HybridFL (4.5.4) están montados sobre el protocolo de selección de clientes FedCS [8], que ha sido de los pioneros en los métodos de selección en base a las restricciones de tiempo y recursos en escenarios de **clientes heterogéneos**, por tanto, también daremos una introducción a este protocolo, necesario para entender HybridFL. Dyn-HybridFL por el otro lado, se construye sobre HybridFL como una mejora a una debilidad de este último.

El protocolo FedCS [8], es un técnica de selección de clientes que basa su criterio en los recursos computacionales, de comunicación y de datos de clientes heterogéneos; es decir, clientes con especificaciones y características diferentes entre sí en cuanto a recursos y datos. [8] basa el diseño e implementación del protocolo en escenarios MEC [100], donde los clientes son en su mayoría teléfonos móviles distribuidos y el servidor central que recoge los parámetros de cada cliente, es un operador de MEC; por tanto, en esta sección y la siguiente (4.5.4), nos referiremos indistintamente a un operador MEC como el servidor central y viceversa.

Una representación visual del protocolo que implementa FedCS es la mostrada en la figura 4.9. Como podemos ver en el, FedCS incorpora nuevos pasos al protocolo estándar que hemos podido ver en 1 (capítulo 2). Estos nuevos parámetros son los de **Client Selection** y **Resource Request**⁵.

La idea principal del protocolo FedCS es que, en vez de seleccionar una fracción C de clientes K : $\lceil K \times C \rceil$ de forma aleatoria, se realiza un paso de selección de clientes **Client Selection** de entre los $\lceil K \times C \rceil$ clientes seleccionados aleatoriamente⁶. Se introduce por tanto también un paso de petición de información de recursos **Resource Request**, por el que el operador MEC (servidor central), realiza una petición a los $\lceil K \times C \rceil$ clientes aleatorios para recoger la información de sus recursos. La información que puede pedir el servidor varían en el diseño del algoritmo de selección: estado de los canales inalámbricos, ancho de banda, capacidades computacionales (CPUs o GPUs, ambos, ...), tamaño de los recursos de datos que posee, por ejemplo la cantidad de ejemplos de cada clase en un problema de clasificación, etc. El servidor utiliza esta información para seleccionar posteriormente los clientes de entrenamiento en el paso de **Client Selection**, para el que *estima*

⁵Obviamos los prefijados con *Scheduled* ya que son los mismos pasos que en 1 pero con una estimación de su tiempo de reloj transcurrido.

⁶Nótese que aunque estemos seleccionando clientes determinados, estos se toman de forma aleatoria, esto para evitar introducir un sesgo en la selección a los mejores $\lceil K \times C \rceil$ clientes, salvando por un lado la estocasticidad y por otro lado, evitando la inanición de los *peores* clientes.

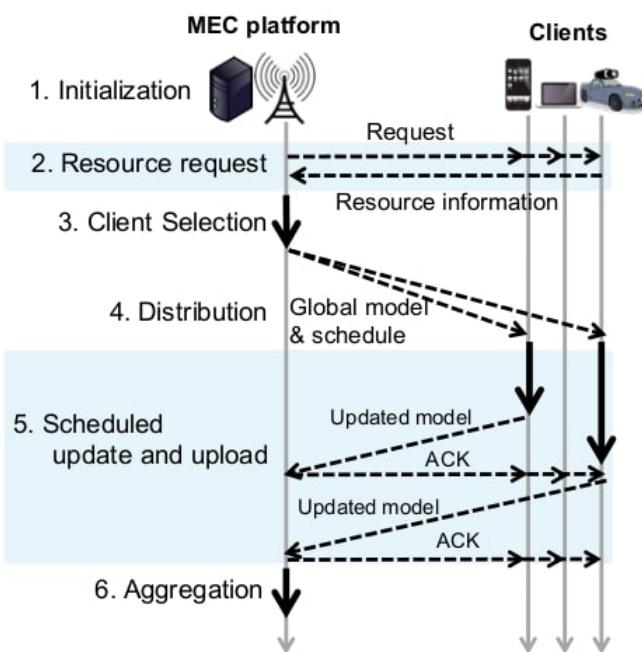


Figura 4.9: Vista general del protocolo FedCS. Las líneas continuas denotan procesos computacionales (tanto del servidor como de los clientes) y las discontinuas, procesos de telecomunicación. El término *Scheduled* se refiere a las actualizaciones y subidas del modelo que se estiman de cumplir con los *deadlines* impuestos. Imagen extraída de [8].

el tiempo de distribución (tiempo de descarga), el tiempo de actualización y de subida agendadas de los modelos locales. Los pasos de agregación e inicialización son similares a los propuestos en [4] para el algoritmo FedAvg.

El protocolo FedCS, intenta así mitigar el problema de la sobrecarga en las comunicaciones de un escenario de aprendizaje federado (FL). Mediante la cooperación de los clientes a proporcionar información acerca del estado de sus recursos inalámbricos, computacionales y de datos, FedCS logra una selección sensata en cuanto a que clientes seleccionar sujetos a cumplir las limitaciones de tiempo impuestas de antemano. Esto hace que el entrenamiento por ronda sea eficiente, maximizando a la vez el número de actualizaciones por ronda, logrando una convergencia hacia un buen rendimiento en un número de iteraciones menor al algoritmo convencional FedAvg tanto para datos IID como para datos no-IID. Sin embargo, los resultados experimentales de FedCS en [8], muestran que el protocolo es vulnerable a la presencia de datos no idéntica e independientemente distribuidos, aunque supere el *baseline* FedAvg. Como hemos explicado en el estudio del algoritmo GreedyFed (sección 4.5.2), una de las principales justificaciones de los algoritmos sesgados (i.e. algoritmos que no buscan la participación justa de los clientes), es que en la práctica, los datos en FL están altamente sesgados con un alto índice de datos no-IID, siendo un foco y principal debilidad de este protocolo de FedCS.

HybridFL [58] propone un mecanismo cooperativo para **mitigar la degradación inducida por el alto índice de datos No-IID en un escenario de FL**. Esto lo realiza asumiendo que un número limitado de clientes (e.g. menos del 1%) **permiten subir sus datos** al servidor central de manera que este último entrena de manera centralizada sobre estos datos, induciendo un entrenamiento del modelo global con datos IID, mejorando el rendimiento del modelo final. Así se propone un mecanismo de enfoque *híbrido* en el que el servidor ajusta el modelo global con los datos obtenidos de estos clientes, construyendo así un conjunto de datos IID, y agrega las actualizaciones locales del modelo de los demás clientes. Este enfoque híbrido, a diferencia de FedCS, incrementa el número de clientes participativos, y por tanto, los datos usados por ronda; mejorando así la precisión del modelo entrenado por esta técnica de selección de clientes.

Protocolo HybridFL

Como hemos explicado, HybridFL basa su protocolo en el de FedCS. Como podemos observar en la Figura 4.10, se consideran los mismo pasos que en FedCS: el paso de petición de información de los clientes (**Resource Information**), la selección de los clientes (**Client Selection**), la subida síncrona de los modelos actualizados, etc. Y se agregan dos nuevos pasos

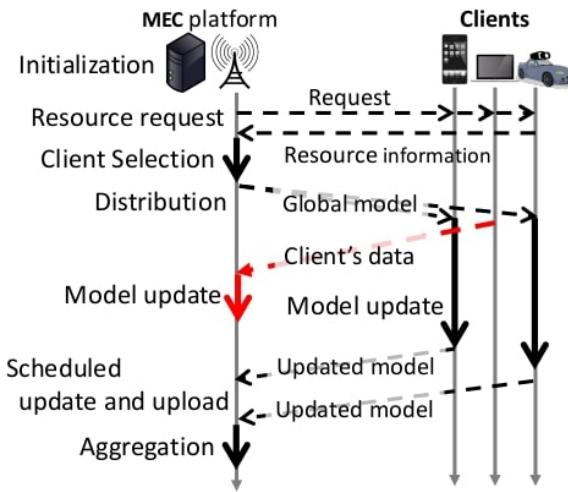


Figura 4.10: Vista general del protocolo usado por HybridFL. Nótese las similitudes con FedCS a excepción del paso se subida de datos de los clientes al servidor (líneas rojas). Imagen extraída de [58].

en el que un número de clientes limitado suben sus datos al servidor y el servidor de forma paralela a los demás clientes ajusta el modelo con estos datos. En el Algoritmo 6 se pueden observar los pasos llevados a cabo en 4.10.

La idea principal del protocolo de HybridFL es que una parte de los clientes se encarga de entrenar localmente el modelo mientras que otra parte sube sus datos al servidor y este último entrena el modelo con esos datos. El servidor en **Request Information** pregunta a clientes aleatorios por sus recursos computacionales, de comunicación y la cantidad de datos por clase, así como preguntar si el cliente está dispuesto a subir sus datos al servidor; con esta información, el servidor es capaz de estimar en el paso siguiente el tiempo requerido para la distribución, actualización y subida de los modelos y finalmente decidir que clientes serán los seleccionados (similar a FedCS). Esta información será utilizada también en el paso 3 para seleccionar clientes que suben sus datos en los límites de tiempo requeridos. En los pasos siguientes, se distribuye el modelo global, los clientes que suben sus datos los suben al servidor, este junto a los clientes seleccionados entrena en paralelo el modelo con sus datos globales y el servidor con los datos obtenidos de los clientes (pasos 4 y 5). Finalmente, los clientes que entran localmente suben sus modelos actualizados (paso 6) y el servidor agrega todos los modelos completando una iteración (paso 6).

Algoritmo 6 Protocolo HybridFL. K denota el número de clientes totales, $C \in (0, 1]$ la fracción de clientes aleatorios que reciben la petición de recursos en cada ronda. Protocolo extraído de [58].

- 1: **Initialization:** El servidor primero inicializa un modelo global ya sea aleatoriamente o preentrenando con datos públicos.
 - 2: **Resource Request:** El servidor pide a $\lceil K \times C \rceil$ clientes seleccionados aleatoriamente a participar en la ronda de entrenamiento actual. Los clientes que reciben la petición notifican al servidor sobre su información de recursos y datos y tanto si permiten o no la subida de sus datos.
 - 3: **Client and Data Selection:** Usando la información de los clientes, para cumplir los pasos en un tiempo limitado impuesto (*deadline*), el servidor determina cuales de los clientes proceden a los siguientes pasos. Selecciona los conjuntos de clientes que entrena localmente el modelo (selección de clientes) y los clientes que suben sus datos al servidor (selección de datos).
 - 4: **Distribution:** El servidor distribuye los parámetros del modelo global a los clientes seleccionados para actualizar el modelo localmente.
 - 5: **Model Update and Data Upload:** Cada conjunto de los clientes seleccionados actualizan los modelos o suben sus datos para clases específicas en paralelo.
 - 6: **Scheduled Upload:** Los clientes seleccionados para actualizar los modelos localmente suben sus parámetros nuevos al servidor.
 - 7: **Aggregation:** El servidor entonces agrega los nuevos parámetros de los clientes y por el servidor, y asigna este nuevo modelo agregado al nuevo modelo global.
 - 8: Todos los pasos a excepción de **Initialization**, se repiten de forma iterativa hasta que alcance un rendimiento final deseado o se cumpla el límite de tiempo de entrenamiento T_{final} .
-

Selección de clientes y de datos

HybridFL propone dos selecciones, una de clientes y otra de datos, correspondientes a la selección de clientes que entran con sus datos locales, y los datos que otros clientes suben al servidor.

La selección de clientes (paso **Client Selection**) tiene como objetivo permitir que el servidor central agregue la mayor cantidad de actualizaciones posibles⁷ dentro de un cierto *deadline* de tiempo (cierto criterio de las condiciones de los clientes). Nótese que a diferencia de los métodos basados en el rendimiento del modelo (GreedyFed [60] y AFL [57]), aquí tratan el problema de la sobrecarga de comunicación incrementando el número de actualizaciones del modelo, para así converger a un rendimiento deseado en un menor número de rondas. Por tanto, no seleccionamos un conjunto de clientes que se estimen mejores para converger el modelo, sino que maximiza el número de actualizaciones sujetas a restricciones de tiempo.

El algoritmo de **Client Selection** se resume en un problema de optimización expresado en 4.7. El objetivo es maximizar el número de clientes seleccionados para el entrenamiento $|S_t|$ sujeto a la condición de que el tiempo total estimado de la ronda, exceptuando el tiempo del paso **Resource Request** (es decir, a partir del paso **Client Selection**), esté dentro de un tiempo de plazo o *deadline* definido. El tiempo total de ronda resulta en una estimación que el servidor central realiza en base a la información proporcionada por los clientes en la fase de petición de información.

En 4.7 se muestra como se puede computar el tiempo estimado de cada ronda. T_{cs} y T_{agg} son los tiempos que toma el servidor en elegir los clientes y agregar las actualizaciones de los modelos locales al modelo global. $T_{S_t}^d$ es el tiempo de distribución del modelo global a los clientes seleccionados (nótese que depende por tanto de S_t). Θ_k es el tiempo transcurrido en la actualización y subida de los modelos locales hasta el cliente k ⁸ y que se expresa por 4.8. Note que el tiempo de las subidas es simplemente su acumulación entre todos los clientes hasta i , ya que las subidas son secuenciales (subida síncrona); mientras que las actualizaciones son paralelas entre todos los clientes, solo se toma el tiempo transcurrido no paralelo, es decir, el tiempo de actualización de un cliente no consume T_i^{UD} siempre y cuando este dentro del tiempo transcurrido $T_i^{UD} < \Theta_{j-1}$.

⁷Esto se basa en lo visto en [4] en el que a mayor cantidad de clientes seleccionados, menor cantidad de rondas serán necesarias para alcanzar cierto rendimiento. En aquel trabajo se refería a esta estrategia como *incrementar la computación en los clientes*.

⁸Recordemos que la subida se realiza de manera ordenada, por tanto $\Theta_{|S_t|}$ corresponde al tiempo transcurrido de actualización y subida total.

$$\begin{aligned} & \max_{S_t} |S_t| \\ \text{s.t. } & T_{\text{round}} \geq T_{cs} + T_{S_t}^d + \Theta_{|S_t|} + T_{agg} \end{aligned} \quad (4.7)$$

$$\Theta_i = \begin{cases} 0 & \text{if } i = 0; \\ T_i^{UD} + T_i^{UL} & \text{otherwise,} \end{cases} \quad (4.8)$$

$$T_i^{UD} = \sum_{j=1}^i \max \{0, t_{k_j}^{UD} - \Theta_{j-1}\} \quad (4.9)$$

$$T_i^{UL} = \sum_{j=1}^i t_{k_j}^{UL} \quad (4.10)$$

En HybridFL, se denomina el tiempo estimado transcurrido de una ronda cuando se incluye un cliente k a S_t como $f(S, k) = \Theta_k = T_{\text{inc}}(S, k)$ que se calcula de la manera que hemos visto anteriormente en 4.8. Además, HybridFL añade un término a la ecuación denominado coeficiente de variación $CV(N_r)$ expresado por 4.11

$$CV(N_r) = \frac{\sum_{l=1}^L (n_l - \bar{n})^2 / L}{\bar{n}} \quad (4.11)$$

donde $\bar{n} = \sum_{l=1}^L n_l / L$. Este coeficiente se evalúa por ronda r y **cuantifica el sesgo existente en $N = n_1, \dots, n_l, \dots, n_L$** , siendo n_l el número de datos de la clase l almacenado por los clientes seleccionados. De esta manera, la función de valuación final se expresa con $f(S, k) = T_{\text{inc}}(S, k) \cdot CV(N_r)$; el algoritmo final de selección utilizando $f(S, k)$ como función a optimizar (en este caso minimizar) esta expresado en el Algoritmo 7 que implementa un algoritmo de optimización *greedy* donde la lista de candidatos son los M clientes aleatoriamente y devuelve un conjunto de clientes S_t que minimiza la función objetivo $f(S, k)$. El algoritmo recibe como hiperparámetros T_{round} que es el límite de tiempo impuesto de ronda para el que el algoritmo selecciona clientes iterativamente hasta que el tiempo estimado de ronda t llegue o sobrepase el *deadline* T_{rounds} (*knapsack constraint*).

En cuanto a la selección de datos donde el servidor elige los clientes que permiten la subida de sus datos al servidor en un tiempo limitado. Concretamente, en el trabajo original [58] se establece que los clientes pueden subir sus datos *hasta* que el primer modelo local sea subido al servidor, de ahí que el tiempo estimado para subir los datos depende del tiempo estimado de actualización y subida de los modelos localmente entrenados. El servidor selecciona los datos que pueden ser subidos en este tiempo *por cada clase* (esto evidentemente para evitar sesgo cuando hay desbalanceo de clases).

Algoritmo 7 Algoritmo Client Selection de HybridFL. Algoritmo extraído de [58].

Input: Conjunto de M clientes aleatoriamente seleccionados: K'

Output: Conjunto de clientes seleccionados para actualizar el modelo: S_t

Hyperparameters: T_{round} es el límite de tiempo de ronda estimado para la selección de clientes.

Initialize: $S_t \leftarrow \{\}, t \leftarrow 0$

```

1: while  $|K'| > 0$  do
2:    $x \leftarrow \arg \min_{k \in K'} f(S_t, k)$ 
3:   remover  $x$  de  $K'$ 
4:    $t' \leftarrow t + T_{\text{inc}}(S_t, x)$ 
5:   if  $t' < T_{\text{round}}$  then
6:      $t \leftarrow t'$ 
7:     agregar  $x$  a  $S_t$ 
8:   end if
9: end while
10: return  $S_t$ 
```

Algoritmo 8 Algoritmo de selección de los datos para ser subidos por los clientes.

Input: t^{UD}, U, D_u

Output: Conjunto de datos a ser subidos al servidor: D^{UL}

Initialize: $D^{UL} \leftarrow \emptyset$, flag \leftarrow True

```

1: while flag do
2:   for  $l$  in  $1, \dots, L$  do
3:      $x \leftarrow \arg \max_{u \in U} \theta_u^{\text{avg}}$  where  $d_{ul} \neq \emptyset$ 
4:      $d \leftarrow$  el primer elemento de  $d_{xl}$ 
5:     if  $t_{D^{UL} \cup d} \leq t^{UD}$  then
6:       agregar  $d$  a  $D^{UL}$ 
7:       remover  $d$  de  $d_{xl}$ 
8:     end if
9:     if  $t_{D^{UL} \cup d} > t^{UD}$  or  $D_u = \emptyset, \forall u \in U$  then
10:      flag  $\leftarrow$  False
11:    end if
12:   end for
13: end while
14: return  $D^{UL}$ 
```

Concretamente, el servidor busca construir un **conjunto de datos IID** para poder ser entrenado de forma centralizada. Se sigue luego lo establecido en el Algoritmo 8, donde t^{UD} es el tiempo estimado para el cual los datos son actualizados, D^{UL} son los datos subidos al servidor en el paso **Model Update and Data Upload** (i.e. los datos que se van a subir al servidor) y $t_{D^{\text{UL}}}$ el tiempo requerido para subir los datos indexados por D^{UL} . El algoritmo itera sobre cada clase en orden, este selecciona para una determinada clase un cliente $u \in U$ (U son los clientes que han permitido la subida de sus datos) con mayor *throughput* θ_k^{avg} , i.e. mejores recursos de comunicación, si y solo si tiene datos de esa clase, $d_{ul} \neq \emptyset$; y coge los datos de ese cliente para esa clase d_{ul} para el cual escoge el primer de ellos d y lo añade a D^{UL} . El algoritmo termina cuando $t_{D^{\text{UL}} \cup d} > t^{\text{UD}}$ o si se han agotado los datos para todos los clientes $D_u = \emptyset, \forall u \in U$. Este algoritmo construye un dataset que cumple la propiedad de ser idénticamente distribuido por seleccionar de manera iterativa los datos por clase (cada clase es seleccionada el mismo número de veces) y de ser independientemente distribuido por la aleatoriedad impuesta al principio del algoritmo al seleccionar los M clientes aleatorios a participar en la ronda de entrenamiento (el conjunto es independiente de la selección de los clientes).

Recursos de comunicación y computabilidad

Un factor importante a definir en esta clase de métodos es el cómo vamos a medir la calidad de los recursos tanto en la comunicación como en poder computacional. En este sentido seguiremos las mismas métricas de rendimiento de recursos que en [8], para el que los recursos de comunicación son representados como el **ancho de banda** (i.e. información enviada por unidad de tiempo) medida en Mbit por segundo. Mientras que los recursos computacionales son interpretados como la **cantidad de datos procesados para entrenar el modelo por unidad de tiempo** medidos en datos o ejemplos por segundo. También es importante notar que estos recursos no son estáticos y que cambian con el transcurso de las rondas para todos los clientes; esto se hace así para simular de la forma más realista posible los cambios en el procesamiento de los datos de los dispositivos que participan como clientes (los dispositivos normalmente tienen procesos ejecutándose que demandan recursos de CPU o GPU) y los cambios en los recursos inalámbricos con el transcurso del tiempo (cambios meteorológicos, interferencias, ...).

4.5.4. Dyn-HybridFL

Hemos visto en la sección anterior 4.5.3, que el protocolo HybridFL, específicamente el algoritmo de selección de clientes 7 depende de un hi-

iperparámetro T_{round} que establece el límite de tiempo que transcurre una ronda. Ya desde la propuesta de FedCS en [8] se ha estudiado el efecto que tiene T_{round} sobre el rendimiento del protocolo sobre clientes heterogéneos y se ha mostrado que influye significativamente en el número de actualizaciones totales que se realizan. Concretamente, en el caso de elegir un T_{round} pequeño (e.g. 1 minuto), el algoritmo no logra incorporar tantos clientes a S_t y por tanto reduce el número de actualizaciones y en definitiva el rendimiento del modelo final; además, esto induce un cierto sesgo hacia los clientes con los mejores recursos, tomando el riesgo de que en una selección desafortunada de clientes con peores recursos, S_t pueda ser un *conjunto vacío* (no hay clientes que cumplan los requisitos de tiempo). Por otro lado, si se elige un T_{round} muy elevado (e.g. $T_{\text{round}} = 10$ minutos) esto expande más el baremo de entrada de clientes al conjunto S_t como es esperado, y por tanto aumentaba el número de actualizaciones por ronda; sin embargo, esto a la vez reduce el número de rondas totales hasta completar el límite total de tiempo de entrenamiento T_{final} , y por tanto tomará igual de tiempo en llegar a un rendimiento determinado que con un T_{round} menor. Tanto en FedCS como en HybridFL, los autores llegaron a la conclusión de que establecer un $T_{\text{round}} = \{3, 5\}$ minutos logra tener unos resultados deseados, aunque esto por supuesto depende del problema.

Como hemos visto, establecer un tiempo de ronda T_{round} adecuado para un problema para HybridFL o para FedCS no es un problema trivial, y de hecho es una de las desventajas de tener un algoritmo con hiperparámetros: que tenemos que realizar una búsqueda de los mejores valores de éstos para poder obtener buenos resultados. Sin embargo, T_{round} puede ser seleccionado de forma dinámica y que depende de ciertas condiciones que se *adapten* a la ronda actual, aumentando o disminuyendo el límite de tiempo por ronda de forma inteligente.

En este TFG, además de analizar y comparar los resultados de los algoritmos vistos en las secciones anteriores, se propone un algoritmo basado en HybridFL que **mejora** la debilidad de este en cuanto a definir un T_{round} dinámico en base a los recursos de los clientes en la ronda actual. Este será el último algoritmo que estudiaremos y por evidentes razones pertenecerá al sector de métodos de selección de clientes enfocados a los recursos sobre clientes heterogéneos y compararemos su rendimiento con el método que intenta mejorar, HybridFL. Este nuevo método de selección de clientes la denominaremos **Dyn-HybridFL**.

T_{round} dinámico

La idea principal de poder elegir un T_{round} en base a los recursos de los clientes, es permitir más clientes a ser seleccionados cuando los recursos de

los clientes candidatos son en general *peores*, aumentando T_{round} ; y restringir la entrada de más clientes cuando sean en general *mejores* disminuyendo T_{round} en vista de que al tener mejores recursos habrán clientes que si puedan cumplir las restricciones de tiempo.

Una factor importante es saber cuando mejoran o cuando empeoran los recursos de los clientes. Una forma saber esto es viendo el cambio de los recursos de los clientes en promedio con respecto a los recursos medios de la ronda anterior, de manera que el factor de cambio defina también el cambio del valor T_{round} , todo esto en base al tiempo estimado medio de los clientes en la actualización y subida de los modelos, esto para que el factor de cambio sea en base a la misma unidad de T_{round} . Formalmente, si tenemos los recursos de comunicación y computabilidad de todos los clientes obtenidos a partir del paso **Resource Information** (véase 6) del cual estimamos sus tiempos de actualización y subida: $\{\text{UD}_k^{(t)}\}, \{\text{UP}_k^{(t)}\}$ respectivamente, la representación de los recursos medios de los clientes $k \in K'$ se expresan como 4.12 que suma los valores medios de los tiempos de actualización y subida

$$\phi^{(t)} = \text{UD}^{(t)} + \text{UP}^{(t)}, \text{UD}^{(t)} = \frac{1}{M} \sum_{k \in K'} \text{UD}_k^{(t)}, \text{UP}^{(t)} = \frac{1}{M} \sum_{k \in K'} \text{UP}_k^{(t)} \quad (4.12)$$

Con este valor $\phi^{(t)}$ que representa de forma *general* los recursos medios de los clientes candidatos K' en la ronda actual t en base al tiempo de actualización y subida, establecemos el $T_{\text{round}}^{(t)}$ para la ronda actual t , en base a la relación de la suma de los tiempos medios de los clientes actuales ($\phi^{(t)}$) con los de la ronda anterior 4.13. La expresión 4.14 calcula la **mejora** o *speed up* de los tiempos medios de los clientes con respecto a los de la ronda anterior.

$$T_{\text{round}}^{(t)} = T_{\text{round}}^{(t-1)} \times S \quad (4.13)$$

$$\begin{cases} S = \frac{\phi^{(t)}}{\phi^{(t-1)}}, & \text{si } t > 0 \\ S = 1, & \text{si } t = 0 \end{cases} \quad (4.14)$$

Con estos pasos de cálculo del T_{round} para la ronda actual t el algoritmo de **Client Selection** de Dyn-HybridFL queda definido en el Algoritmo 9⁹.

Es evidente que este algoritmo no es completamente independiente del uso de hiperparámetros, dado que aún tenemos que establecer el *deadline* de ronda inicial $T_{\text{round}}^{(0)}$. Para esto, sencillamente podemos establecer un $T_{\text{round}}^{(0)}$ por defecto obtenido por conocimiento experto o un valor que haya mostrado

⁹Se utiliza la letra griega τ para referirse al número de ronda para así evitar el conflicto con el la variable de tiempo transcurrido acumulado t

Algoritmo 9 Algoritmo Client Selection de Dyn-HybridFL para el uso de T_{round} dinámico.

Input: Conjunto de M clientes aleatoriamente seleccionados: K' , límite del tiempo de ronda de la época anterior $T_{\text{round}}^{(\tau-1)}$.

Output: Conjunto de clientes seleccionados para actualizar el modelo: S_t

Initialize: $S_t \leftarrow \{\}, t \leftarrow 0$

```

1:  $\text{UP}^{(\tau)} \leftarrow \text{Mean}(\{\text{UP}_k^{(\tau)}\})$ 
2:  $\text{UD}^{(\tau)} \leftarrow \text{Mean}(\{\text{UD}_k^{(\tau)}\})$ 
3:  $\phi^{(\tau)} \leftarrow \text{UP}^{(\tau)} + \text{UD}^{(\tau)}$ 
4: if  $\tau = 0$  then
5:    $S \leftarrow 1$ 
6: else
7:    $S = \frac{\phi^{(t)}}{\phi^{(t-1)}}$ 
8: end if
9:  $T_{\text{round}}^{(\tau)} \leftarrow T_{\text{round}}^{(\tau-1)} \times S$ 
10: while  $|K'| > 0$  do
11:    $x \leftarrow \arg \min_{k \in K'} f(S_t, k)$ 
12:   remover  $x$  de  $K'$ 
13:    $t' \leftarrow t + T_{\text{inc}}(S_t, x)$ 
14:   if  $t' < T_{\text{round}}$  then
15:      $t \leftarrow t'$ 
16:     agregar  $x$  a  $S_t$ 
17:   end if
18: end while
19: return  $S_t$ 

```

tener buenos resultados en la literatura, como es el caso de $T_{round} = \{3, 5\}$ minutos en FedCS y HybridFL ([8],[58]).

4.6. Diseño de los experimentos

Este TFG propone un estudio experimental comparativo entre los métodos de selección que hemos visto y descrito a detalle en la sección anterior 4.5. En esta sección describiremos como va a ser el cuerpo experimental en la que nos apoyaremos para poder analizar los resultados y en definitiva, comparar los métodos en base a éstos.

En primer lugar, realizaremos dos experimentos en las que compararemos un subconjunto de métodos entre sí. Concretamente, el primer experimento será entre los algoritmos con enfoque al rendimiento del modelo, mientras que el segundo será entre los que tienen enfoque en los recursos de los clientes. Esto lo haremos así ya que no es viable en principio comparar un método con un enfoque diferente al otro, ya que los que tienen un enfoque concreto también van dirigidos a una configuración de Aprendizaje Federado diferente a los de otro enfoque. Por ejemplo, AFL (4.5.1) así como GreedyFed (4.5.2) no toman en cuenta los recursos de los clientes en cuanto a que solo se enfocan en los datos de éstos; por otro lado HybridFL, FedCS y Dyn-HybridFL si se encuentran en un escenario de clientes heterogéneos en donde los recursos de los clientes toma un rol crítico en la selección de los clientes en cada ronda. Tomando en cuenta esto, es evidente que si tratamos de comparar métodos de diferentes enfoques, no tendría sentido debido a que ambos, aunque *tengan los mismos objetivos*, estos los resuelven con enfoques diferentes que son incompatibles para una comparación. Es por esto que optamos por dos estudios separados en el enfoque de los métodos de selección. Sin embargo, ambos experimentos tendrán como *baseline* el algoritmo de Random Sampling; la comparación con este algoritmo es viable en ambos experimentos ya que no tiene un enfoque concreto en sí, pudiendo ser fácilmente comparado tanto con los que están enfocados en el rendimiento como los enfocados en recursos, seleccionando sencillamente de forma aleatoria los clientes participantes en la ronda actual haciendo caso *omiso* tanto a los datos de los clientes como a los recursos de éstos.

4.6.1. Experimento enfocado en el rendimiento

Para el experimento donde comparamos los métodos enfocados en el rendimiento, concretamente AFL y GreedyFed, vamos a ejecutar el entrenamiento de los modelos comentados en la sección 4.3: el *Multilayer Perceptron* para MNIST y la Red Neuronal Convolutacional (SimpleCNN) para CIFAR-10. En este entrenamiento utilizaremos un número fijo de clientes a

seleccionar M (un *presupuesto* de selección) para cada problema, de manera que los algoritmos de selección elijan el mejor subconjunto de M clientes para actualizar el modelo localmente en cada ronda. Esto lo haremos por un número determinado de rondas totales de entrenamiento T .

Los resultados que obtendremos de estas ejecuciones son por un lado los valores de pérdida del modelo global $\{l_t\}_{t \in [T]}$ para cada ronda t , la precisión o *accuracy* correspondiente $\{Acc_t\}_{t \in [T]}$ de cada ronda t y el tiempo transcurrido de reloj de cada ronda de entrenamiento. La elección de haber tomado esta última “métrica” no del modelo sino del entrenamiento en sí, nos puede interesar en el sentido de que muchos de los algoritmos, especialmente los que generan valores de Shapley como es el caso de GreedyFed, toman un tiempo muy alto de reloj por cada ronda relacionado a factores externos al entrenamiento del modelo, como lo son la generación de los valores de Shapley [99]; por tanto, nos interesa estudiar la compensación que existe de convergencia con el tiempo que toma cada ronda. Por otro lado, elegimos como métrica de rendimiento del modelo la *accuracy* que se define como 4.15 para poder tener una perspectiva más cercana al rendimiento del modelo que solamente la *loss* que indica la función de Entropía Cruzada 4.3.

$$Acc = \frac{\text{Número de predicciones correctas}}{\text{Número total de predicciones}} \quad (4.15)$$

Nótese que en este y los demás casos utilizamos solo la *accuracy* como métrica de rendimiento de los modelos de ML. Esto es porque realmente, solo nos hace falta una métrica de rendimiento para poder realizar nuestro análisis comparativo, ya que solo nos interesa comparar, en este caso, la convergencia del modelo a partir la curva de aprendizaje que indica la precisión de éste. En otras palabras, no nos interesa utilizar otras métricas de rendimiento (e.g. *R2 Score*, *Specificity*, ...) ya que estamos comparando métodos de selección bajo el mismo entrenamiento de los datos (mismos modelos, mismos datasets), con una sola métrica (la *loss* o la *accuracy*) nos basta para llegar a las mismas conclusiones que con otras. En la mayoría de trabajos de selección de clientes en Aprendizaje Federado utilizan para este caso la *accuracy* para poder comparar con otros métodos de selección [43].

El objetivo de estos métodos enfocados en el rendimiento es justamente seleccionar los mejores clientes que contribuyan a la convergencia rápida del modelo. Esto lo podemos medir cómo el número necesario de rondas para llegar a una cierta precisión, al que denominaremos como **RoA@ x^{10}** (Ronda de llegada o *Round of Arrival* a un *accuracy* x). Entonces, un algoritmo de selección A es mejor que un algoritmo B en términos de convergencia si $RoA@x_A < RoA@x_B$.

¹⁰Notación inspirada en las métricas usadas en [8] para FedCS.

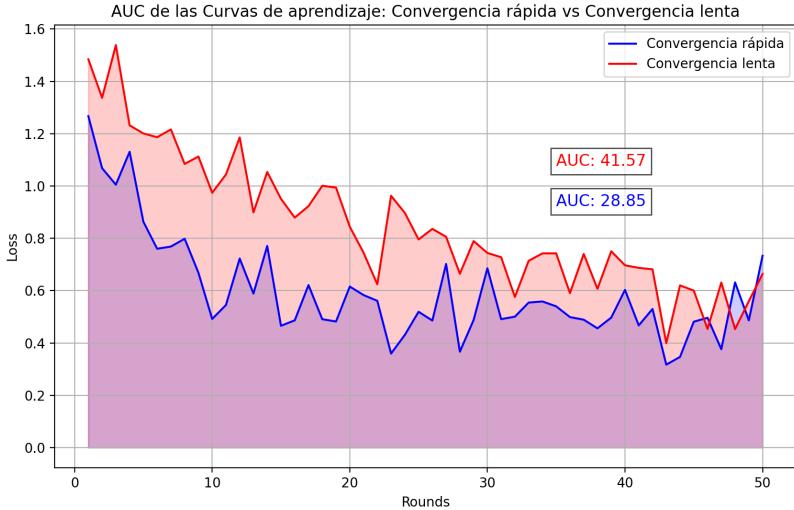


Figura 4.11: Áreas bajo la curva de dos curvas de aprendizaje diferentes. Una de ellas converge más rápido que la otra (azul) y en consecuencia cuando calculamos su AUC el área que ocupa esta curva es menor que la curva que converge más lentamente.

Por otro lado, también nos podría interesar el tiempo transcurrido medio por ronda y total de entrenamiento. Esto nos podría dar información complementaria en cuanto a la viabilidad de cada algoritmo para resolver el problema en un tiempo considerable.

Tuning de Hiperparámetros

Como hemos podido ver en la sección 4.5.1, para AFL necesitamos establecer tres hiperparámetros $\alpha_1, \dots, \alpha_3$. En este caso no podemos utilizar los hiperparámetros de los autores [57], o no sería apropiado, dado que ellos realizaron sus experimentos en otros datasets distintos a los que estamos usando. Por tanto, nosotros deberíamos de realizar nuestro propio tuning de parámetros para AFL.

Para el tuning de hiperparámetros, existen diversas técnicas como la búsqueda aleatoria o uso de algoritmos genéticos. Sin embargo, optaremos por la técnica de *Grid Search* o búsqueda exhaustiva [101] por ser una de las técnicas de tuning de hiperparámetros más usadas por su fácil configuración y uso. Realizaremos la búsqueda en base a la convergencia del modelo. En este caso no utilizaremos la métrica RoA@ x debido a que queremos calcular la convergencia global del modelo y no tener que especificar un *accuracy* concreto. Para ello, lo que haremos será calcular el área bajo la curva apren-

dizaje de la pérdida o AUC, de manera que una curva que converja más rápido tendrá un AUC menor que una que tarde en converger tal y como se ilustra en la Figura 4.11 para dos curvas de aprendizaje diferentes¹¹.

4.6.2. Experimento enfocado en los recursos

Para los experimentos de los métodos enfocados en los recursos seguiremos una línea similar a los anteriores enfocados en el rendimiento: utilizaremos en este caso la MLP para MNIST y la Red Neuronal Convolucional compleja (ComplexCNN) para CIFAR-10 como hemos explicado en la sección 4.3, utilizar un modelo más complejo para así tener un modelo más grande en términos de tamaño y que se pueda simular de manera más realista en el uso de estos métodos. En cuanto a las métricas obtenidas de las ejecuciones tendremos las mismas que en : la pérdida o *loss*, la precisión o *accuracy* y el tiempo transcurrido; las tres para todas las rondas transcurridas.

En lo que concierne al entrenamiento federado de ambos métodos, realizaremos las ejecuciones de ambos más el de Random Sampling, esta vez para un tiempo límite de entrenamiento final T_{final} , de manera que todos terminarán en el mismo tiempo de reloj. Esto viene a ser debido a que el objetivo de estos algoritmos es poder **maximizar** el número de actualizaciones del modelo sujetos a límites y restricciones de tiempo. Luego las métricas de rendimiento que utilizaremos son, por un lado el número total de actualizaciones del modelo N_{UD} , el cual es una clara métrica para saber que método logra maximizar el número de actualizaciones del modelo global y por tanto maximizar su rendimiento final; siguiendo la misma línea de razonamiento, nos interesará saber cuantas rondas el algoritmo ha realizado en total, lo cual nos servirá para determinar entre otras cosas si Dyn-HybridFL en efecto y gracias a la gestión dinámica del T_{round} logra aumentar el número de rondas que realiza. Otra métrica que nos puede ayudar a apreciar si un método logra actualizar más veces el modelo global es el tiempo en alcanzar un cierto *accuracy*, muy similar al antes visto RoA@ x , tenemos ToA@ x ¹², que indica el tiempo de llegada a un *accuracy* x en segundos (*Time of Arrival*). Y por otro lado tendremos el *accuracy* que obtienen los modelos justo al final del *deadline* T_{final} para poder apreciar si los modelos convergen a un modelo con buen rendimiento dentro de la restricción de tiempo de entrenamiento.

Finalmente, para todos los parámetros de configuración e hiperparámetros utilizados en los experimentos realizados están detallados en el capítulo 6 así como otros detalles de configuración de los escenarios de Aprendizaje

¹¹Esta figura no muestra resultados reales y está hecha únicamente para fines ilustrativos.

¹²Esta métrica la prestamos directamente de [8].

Federado necesarios para armar el entorno experimental.

Capítulo 5

Implementación

En este capítulo vamos a dar una introducción a la implementación del código que implementa y ejecuta los métodos de selección de clientes considerados así como utilidades y obtención de resultados. Como hemos comentado en el capítulo 4, la implementación se ha hecho utilizando los Jupyter Notebooks [16], los cuales son una herramienta muy útil para poder realizar análisis de datos y entrenamientos de modelos de ML a la vez que dotamos de texto para describir el proceso de implementación.

Todos los detalles de implementación de los métodos de selección así como la carga de los datos, creación de modelos y construcción del entorno de Aprendizaje Federado además de detalles específicos de cada estrategia están explicados a detalle y de forma estructurada en estos notebooks. Todo el código de implementación se encuentra alojado en el repositorio de GitHub: <https://github.com/mayoras/flex-cs>. En este capítulo por tanto solo describiremos la estructura del proyecto desde el código que se ha implementado así como la estructura general y proceso de implementación de las estrategias de selección de clientes.

5.1. Estructura del código

Listing 5.1: Estructura de ficheros de Notebooks de implementación de los métodos de selección

```
1 .
2 '-- notebooks/
3   # Metodos basados en rendimiento
4   |-- random.ipynb      <- RandomSampling
5   |-- active.ipynb      <- AFL
6   |-- greedy.ipynb      <- GreedyFed
7   # Metodos basados en recursos
8   |-- sched_random.ipynb <- RandomSampling (Scheduled)
```

```

9 |     |-- hybrid.ipynb          <- HybridFL
10|    '-- dynhybrid.ipynb       <- Dyn-HybridFL

```

Listing 5.2: Estructura de ficheros de los resultados de los métodos de selección

```

1 .
2 '-- results/
3   |-- random/
4   |   '-- *.csv           <- Metricas resultantes en formato CSV
5   |-- active/
6   |   '-- *.csv
7   |-- greedy/
8   |   '-- *.csv
9   |-- sched_random/
10  |   '-- *.csv
11  |-- hybrid/
12  |   '-- *.csv
13  |-- dynhybrid/
14  |   '-- *.csv
15  |-- perf_results.ipynb  <- Analisis descriptivo para los metodos
16  | basados en rendimiento
17  |-- res_results.ipynb   <- Analisis descriptivo para los metodos
   | basados en recursos
   '-- util.py            <- Utilidades para cargar, procesar los
                           datos, ...

```

La estructura del código alojado en el repositorio <https://github.com/mayoras/flex-CS> se puede resumir en los siguientes directorios:

- **notebooks.** En este directorio se alojan los notebooks de Jupyter para cada método de selección. Su estructura de árbol se muestra en el Listing 5.1. Puede notar que se han implementado dos tipos de métodos de RandomSampling, uno para cada enfoque de selección; ambos prácticamente realizan la misma selección, con la diferencia de que `sched_random.ipynb` entrena sobre un entorno diseñado para simular restricciones de tiempo (tiempos de espera en actualización y subida de modelos, simular tiempos de distribución del modelo, ...), lo cual es necesario para poder comparar RandomSampling con los otros métodos basados en recursos.
- **results.** En este directorio se alojan todos los resultados obtenidos de las ejecuciones de los métodos de selección, además de proporcionar notebooks adicionales de análisis descriptivos sobre estos datos para generar las gráficas mostradas en este notebook y demás datos. La estructura de árbol es la mostrada en el Listing 5.2. En los directorios de cada método pueden haber datos adicionales como por ejemplo los datos del *Grid Search* realizado para la elección de hiperparámetros en AFL, o la salida dada por la ejecución de entrenamiento de los algoritmos.

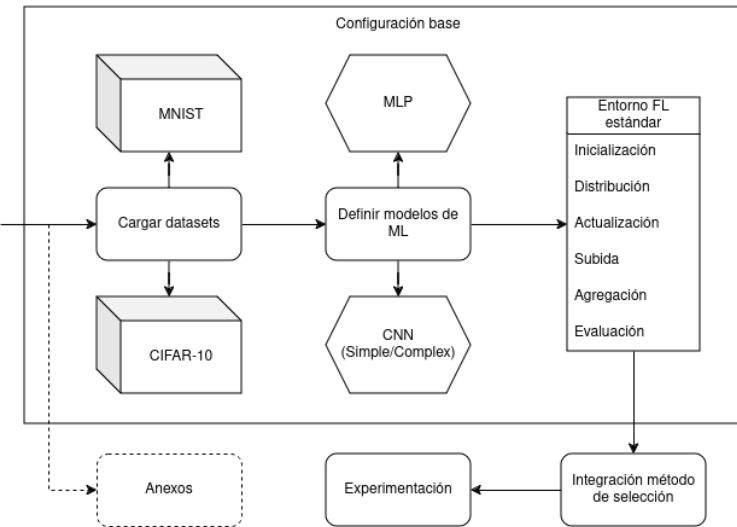


Figura 5.1: Procedimiento de implementación de un método de selección de clientes.

- Además de los directorios en puntos anteriores, existen ficheros de configuración de instalación de paquetes tomados del repositorio del proyecto de FLEXible [67], el framework de Aprendizaje Federado que estamos utilizando.

5.2. Procedimiento de implementación

En los notebooks de implementación de cada algoritmo de selección se detalla un cierta estructura o *outline* que se sigue para todos los métodos de selección. Este diseño lo hemos escogido para poder integrar de una forma metódica cada algoritmo sobre un entorno de Aprendizaje Federado estándar. Por tanto, el procedimiento de implementación e integración de los algoritmos sigue la secuencia mostrada en la Figura 5.1. Como podemos ver, cuando queremos implementar un método de selección lo hacemos sobre una configuración básica de Aprendizaje Federado, específicamente utilizamos la configuración básica inspirada en los ejemplos de FLEXible (<https://github.com/FLEXible-FL/FLEXible/tree/main/notebooks>). También en algunos notebooks, incluimos anexos adicionales que se deben de ejecutar por separado de la ejecución en la fase de experimentación. Por ejemplo, en la implementación de AFL en `active.ipynb` incluimos un anexo para computar el mejor conjunto de hiperparámetros por *Grid Search*; esto evidentemente se espera ser ejecutado antes que la ejecución definitiva donde usamos esos hiperparámetros.

5.3. Ejecución del entrenamiento de los modelos

5.3.1. Usando Google Colab (recomendado)

Como hemos explicado en la sección 1.4, la implementación y ejecución del código usado en este estudio se ha realizado sobre la plataforma de Google Colab [15]. Nosotros **recomendamos** utilizar este servicio para ejecutar los notebooks proporcionados en los materiales de este TFG. En este caso lo único que debe de hacer es dirigirse a la plataforma y subir el fichero del notebook `.ipynb` para luego poder abrirlo y ejecutarlo. Puede ser necesario tener una cuenta de Google e iniciar sesión para poder utilizar la plataforma, debido a que los notebooks se suben a una cuenta de Google Drive.

5.3.2. Ejecución en local

En caso de que se quiera ejecutar los notebooks de los métodos de selección para poder obtener los resultados experimentales de este estudio, es necesario en primer lugar instalar los paquetes necesarios de Python para poder ejecutar todos los notebooks. Estos paquetes se encuentran definidos en el fichero `requirements.txt` y se pueden instalar con el siguiente comando (se necesita tener instalado una versión de Python preferiblemente las versiones 3.10.x):

```
1 $ pip install -r requirements.txt
```

Después de instalar los paquetes, es necesario tener un entorno de ejecución de estos notebooks. El más común es utilizar los Jupyter Notebooks. Si no lo tiene instalado puede hacer de la siguiente manera, instalando el paquete de Jupyter:

```
1 $ pip install notebook
```

A continuación, diríjase a la carpeta donde se encuentra el notebook que quiere ejecutar y ejecute el siguiente comando:

```
1 $ jupyter notebook
```

Esto abrirá una interfaz web donde podrá ejecutar el notebook de forma interactiva y editar las celdas de código.

Advertencia: En los notebooks de este proyecto se incluyen celdas de código de la forma `!<comando>` que en local pueden o no funcionar, esto es así porque los notebooks se desarrollaron en la plataforma Google Colab (Véase sección 1.4 del capítulo de Introducción), que incluye comandos

adicionales para instalar paquetes. No obstante, si usted ha instalado los paquetes necesarios definidos en `requirements.txt` no tendrá problemas con estas celdas de código, sin embargo es importante recalcar este punto.

Nota: En el código proporcionado en el material de la entrega de este TFG los resultados obtenidos de los notebooks se generan (además de en el propio notebook) en el directorio del notebook y no en el de resultados (`results`), esto se ha hecho así ya que las celdas de código escritas en los notebooks no se han ejecutado en local sino en el servicio Google Colab. Es por esto que si se quiere ejecutar los notebooks y obtener los resultados en otra ubicación se debe de indicar explícitamente en el código en las funciones llamadas `save_data`. De todas formas junto al código se proporcionan los datos obtenidos en el directorio `results` en caso de que se quiere ejecutar los notebooks de análisis de resultados.

Capítulo 6

Experimentos y Resultados

En este capítulo vamos mostrar los resultados de los experimentos diseñados en el capítulo 4 en la sección 4.6. Para cada experimento especificaremos la configuración de los parámetros de los que dependen así como la configuración en cuanto a las tareas relacionadas con el Aprendizaje Automático como las tasas de aprendizaje o *learning rates*, *momentum* utilizado, tamaño de *batch*, número de épocas de entrenamiento local, etc. Así como la distribución de los datos de los clientes (a lo que también llamamos *federación* del dataset).

A partir de los resultados de cada experimento, concretamente para los experimentos basados en rendimiento y los basados en los recursos, vamos a realizar un análisis comparativo de estos resultados, lo que conllevaría a estudiar el comportamiento del aprendizaje de cada algoritmo en busca de respuestas en la que concluyamos el porque un algoritmo es mejor que otro, o simplemente justificar el porqué rinden los algoritmos sobre la configuración usada.

Este capítulo se dividirá en las siguientes secciones:

1. **Distribución de los datos.** En esta sección detallaremos la federación de los datos sobre un número K de clientes. Esta federación será compartida sobre *todos* los métodos de selección de clientes, y su objetivo será la distribución **no idéntica e independientemente distribuida** (no-IID) de los datos. La elección de realizar una distribución no-IID de los datasets sobre los K clientes, es debido a que la solución con una distribución IID no es diferente al aprendizaje distribuido [6] y por tanto, para el problema de selección de clientes, el problema es *trivial* para los métodos basados en rendimiento (solo elegimos los clientes con RandomSampling y tendremos el mejor aprendizaje) y no cumpliría la propiedad de clientes heterogéneos *sobre los datos* para los basados en recursos. Por tanto, elegiremos una distribución no-IID

para todos los experimentos.

2. **Configuración para tareas de ML.** Es necesario definir los parámetros usados en las tareas relacionadas con Aprendizaje Automático (ML). Al igual que en **Distribución de los datos**, esta configuración será la utilizada en todos los experimentos¹

6.1. Distribución de los datos

Como hemos comentado al principio de este capítulo, la distribución de los datos será única para todos los métodos de selección y tendrá como objetivo generar un conjunto de clientes heterogéneos en cuanto a sus datos de forma que tanto las clases como el mismo tamaño de los datasets sean distintos, generando únicamente un conjunto global de datos no-IID. Concretamente, distribuiremos los datos tal y como lo hacen en [102]:

“[...] muestrear dos/diez clases para cada cliente para CIFAR-10/CIFAR-100²; Luego, para cada cliente i y una clase seleccionada c , muestreamos $\alpha_{i,c} \sim U(.4, .6)$, y lo asignamos con $\frac{\alpha_{i,c}}{\sum_j \alpha_{j,c}}$ de los ejemplos para esta clase. Repetimos lo anterior para 10, 50 y 100 clientes.”. De esta manera produciremos con “diferentes números de ejemplos por clases.”

Una forma de implementar esto y que viene explicado en el código de este TFG es mediante la asignación de ponderaciones para cada par *cliente-clase*. En [67] muestran como hacer la federación de [102] usando FLEX y es la que utilizaremos en nuestro caso (véase Listing 4.4).

En la Figura 6.1 se muestran la distribución de los datos de MNIST sobre 10 clientes para el caso de una federación IID y una no-IID. Como podemos ver en el caso de una distribución IID 6.1(a), los datos se ven uniformemente distribuidos por clases, mientras que en el caso de No-IID 6.1(b), se distribuyen de forma menos uniforme. Si vemos por ejemplo, la clase 1 en 6.1(a), esta clase está distribuida uniformemente entre los 10 clientes, pero en No-IID 6.1(b), la mayor parte de esta clase la posee el cliente 3, 6, 10, por

¹Note que aunque la CNN utilizada en los experimentos basados en rendimiento (SimpleCNN 4.2) y en recursos (ComplexCNN 4.3) sean diferentes los parámetros de entrenamiento serán los mismos. Esto lo elegimos por conveniencia ya que, como hemos establecido en las métricas de rendimiento en 4.6.2, el uso de mejores o peores parámetros no afectan a las conclusiones que obtendremos **si todos los métodos lo utilizan** y no afecten en el mismo proceso de selección de los clientes (e.g. cambiar el presupuesto de selección por ronda M , o la condición de parada T).

²En nuestro caso solo será para CIFAR-10 y MNIST, es decir, suponemos siempre 10 clases.

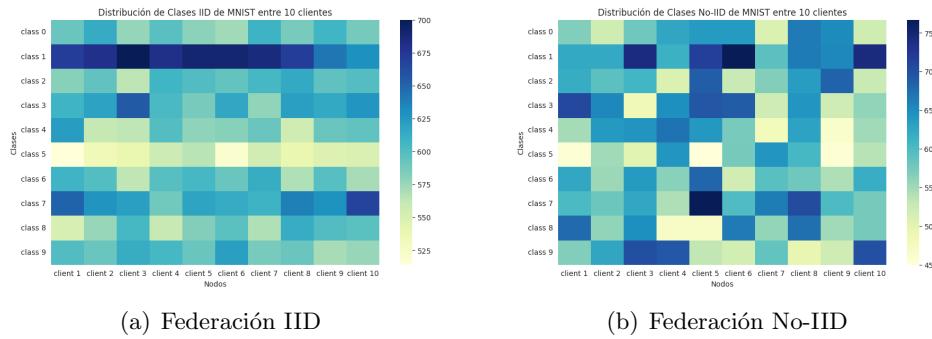


Figura 6.1: Mapa de calor de la federación de MNIST sobre 10 nodos/clientes para los dos casos de distribución IID y No-IID utilizando la federación de [102].

lo que existe una distribución más desplazada (*skewed*) hacia unos clientes que otros. Muchos otros autores en selección de clientes, utilizan funciones de distribución *skewed* como la distribución de Dirichlet en [60], para poder conseguir este tipo de desplazamiento de los datos hacia un subconjunto de clientes. Sin embargo, utilizaremos el dado por [102] que para nosotros será suficiente para conseguir este efecto.

6.2. Configuración para tareas de ML

En el capítulo 4 hemos establecido los modelos de aprendizaje y el optimizador que utilizaremos en los métodos de selección, concretamente hemos optado por una MLP y dos CNN (una simple SimpleCNN y una más compleja ComplexCNN), además hemos optado por un optimizador *Adam* que explicamos en la sección 4.3.2.

En esta sección nos queda por definir los parámetros para el entrenamiento de los modelos en los clientes (actualizaciones de los clientes). Como estos parámetros son completamente independientes del método de selección a usar, y por tanto, no afectan a estos en el análisis comparativo, utilizaremos un mismo conjunto de parámetros de entrenamiento en ML para todos los métodos de selección. En concreto, utilizaremos un número de $E = 5$ épocas, un tamaño de $B = 20$ de *batch*; para el optimizador Adam utilizaremos los valores usados por sus autores en [26] para los datasets de MNIST y CIFAR10 (además de otros): Learning rate $\eta = 0.001$, betas para calcular las medias móviles de los gradientes y sus cuadrados $\beta_1 = 0.9$, $\beta_2 = 0.999$.

6.3. Experimentos de métodos basados en rendimiento

Para este primer análisis comparativo entre los métodos de selección basados en rendimiento del modelo, vamos a ejecutar RandomSampling, GreedyFed y AFL sobre los dos problemas de Visión por Computador anteriormente vistos: MNIST y CIFAR-10, con las siguientes condiciones. En primer lugar, ejecutamos sobre un número de $T = 150$ rondas para MNIST y $T = 100$ rondas para CIFAR-10. Para MNIST, establecemos un total de $K = 300$ clientes donde habrá un presupuesto de selección (máximo número de clientes seleccionados por ronda) $M = 3$, mientras que en CIFAR-10 nos valdremos con $K = 100$ clientes con un presupuesto de selección $M = 6$ clientes por ronda. Estos parámetros globales de entrenamiento fueron fuertemente influenciados por la configuración experimental de los autores de GreedyFed [60] en cuanto a la complejidad computacional que conlleva el incremento de M al calcular los valores de Shapley; aunque en RandomSampling y AFL este incremento apenas se notaba o era despreciable, en GreedyFed aumentaba su complejidad en tiempo de forma cuadrática (se puede apreciar más adelante en la Tabla 6.1 de resultados), es por esto que en CIFAR-10 optamos por reducir a $M = 6$ aunque en su artículo original utilizaba un M más grande, es importante anotar estos cambios ya que los resultados pueden no ser los mismo que los hechos por sus autores y pueden influir en el resultado final.

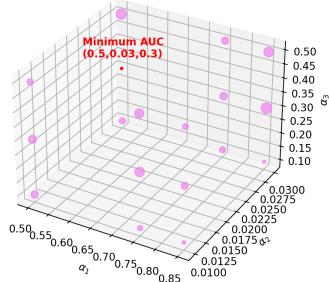
En cuanto a las configuraciones específicas de los métodos, en AFL se eligió un conjunto de hiperparámetros $\alpha_1, \dots, \alpha_3$ por búsqueda exhaustiva como hemos mencionado en la sección 4.6.1, utilizando la técnica de *Grid Search*. En nuestro caso hemos utilizado el siguiente *grid* de parámetros inspirado en los valores utilizados por los autores en [57] 6.1. Sobre este *grid* de parámetros se han ejecutado un número de $T = 15$ rondas sobre MNIST y CIFAR-10, para los que se ha obtenido la curva de aprendizaje de pérdida del modelo sobre todas las combinaciones del *grid*. Calculamos entonces sobre cada curva de aprendizaje el área bajo ésta para determinar su convergencia; de manera que entre más pequeña el área, mejor ha convergido (Véase Figura 4.11).

$$\begin{cases} \alpha_1 = \{0.5, 0.75, 0.85\} \\ \alpha_2 = \{0.01, 0.03\} \\ \alpha_3 = \{0.1, 0.3, 0.5\} \end{cases} \quad (6.1)$$

En la Figura 6.2 podemos observar los resultados de la búsqueda exhaustiva Grid Search para ambos problemas MNIST y CIFAR-10 con AFL³,

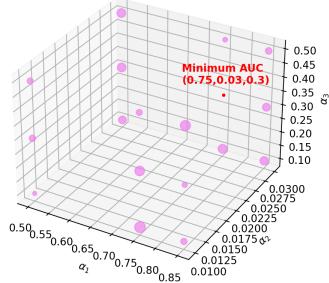
³Se muestran los resultados de las áreas bajo la curva *normalizados* para proporcionar

Valores AUC para las combinaciones de parámetros en Grid Search (MNIST)



(a)

Valores AUC para las combinaciones de parámetros en Grid Search (CIFAR-10)



(b)

Figura 6.2: Valores AUC de las curvas de aprendizaje resultante de la búsqueda exhaustiva de parámetros para AFL (menor valor AUC es mejor). En rojo se marcan los mejores valores AUC que marcan la mayor velocidad en convergencia sobre esos parámetros.

donde podemos concluir en que los parámetros que consiguen una mejor convergencia son $\alpha_1 = 0.5, \alpha_2 = 0.03, \alpha_3 = 0.3$ para MNIST (Subfigura 6.2(a)) y $\alpha_1 = 0.75, \alpha_2 = 0.03, \alpha_3 = 0.3$ para CIFAR-10 (Subfigura 6.2(b)). Estos parámetros serán los utilizados en los experimentos definitivos para AFL y con los cuales compararemos con los otros métodos de selección.

En cuanto al método GreedyFed, existen ciertos parámetros a establecer. En primer lugar, la partición de un subconjunto de datos de validación \mathcal{D}_{val} que es necesario para poder calcular los valores de Shapley con el algoritmo de aproximación GTG-Shapley [99]. Este conjunto de validación lo obtendremos del conjunto original de entrenamiento $\mathcal{D}_{\text{train}}$ de manera que siempre estaremos entrenando y tomando decisiones sobre este conjunto, dejando el de test para solo evaluación. De esta manera se sigue la partición de 6.2,

una mejor visualización.

donde partimos el conjunto de validación de manera que su tamaño $|\mathcal{D}_{\text{val}}|$ sea el 10 % del conjunto total de entrenamiento.

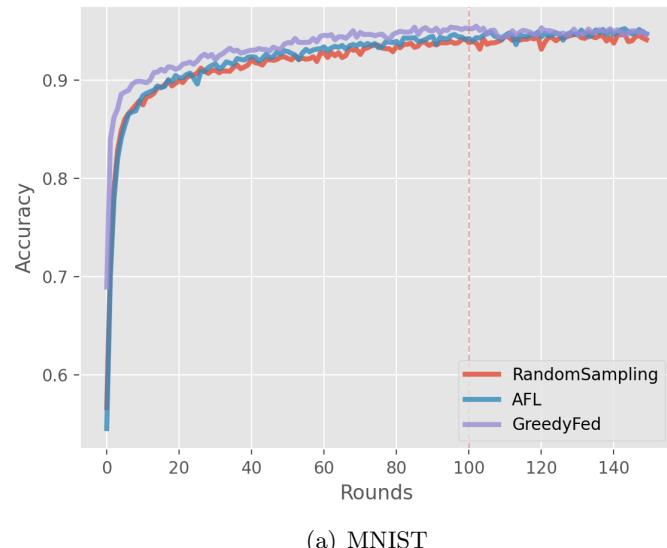
$$\mathcal{D}_{\text{T}} = \mathcal{D}_{\text{train}} \cup \mathcal{D}_{\text{val}}, \begin{cases} N &= |\mathcal{D}_{\text{T}}|, \\ N_{\text{val}} &= \lfloor 0.1 \times N \rfloor, \\ N_{\text{train}} &= N - N_{\text{val}} \end{cases} \quad (6.2)$$

Los siguientes parámetros son relacionados con el algoritmo de aproximación GTG-Shapley. Por una parte, tenemos el total de iteraciones que realiza el algoritmo que se cumplen cuando no llegan a converger los SV del conjunto de clientes seleccionados S_t . En este caso, los autores utilizan un número suficiente de $50 \times |S_t|$, sin embargo, utilizaremos un número menor de $30 \times |S_t|$ por viabilidad de tiempo de calculo pero conservando suficientes iteraciones de tolerancia hasta que los valores converjan. Utilizaremos también un umbral $\epsilon = 10^{-4}$ para establecer el valor contribución mínima para calcular valores de Shapley. Finalmente, utilizamos el criterio de convergencia por defecto del método ([99]) que consiste en computar el cambio de los SV medios en las últimas 20 iteraciones relativos al último valor, y se considera que han convergido cuando este cambio medio es menor que 0.01. Formalmente, si tenemos una secuencia de SV $\{v_1, \dots, v_n\}$, entonces calculamos el cambio medio absoluto relativo al último valor $C_{\text{mean}} = \frac{1}{20} \sum_i |v_i - v_n|$; luego calculamos el cambio relativo a v_n , $C_{\text{rel}} = \frac{C_{\text{mean}}}{|v_n|}$. Si $C_{\text{rel}} < 0.01$ los valores han convergido en los últimos 20 valores, de lo contrario se considera que no han convergido⁴.

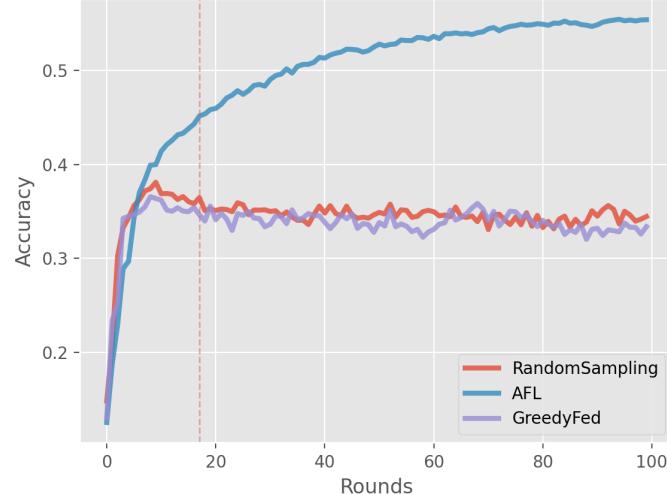
En la Figura 6.3 se muestran las curvas de aprendizaje de los algoritmos basados en el rendimiento del modelo para la precisión del modelo global en cada ronda de entrenamiento y en la Tabla 6.1 se muestran las métricas obtenidas del entrenamiento utilizando los métodos de selección. Empezando por MNIST en la Figura 6.3(a), podemos observar que ambos algoritmos de aprendizaje tienen comportamientos muy similares en cuanto a la convergencia del modelo, siendo GreedyFed el más rápido en converger llegando a un 90 % de *accuracy* (Véase Tabla 6.1) aún en su etapa de inicialización por Round-Robin; mientras que AFL y RandomSampling muestran resultados muy parecidos entre sí, no obstante, AFL presenta un aprendizaje más uniforme con una curva con menos ruido que la de RandomSampling. También podemos notar que en cuanto GreedyFed termina de inicializar los valores SV de sus clientes (ronda $t = 100$), éste presenta un sutil sobreajuste del modelo al empezar a seleccionar los clientes por sus valores de Shapley.

Por otro lado, en CIFAR-10 (Figura 6.3(b)) tenemos comportamientos más bien atípicos que con MNIST. En primer lugar, podemos apreciar un

⁴Este test de convergencia es extraído del código de los autores originales de GreedyFed con ligeras modificaciones: <https://github.com/pringlesinghal/GreedyFed/utils.py>.



(a) MNIST



(b) CIFAR-10

Figura 6.3: Curvas de aprendizaje resultantes de los algoritmos basados en el rendimiento del modelo: RandomSampling, AFL y GreedyFed. La línea roja discontinua indica la ronda final de inicialización Round-Robin de valores de Shapley en GreedyFed.

Dataset	Algoritmo	RoA@0.85	RoA@0.9	Tiempo de ronda medio (s)	Accuracy
MNIST	RandomSampling	5	22	2.6 ± 0.42	0.9409
	AFL	5	17	3.13 ± 0.43	0.9469
	GreedyFed	2	13	205.86 ± 79.27	0.9463
CIFAR10		RoA@0.3	RoA@0.5		
	RandomSampling	2	N/A	6.98 ± 0.7	0.3446
	AFL	5	33	7.17 ± 0.51	0.5538
	GreedyFed	3	N/A	710.7 ± 419.28	0.3338

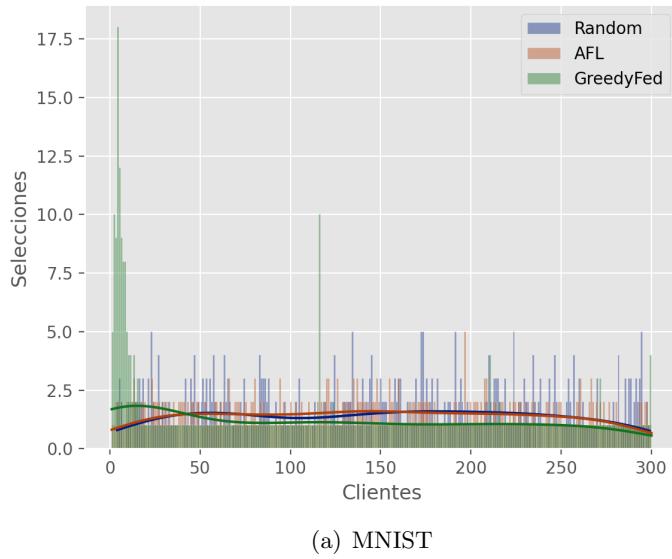
Tabla 6.1: Métricas de rendimiento obtenidas de los experimentos de los métodos de selección basados en rendimiento del modelo. Se marcan en negrita los que resultan vencedores. Los resultados que muestran “N/A” en las métricas RoA@ x indican que no se ha podido llegar a cierta *accuracy* en el número de rondas establecido.

claro sobreajuste del modelo tanto para GreedyFed como para RandomSampling en fases tempranas del aprendizaje, en el caso de GreedyFed es hasta incluso antes de terminar las rondas de inicialización. Sin embargo, AFL es el único de los tres que bajo las mismas condiciones consigue converger de una manera uniforme y monótona a los demás métodos, así como presentar un aprendizaje mucho más directo con una variabilidad con respecto a la media muy baja.

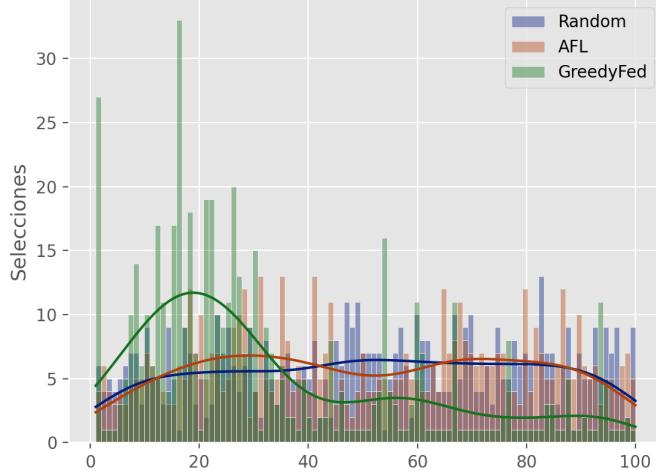
En la Tabla 6.1 también podemos apreciar el tiempo que toma cada estrategia por ronda de media. Podemos ver que en efecto GreedyFed toma un tiempo de cómputo considerablemente más grande que los otros métodos, en gran parte gracias al algoritmo de aproximación GtG-Shapley que crece de forma cuasilineal con el número de clientes por ronda M ([60]).

6.3.1. Análisis de los resultados

Si analizamos las curvas de aprendizaje tanto de MNIST como de CIFAR-10, nos podemos dar cuenta a simple vista que GreedyFed converge de una forma más acelerada que los otros algoritmos en las primeras fases del aprendizaje. Recordemos que estas primeras fases de GreedyFed corresponden a una etapa de exploración, en donde inicializa los valores de todos sus clientes, siendo los primeros clientes en participar en el entrenamiento del modelo. Esto último justifica la rápida convergencia de GreedyFed, si y solo si justamente son **los primeros clientes son los que poseen mejores datos**. En la Figura 6.4 podemos observar las selecciones por clientes de CIFAR-10 y MNIST para los métodos de selección Random, AFL y GreedyFed. Podemos observar como GreedyFed en ambos casos tiene una gran *tendencia* a elegir los primeros clientes. Esto por un lado explica el sobreajuste del modelo con GreedyFed, al final no considera todos los clientes de una forma equitativa como si lo hacen AFL y RandomSampling (el primero a partir de valuar cada cliente por ronda en base a su pérdida y el segundo por selec-



(a) MNIST

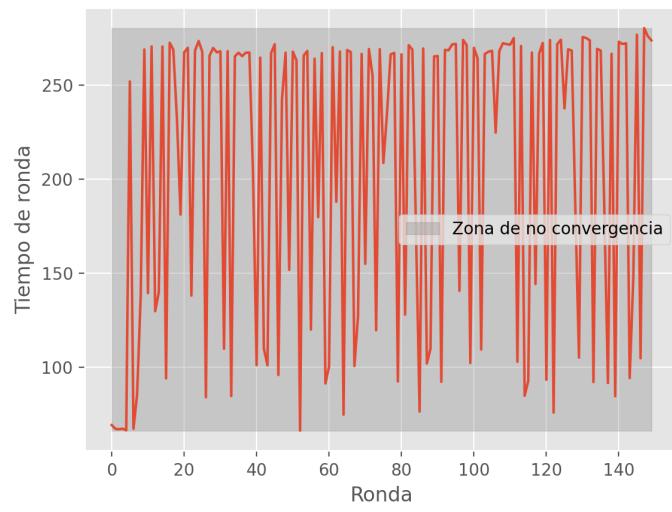


(b) CIFAR-10

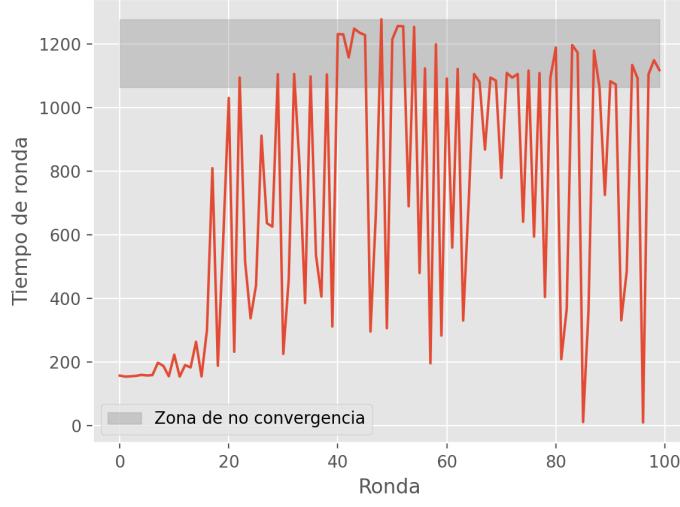
Figura 6.4: Comparativa en la selección de los clientes por los métodos Random, AFL y GreedyFed para MNIST y CIFAR-10. Las gráficas de línea muestran la estimación de la función de densidad de los resultados de selección, o también denominado KDE (*Kernel Density Estimation*) [103].

ción aleatoria y uniforme), pero por otro lado explica una de las deficiencias de GreedyFed: la exploración implícita vista en 4.5.2, es deficiente cuando tenemos pocas selecciones por ronda. Esto lo podemos observar de manera directa en MNIST (Figura 6.4(a)), donde solo tenemos un presupuesto de $M = 3$ clientes por ronda, esto hace que ante un número muy grande de clientes de $K = 300$, GreedyFed no sea capaz de explorar más allá de los primeros clientes y las rondas de inicialización Round-Robin, lo que resulta en un sobreajuste que se manifiesta últimamente en CIFAR-10 donde a diferencia de MNIST, es un problema donde tenemos menos datos (240,000 frente a solo 50,000 en CIFAR-10, Sección 4.2) y por tanto, una peor generalización que se hace notar con un *overfitting* como lo vemos en 6.4(b). MNIST al ser un problema más “*sencillo*”, con tener solo unos cuantos clientes que posean todas las clases (en este caso los primeros seleccionados por RR), ya el modelo podría ser capaz de captar los patrones subyacentes para poder generalizar de manera efectiva; es por esto que GreedyFed converge de manera eficiente para este caso concreto. Mientras que en CIFAR-10, la etapa de inicialización por RR termina de forma más temprana, sin embargo, podemos observar que ya existe un *overfitting* antes de esta finalización que se *agrava* más con las selecciones *greedy* sobre los primeros clientes.

Ante el análisis previo, nos surge la siguiente cuestión: ¿Por qué GreedyFed tiene una tendencia a seleccionar los primeros clientes? En primer lugar, debemos de entender como funciona GreedyFed; GreedyFed realiza una selección voraz de los clientes que tengan una mayor contribución marginal del modelo global (4), tal contribución marginal se calculan con los valores de Shapley (5). Estos valores de Shapley se inicializan para todos los clientes con política Round-Robin (RR), sin embargo, el modelo **se entrena** bajo esta inicialización, por lo que el modelo con que se calculan los SV para los clientes posteriores será muy diferente al modelo que se usa para los primeros clientes, de hecho el modelo para los posteriores estará más *influenciado* por los primeros clientes. La Figura 6.5 podemos observar los tiempos de entrenamiento de cada ronda, note que las rondas donde no han convergiendo los valores de Shapley con el algoritmo GTG-Shapley tienen tiempos de entrenamiento muy elevados. En esta Figura podemos ver claramente como en las primeras rondas de entrenamiento, en la fase de inicialización de los SVs, el calculo de estos valores de Shapley convergen en todo momento y con tiempos mínimos. Esto tiene sentido, ya que en un principio tenemos un modelo *virgen* que no se ha entrenado con ninguno de los clientes, es por esto que convergen estos SVs para los primeros clientes, porque su contribución es o es casi absoluta en el modelo y por tanto su calculo es trivial, mientras que para los clientes posteriores este calculo no es tan sencillo puesto que el modelo tiene influencia de un número mayor de clientes. Esto explicaría entonces porque GreedyFed tiene una gran preferencia hacia los primeros clientes: siempre tienen una gran contribución al modelo en todo el entre-



(a) MNIST



(b) CIFAR-10

Figura 6.5: Tiempos de entrenamiento por ronda de GreedyFed en MNIST y CIFAR-10. El área gris corresponde a la zona de no convergencia derivada de las rondas donde no han convergido los valores de Shapley tomando como límites el mínimo y el máximo tiempo transcurrido.

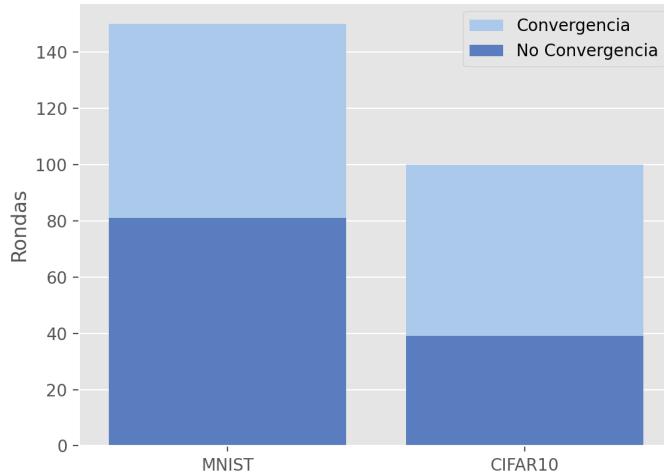


Figura 6.6: Proporción de rondas en los que el algoritmo GtG-Shapley ha convergido en MNIST y CIFAR-10.

namiento por el privilegio de haber sido los primeros en entrenarlo. Por supuesto, GreedyFed tiene una forma de paliar este problema y es con la exploración implícita que hemos comentado anteriormente que toma lugar en la inicialización de valores, sin embargo, en CIFAR-10 (Figura 6.5(b)) al ser esta fase muy corta (hasta la ronda $t = 17$) y solo tener un presupuesto de clientes muy bajo por ronda (solo $M = 6$ clientes) esto hace que GreedyFed no pueda generalizar de manera adecuada y se estanque en un sobreajuste. En cuanto a MNIST (Figura 6.5(a)), vemos que la zona de no convergencia es muy incierta y esto es debido a que el cálculo de los SV se hace de manera muy rápida, en parte por tener un M muy bajo reduce la complejidad del algoritmo GtG-Shapley, lo que hace que el tiempo en general al aproximar los SV sea más homogéneo.

Adicionalmente, podemos observar en la Figura 6.6 la proporción de veces en que los valores de Shapley han convergido o no han convergido para MNIST y CIFAR-10. Podemos observar que en CIFAR-10, el número de rondas donde el algoritmo ha convergido es mayor que en MNIST; esto nos da a entender que en CIFAR-10, GreedyFed elegía a clientes cuyos valores convergían de manera más rápida, es decir, los primeros clientes quienes eran los favoritos a ser seleccionados; luego esto nos indica que en CIFAR-10 hubo definitivamente un sobreajuste por elegir un número mayor de veces a un subconjunto de clientes. En MNIST esto no se logra apreciar y es debido a que al tener un número mayor de rondas de inicialización, GreedyFed exploraba un número mayor de clientes que en CIFAR-10 y por tanto la

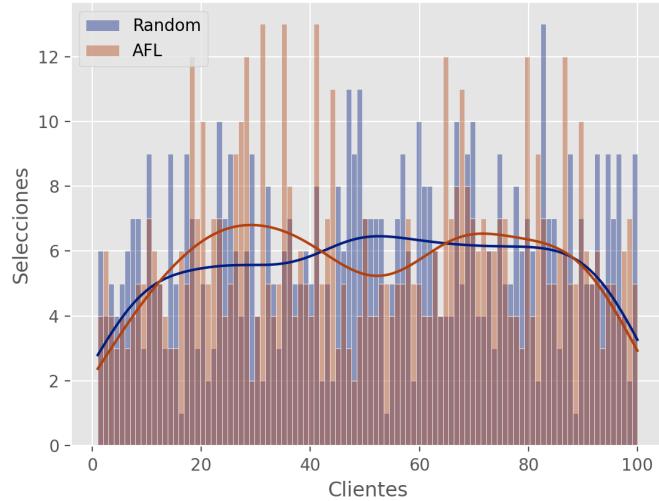


Figura 6.7: Comparativa en la selección de RandomSampling y AFL en CIFAR-10.

proporción de rondas de no convergencia es mayor.

En cuanto a nuestro otra estrategia de selección, AFL presenta un comportamiento muy diferente a GreedyFed. Podemos observar de nuevo en las Figuras 6.4 como AFL realiza una selección lo más homogénea y equitativa posible, lo que es un factor crucial al generalizar el modelo, en especial con CIFAR-10. Esto hace que tanto en MNIST como en CIFAR-10, sea el único que alcanza converger a una precisión esperada del 50 % en un número de rondas de 33, mientras que los demás no pudieron alcanzar estos valores. Sin embargo, podemos observar en las curvas de aprendizaje de CIFAR-10 en 6.3(b) que aunque AFL y RandomSampling elijan de una forma uniforme a los clientes en cada ronda (Figura 6.4(b)), RandomSampling no consigue converger como lo hace AFL, de hecho, presenta un comportamiento muy similar a GreedyFed y en el caso de MNIST AFL logra converger ligeramente a un mejor *accuracy* que RandomSampling. Para analizar esto, basta con ver la selección que hacen ambos métodos en 6.7 para CIFAR-10, en la que aunque RandomSampling presente una mayor homogeneidad que AFL, se puede ver que AFL elige con una mayor frecuencia ciertos clientes. Según el método de AFL (Sección 4.5.1) estos clientes “favoritos” poseen datos que el modelo falla en predecir con mayor frecuencia (poseen una pérdida mayor), esto por tanto hace que el modelo converja más rápido al promover entrenar con datos más *difíciles*. No obstante, para evitar el error de elegir de forma voraz estos clientes en todas las rondas, AFL induce aleatoriedad al seleccionar una proporción α_3 de clientes para evitar *overfitting*, de ahí la

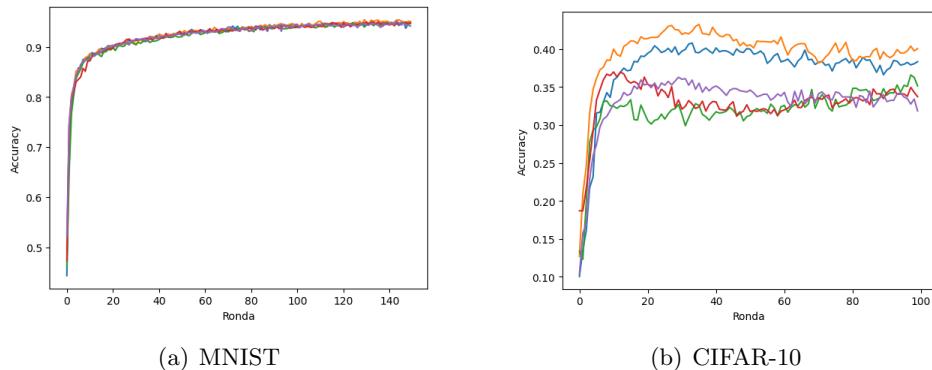


Figura 6.8: Curvas de aprendizaje de RandomSampling de 5 ejecuciones con diferentes semillas aleatorias.

forma característica de la distribución de las selecciones.

El comportamiento de RandomSampling evidentemente depende de la semilla que usemos para seleccionar de forma completamente aleatoria los clientes en cada ronda, esto puede influir mucho sobre todo sobre problemas como CIFAR-10 como ya hemos visto con GreedyFed. En la Figura 6.8 se muestran las curvas de aprendizaje en MNIST y CIFAR-10 de RandomSampling sobre 5 ejecuciones del algoritmo (i.e. 5 entrenamiento con semillas aleatorias diferentes), donde podemos ver que dependiendo de la semilla que usemos el modelo puede converger a ciertos valores que son mejores o peores que otros. Sin embargo, se puede apreciar, especialmente en CIFAR-10 (Figura 6.8(b)) que independientemente de la semilla, al tener un carácter aleatorio sobre datos y clientes heterogéneos hace que las curvas oscilen y produzcan sobreajuste del modelo de forma arbitraria.

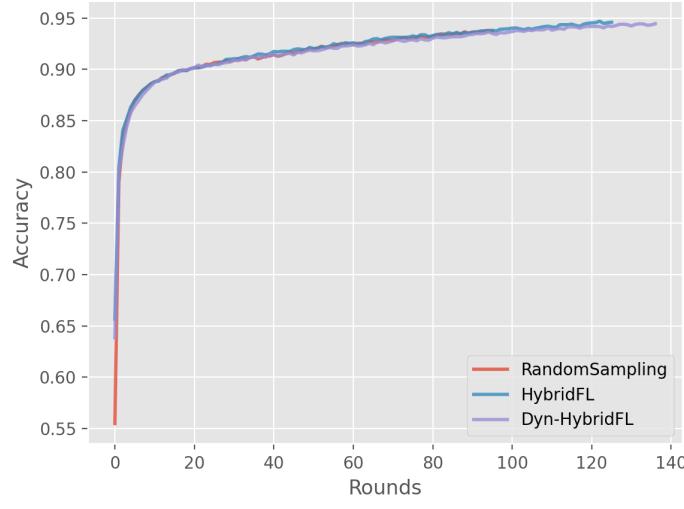
6.4. Experimentos de métodos basados en recursos

Para este análisis comparativo entre los métodos de selección basados en recursos de los clientes, ejecutaremos los métodos de RandomSampling, HybridFL y nuestra propuesta de mejora Dyn-HybridFL para los problemas de MNIST y CIFAR-10. Todos los algoritmos serán ejecutados para entrenar ambos conjuntos de datos sobre un tiempo total de entrenamiento $T_{\text{final}} = 400$ minutos, que son aproximadamente 6 horas. Este tiempo final de plazo de entrenamiento se ha tomado de la suite de configuración de HybridFL y FedCS ([58], [8]). Para los tres algoritmos y los dos problemas utilizaremos un presupuesto de selección de $M = \lceil K \times C \rceil$, donde C es la proporción de

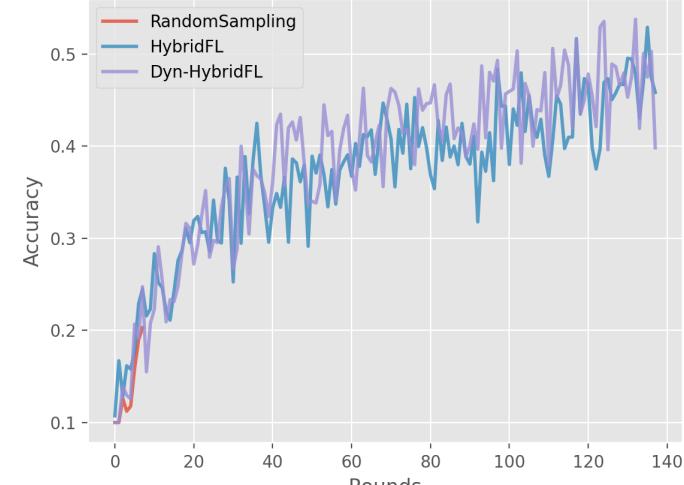
clientes a seleccionar (como máximo) por ronda; en nuestro caso utilizaremos un $C = 0.1$. El número de clientes total para MNIST y CIFAR-10 será de $K = 1000$ clientes (influenciado de igual manera por la configuración experimental de [58]), por tanto, tendremos un presupuesto de selección final de $M = 100$ clientes para ambos problemas.

Para la configuración específica de los métodos de selección, tenemos en primer lugar HybridFL que define el *deadline* de tiempo por ronda T_{round} , en cuyo caso utilizaremos el mismo valor que utilizaron sus autores para CIFAR-10 y MNIST $T_{\text{round}} = 3$ minutos. En el caso de Dyn-HybridFL, este parámetro es dinámico, sin embargo debe de especificarse un valor inicial del cual partir; para este elegimos un *deadline* inicial de $T_{\text{round}}^{(0)} = 3$ minutos al igual que HybridFL. En cuanto a la configuración de simulación de los recursos de los clientes debemos de definir dos parámetros: los recursos de comunicación medios θ_k^{avg} y los recursos de computación medios γ_k^{avg} , los cuales como hemos comentado en el capítulo 4, se miden en Mbit/s y datos procesados por segundo respectivamente; en nuestro caso tomaremos los valores de la configuración experimental de HybridFL [58] los cuales se ha obtenido a partir de simulaciones a tiempo de real de una plataforma MEC (más información de la obtención de estos resultados en [58]): $\theta_k^{\text{avg}} = 1.4$ Mbit/s y para el caso de la capacidad computacional de cada cliente se han determinado de forma aleatoria de una distribución uniforme $U(10, 100)$. Al igual que en [8], a partir de estos parámetros medios de cada cliente, podemos simular su “fluctuación de los recursos a corto plazo” a partir una distribución normal truncada que va de $(1 - r_{\text{var}})\theta_k^{\text{avg}}$ a $(1 + r_{\text{var}})\theta_k^{\text{avg}}$ para el caso de el ancho de banda medio (recursos de comunicación y de $(1 - r_{\text{var}})\gamma_k^{\text{avg}}$ a $(1 + r_{\text{var}})\gamma_k^{\text{avg}}$ en el caso de los recursos de computación; siendo r_{var} un parámetro de varianza u *holgura*; nosotros usaremos la misma varianza que en [58] $r_{\text{var}} = 0.2$. A partir de estos parámetros que representan los recursos de comunicación y computación de los clientes en cada ronda se puede determinar los tiempos estimados de actualización como $t_k^{\text{UD}} = \frac{E \times N_k}{\gamma_k^{(t)}}$, donde se calcula el tiempo que toma procesar el número total de datos a procesar en una ronda de entrenamiento (el número total de datos tantas veces las épocas a realizar), y los tiempos de subida estimados como $t_k^{\text{UL}} = \frac{D_m}{\theta_k^{(t)}}$, donde calculamos el tiempo que toma el subir un modelo de tamaño D_m (en Mbits). Finalmente, establecemos la fracción de clientes que permiten la subida de sus datos para ser entrenados de forma centralizada en el servidor como $r_{\text{UL}} = 0.01$ (estimando menos de un 1% de los clientes), los clientes que son considerados para subir sus datos K_{UL} son determinados aleatoriamente por la fase de *Resource Request* (Véase sección 4.5.3).

Los resultados de las ejecuciones así como las métricas obtenidas de estas se muestran en la Tabla 6.2 así como las curvas de aprendizaje en la *accuracy* del modelo de los métodos de selección en la Figura 6.9. A primera vista,



(a) MNIST



(b) CIFAR-10

Figura 6.9: Comparativa en la selección de los clientes por los métodos Random, HybridFL y Dyn-Hybrid para MNIST y CIFAR-10. Las gráficas de línea muestran la estimación de la función de densidad de los resultados de selección, o también denominado KDE (*Kernel Density Estimation*) [103].

Dataset	Algoritmo	ToA@0.85 (s)	ToA@0.9 (s)	Tiempo de ronda medio	Selecciones totales	Rondas Totales	Accuracy final
MNIST	RandomSampling	245.7	248.8	251.32 ± 6.49	9600	96	0.9370
	HybridFL	187.9	195.6	190.64 ± 5.86	9352	126	0.9459
	Dyn-HybridFL	168.5	180.03	175.96 ± 11.18	9532	137	0.9445
CIFAR10		ToA@0.3 (s)	ToA@0.5 (s)				
	RandomSampling	N/A	N/A	3145.22 ± 37.0	800	8	0.2028
	HybridFL	168.4	181.8	174.62 ± 10.76	607	138	0.4583
	Dyn-HybridFL	184.9	167.9	174.41 ± 11.28	602	138	0.3982

Tabla 6.2: Métricas de rendimiento obtenidas de los experimentos de los métodos de selección basados en recursos. Se marcan en negrita los que resultan vencedores en cuanto a la métrica se refiere. Los resultados que muestran “N/A” en algunas de las métricas indican que no se ha podido obtener cierta métrica en la ejecución del algoritmo.

podemos ver que en el caso de MNIST, nuestro algoritmo de Dyn-HybridFL obtiene un mayor número de rondas totales así como un mayor número de actualizaciones del modelo que su algoritmo base HybridFL. Además, en MNIST presenta una convergencia más rápida que este último fruto de tener un mayor número de actualizaciones del modelo. Por el otro lado, vemos que en CIFAR-10, sin embargo, ambos algoritmos presentan unas métricas muy similares entre sí, cosa que podemos visualizar en la Figura 6.9(b) donde aunque ambos puedan vencerse entre sí en cuanto a convergencia o *accuracy* final, podemos ver que realmente el entrenamiento en CIFAR-10 se puede considerar el mismo, además de solo diferenciarse en el número de actualizaciones en solo 5 actualizaciones del modelo más en HybridFL y Dyn-HybridFL, una diferencia despreciable entre ambos si tenemos en cuenta que en MNIST se distancian en 180 actualizaciones.

En cuanto a RandomSampling, podemos observar datos muy distintos y extremos en comparación a los otros dos. En primer lugar, vemos que en ambos casos (MNIST y CIFAR-10) obtiene un número de selecciones significativamente más alta. Esto debido a que selecciona *ingenuamente* los $[K \times C]$ clientes para entrenar localmente el modelo, esto hace que el número de actualizaciones por ronda aumente. Sin embargo, y como hemos previsto en la sección 4.5.3 cuando comentábamos la elección del T_{round} , cuando aumentamos $|S_t|$ en cada ronda, el número de rondas totales antes del plazo final T_{final} disminuye, en este caso RandomSampling obtiene un número significativamente menor que en los otros métodos; de hecho lo podemos observar en las Figuras 6.9 donde RandomSampling termina de forma muy temprana y con un rendimiento final de modelo subóptima.

En cuanto a los tiempos medios por cada ronda de entrenamiento, vemos que prácticamente Dyn-HybridFL es menor en todos los casos a HybridFL, aunque en CIFAR-10 éste último logra tener un comportamiento muy similar. También vemos como en ambos casos, la varianza en el tiempo de

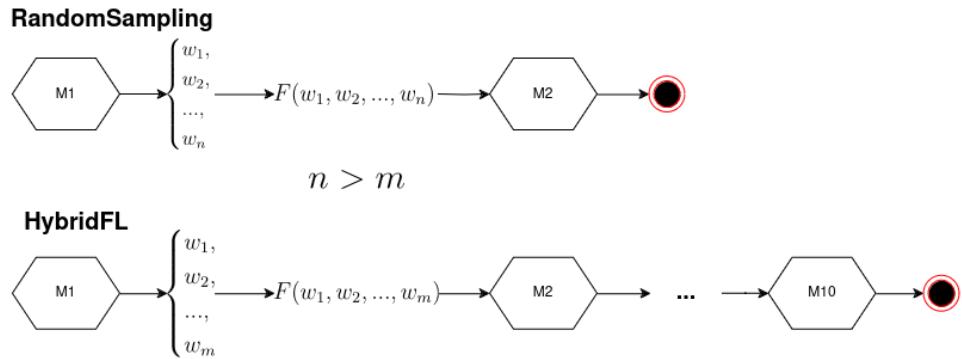


Figura 6.10: Diagrama de ejemplo comparativo entre RandomSampling y HybridFL en el número de actualizaciones y rondas que realizan. Este ejemplo es basado en los experimentos hechos anteriormente en 6.4 como diagrama ilustrativo.

ronda de Dyn-Hybrid es siempre mayor que HybridFL, gracias a que por su carácter dinámico en el *deadline* de tiempo de ronda, es esperable que varíe el tiempo de forma más exagerada. El que los tiempos de ronda sean menores en Dyn-HybridFL nos puede dar un indicio de que Dyn-HybridFL elija frecuentemente reducir el plazo de tiempo T_{round} por ronda de media, lo que podría dar indicios que los recursos de los clientes de media son peores que en la anterior ronda.

6.4.1. Análisis de los resultados

De los resultados vistos anteriormente nos surgen dos cuestiones que intentaremos resolver haciendo un análisis de estos resultados. El primero es relacionado al fenómeno que vemos con RandomSampling, en el que este algoritmo en todos los casos actualiza un mayor número de veces que los otros métodos con la diferencia de realizar pocas rondas totales de entrenamiento. Como hemos dicho en la definición de FedCS y HybridFL en la sección 4.5.3, la estrategia fundamental de estos métodos basados en FedCS, lo cual incluye HybridFL y Dyn-HybridFL, es maximizar el número de clientes seleccionados por ronda $|S_t|$ sujetos a restricciones de tiempo (Véase 4.7), esto para maximizar el número de actualizaciones que aplicamos al modelo, lo que resulta en un modelo con mejor rendimiento final ([4]). Sin embargo, esto no pasa en RandomSampling, que aún teniendo más actualizaciones obtiene un modelo final subóptimo. Podemos pensar que el problema viene de que RandomSampling, al tener menos rondas de entrenamiento totales, éste algoritmo agrega muchas más veces sobre las actualizaciones de un modelo, mientras que los otros métodos agregan menos veces pero sobre más modelos “parciales”. En la Figura 6.10 podemos ver un diagrama de ejemplo

de lo que tratamos de explicar, en este caso RandomSampling esta agregando un número menor de veces (menor número de rondas totales) sobre un número más alto de n modelos, mientras que HybridFL (aunque también puede ser Dyn-HybridFL) está realizando más agregaciones (10 agregaciones en este ejemplo) sobre un número menor de m actualizaciones. Entonces, de los resultados podemos ver que la aproximación de HybridFL, es decir, más agregaciones de un número menor de modelos, resulta en una mejor convergencia que la de RandomSampling donde hay menos agregaciones de muchos modelos.

La razón de que tener más agregaciones aún teniendo menos actualizaciones del modelo por cada agregación es por el hecho de que por *cada agregación, tenemos una diversificación más alta de clientes seleccionados*. Si vemos el protocolo de HybridFL en 6, vemos que por cada ronda se eligen M clientes nuevos y aleatorios, por tanto, si tenemos menos rondas totales como es el caso de RandomSampling, simplemente el algoritmo no es capaz de considerar un número considerablemente alto de clientes para abarcar todos los datos disponibles, empeorando la generalización. Esto lo podemos visualizar en las Figuras 6.11, donde en el primer caso 6.11(a) cuando tenemos menos rondas para muestrear M clientes vemos que se abarcan un menor número de clientes a cuando tenemos un número de rondas mayor, aún teniendo un M menor. Es por esto que, aunque HybridFL y Dyn-HybridFL tengan un número total de actualizaciones menor que RandomSampling, éstos tienen una capacidad de generalización mayor y por tanto, convergen de manera más rápida.

Finalmente, vamos a analizar el entrenamiento de HybridFL y Dyn-HybridFL para estudiar el cambio impuesto por Dyn-HybridFL como mejora a HybridFL. En primer lugar, hemos podido ver que el comportamiento es muy distinto si hablamos de MNIST y CIFAR-10, en el primero parece ser que Dyn-HybridFL si logra, en base a la gestión dinámica del T_{round} , realizar un número mayor de actualizaciones y rondas totales el cual era el objetivo de mejora, y en efecto resultó en una mejora notable a los resultados de HybridFL. Si vemos la Figura 6.12(a), para el caso de MNIST, podemos ver la evolución del número de clientes seleccionados en cada ronda de entrenamiento; podemos ver entonces que en el caso de HybridFL, donde tenemos un T_{round} estático, mantiene un cierto margen constante en el número de clientes seleccionados, que varían evidentemente en los recursos fluctuantes de los clientes preseleccionados de forma aleatoria. Por otro lado, vemos que en Dyn-HybridFL, al cambiar el T_{round} entre rondas podemos ver como el número de clientes seleccionados en S_t varían notablemente tanto para muy pocos como muchos clientes; de hecho, sorprendentemente parece que Dyn-HybridFL tiene una tendencia a tener un conjunto S_t más pequeño en comparación a HybridFL, esto explicaría que Dyn-HybridFL obtiene un número mayor de rondas totales, lo cual explicaría que los resultados en

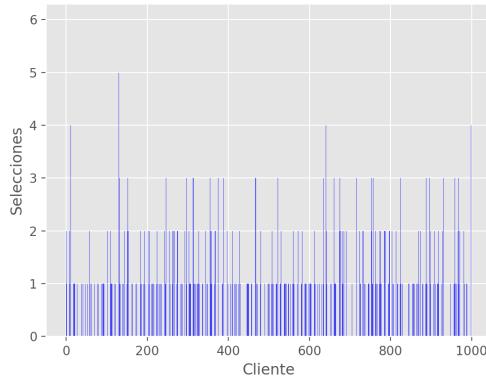
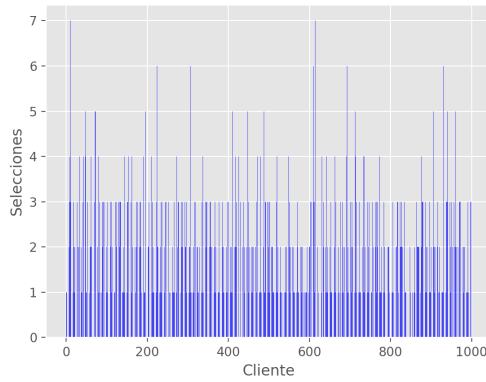
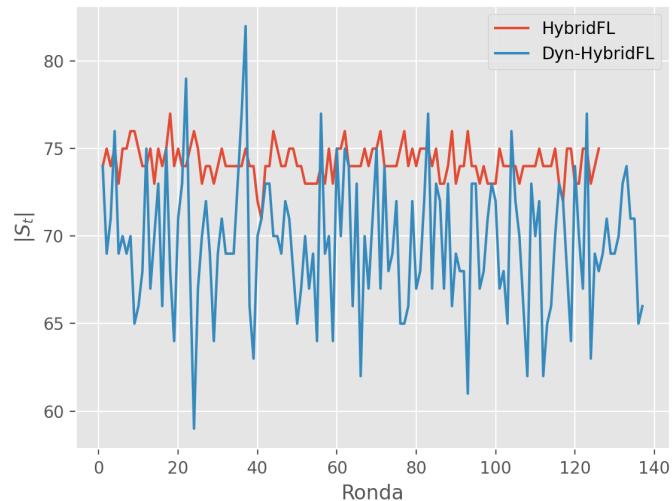
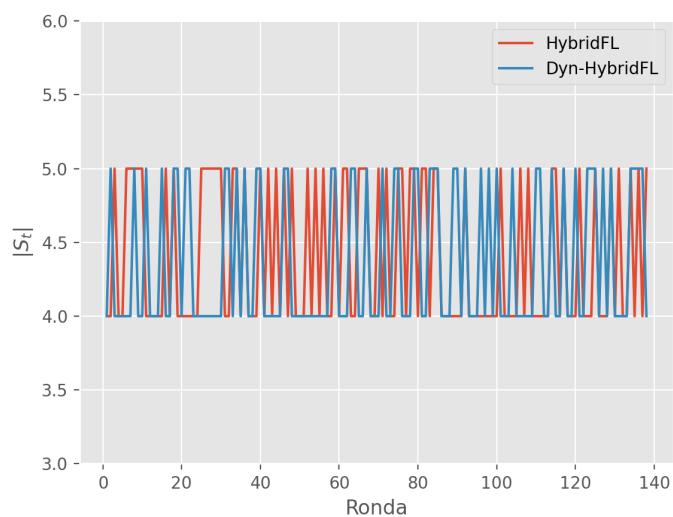
(a) $T = 8, M = 100$ (b) $T = 602, M = 3$

Figura 6.11: Comparativa en la cantidad total de clientes seleccionados aleatoriamente (simulando el paso de selección aleatoria inicial de M clientes de 6) para un número distinto de rondas totales. El objetivo es comparar la capacidad de generalización de RandomSampling (caso (a)) y HybridFL/Dyn-HybridFL (caso (b)). Para el caso (a) se han considerado todos los $M = 100$ clientes a muestrear por cada ronda mientras que en el caso (b) se ha considerado un número mucho menor de clientes ($M = 3$), de esta manera, visualizamos de una manera más fiel al lo que observamos en los resultados de los experimentos.



(a) MNIST



(b) CIFAR-10

Figura 6.12: Comparativa en el número de clientes seleccionados para cada ronda $|S_t|$ de HybridFL y Dyn-HybridFL.

convergencia y *accuracy* final de éste sean mejores que su algoritmo base gracias a lo que hemos explicado anteriormente con lo que pasaba con RandomSampling: **más rondas totales, significan más agregaciones; luego mejor convergencia.** No obstante, el número de actualizaciones totales no siempre va correlacionado con el número de rondas totales (como hemos comentado con RandomSampling); el porqué Dyn-HybridFL presenta un mayor número de actualizaciones viene dado, de nuevo, por la oscilación del número de clientes seleccionados, como vemos en la Figura 6.12(a), en efecto hay muchas rondas donde $|S_t|$ resulta más grande que en HybridFL, esto puede marcar la diferencia en el número total de actualizaciones de Dyn-HybridFL.

Hemos visto en efecto que en el caso de MNIST, Dyn-HybridFL presenta un comportamiento *esperable* a lo que el algoritmo fue diseñado, cumpliendo con su objetivo principal: *aumentar el número de rondas y actualizaciones totales*. Sin embargo, en el caso de CIFAR-10, no parece haber dado fruto esta mejora de Dyn-HybridFL, de hecho, presenta un comportamiento muy similar al algoritmo base HybridFL. Si observamos el número de seleccionados por ronda en la Figura 6.12(b), podemos ver exactamente el motivo de este fenómeno. En este caso, a diferencia de MNIST, $|S_t|$ en Dyn-HybridFL no parece oscilar con una gran varianza en sus valores, de hecho, se mantiene entre 4 y 5 clientes en todo momento al igual que lo hace HybridFL. Es por esto que consideramos que HybridFL y Dyn-HybridFL en CIFAR-10 son prácticamente iguales en rendimiento aunque HybridFL consiga unas 5 actualizaciones de más totales, ya que éstos dos métodos realizan prácticamente el mismo comportamiento de selección. La razón de que esto suceda es que simplemente el tiempo que involucra incluir o no un cliente en S_t en CIFAR-10 es muy *costoso*, sobretodo en lo que la subida del modelo se refiere. El tener un plazo de tiempo de $T_{\text{round}} = 3$ minutos en HybridFL y de base en Dyn-HybridFL (el *deadline* inicial $T_{\text{round}}^{(0)}$) hace que la selección sea muy restrictiva y debido a eso, los valores de $|S_t|$ por ronda presentan una varianza muy pequeña; el haber considerado un T_{round} mucho más grande pudo haber mitigado este comportamiento aunque hubiera empeorado el rendimiento final del modelo, ya que estamos minimizando el número total de rondas. No obstante, el caso de MNIST nos ha servido para poder comprobar que en efecto, nuestro algoritmo Dyn-HybridFL cumple su cometido con éxito en aumentar el número de rondas totales, que a su vez mejora el rendimiento del modelo final.

Capítulo 7

Conclusiones y Futuros Trabajos

En el Trabajo de Fin de Grado que hemos presentado tuvo como objetivo el análisis comparativo de distintas estrategias de selección de clientes en Aprendizaje Federado, concretamente hemos podido estudiar, implementar y analizar los métodos: RandomSampling, AFL, GreedyFed, HybridFL y Dyn-HybridFL, a los que hemos visto sus fortalezas y debilidades sobre dos problemas de Visión por Computador: MNIST y CIFAR-10. Hemos podido observar el comportamiento de cada algoritmo así como analizar el entrenamiento de éstos y estudiar el como hacen frente a las dificultades del Aprendizaje Federado como el entrenamiento sobre datos *no idéntica e independientemente distribuidos* (no-IID) o entrenamiento con clientes heterogéneos en cuanto a sus datos y recursos. A partir de un estudio experimental comprensivo hemos podido tener una percepción más clara y detallada de como estos métodos son capaces de mejorar la convergencia de un modelo de ML cuando el número de clientes es muy alto.

Otro de los objetivos de este TFG ha sido el diseño e implementación de un método de selección que mejore a alguno de los otros métodos en alguna de sus debilidades, que es el caso de nuestra propuesta de mejora Dyn-HybridFL, que hemos demostrado de forma experimental que cumple con su objetivo de mejorar la convergencia de su algoritmo base HybridFL con la gestión del parámetro T_{round} de forma dinámica.

A partir de los resultados que hemos podido analizar, hemos sacado las siguientes conclusiones:

- **Selección diversificada de los clientes.** En los resultados experimentales tanto de los métodos basados en rendimiento como de los basados en recursos, hemos podido observar un patrón muy claro: el modelo converge significativamente más rápido cuando abarca un

número de clientes más amplio. Un caso muy claro es cuando veíamos que en GreedyFed la selección tanto en MNIST como en CIFAR-10 estaba muy sesgada hacia clientes que tenían mayores valores de Shapley (Figura 6.4), esto hacía que en ambos casos GreedyFed terminaría presentando un sobreajuste del modelo. También lo podíamos ver en RandomSampling en métodos basados en recursos, donde aún teniendo más actualizaciones que HybridFL y Dyn-HybridFL, seguía teniendo un rendimiento subóptimo gracias a que al tener pocas rondas totales de entrenamiento no era capaz de seleccionar clientes nuevos (Figura 6.11). El caso contrario lo podíamos ver en AFL, donde este algoritmo promovía el entrenamiento sobre clientes seleccionados de forma aleatoria a la vez que realizaba una selección basada en valuaciones de los clientes, lo cual resultaba en un claro vencedor en los métodos basados en rendimiento.

- **Gestión dinámica del T_{round} en Dyn-HybridFL.** Uno de los hallazgos más importantes en los experimentos de los métodos basados en recursos es la gestión dinámica del parámetro T_{round} en Dyn-HybridFL, y que es estático en HybridFL. En los resultados hemos podido comprobar su eficacia en MNIST (Tabla 6.2, donde efectivamente aumentaba el número de rondas totales de entrenamiento reduciendo de forma general el T_{round} a valores menores (véase Figura 6.12(a)). Con CIFAR-10 podíamos además ver que aún con la gestión dinámica de este parámetro, Dyn-HybridFL no lograba reducir o aumentar el baremo del *deadline* ya que el coste de incluir o no un cliente era mayor que el cambio en los tiempos estimados medios de los clientes.
 - Nosotros hemos usado una representación para los recursos medios de los clientes K' , que hemos denominado $\phi^{(t)}$ (4.13), que calcula el tiempo estimado medio de comunicación y computación de los clientes y los sumaba para obtener un estimado del tiempo de ronda, del cual se obtenía el nuevo T_{round} ; sin embargo, una posible vía de investigación futura es el estudio de diferentes representaciones de los recursos de los clientes, de manera que el cambio de un T_{round} a otro sea más preciso y por tanto más inteligente. Del mismo modo, estudiar diferentes maneras de cambiar este T_{round} ; nosotros hemos usado un cambio basado en la relación entre los recursos medios de la ronda anterior y la actual, sin embargo, diferentes formas de calcular este parámetro así como imponer límites inferiores y superiores, pueden hacer que el algoritmo de selección de clientes sea más robusto.
- **Viabilidad práctica de los algoritmos de selección.** Con los resultados experimentales pudimos ver además los tiempos medios de cada ronda, que podrían proporcionar información acerca del balance

rendimiento/tiempo de los algoritmos de selección. El caso más notorio es el de GreedyFed, que llega a más de 1000 minutos cuando no converge el algoritmo de aproximación GtG-Shapley en su cálculo de valores de Shapley y con una media de 700 minutos (Tabla 6.1). Como podemos observar, GreedyFed al tener una complejidad computacional muy alta en el cálculo de las valuaciones de los clientes, esto hace que el entrenamiento sea inviable en tiempo y más aún cuando incrementamos el número de clientes por ronda M , que como hemos visto en la Figura 6.3(b), esto hace que GreedyFed sobreajuste el modelo y sea necesario aumentar M para contrarrestar es *overfitting* lo que hará que el entrenamiento sea menos tratable en tiempo de ejecución.

- Hoy en día, el cálculo de los valores de Shapley son un problema de investigación abierto en cuanto a la manera más eficiente de aproximar estos valores [99], el trabajo futuro sobre el cálculo de estos algoritmos podrían hacer que métodos como GreedyFed puedan evitar el *overfitting* de los modelos aumentando el número de clientes por ronda y al mismo tiempo no sacrificar un tiempo de cómputo significante.

En resumen, este TFG nos ha servido para estudiar la importancia de la selección de clientes en Aprendizaje Federado en aplicaciones reales donde tenemos un gran número de clientes y necesitamos realizar una selección inteligente que permita converger a un modelo con buen rendimiento y que sea eficiente en los recursos de los clientes. Los resultados experimentales llevados a cabo en este trabajo sirven de base sólida para trabajos futuros relacionados con la selección de clientes en Aprendizaje Federado.

Capítulo 8

Bibliografía

- [1] D. Naik y N. Naik en Advances in Computational Intelligence Systems, (eds.: N. Naik, P. Jenkins, P. Grace, L. Yang y S. Prajapat), Springer Nature Switzerland, Cham, **2024**, págs. 3-17.
- [2] D. Naik y N. Naik en Advances in Computational Intelligence Systems, (eds.: N. Naik, P. Jenkins, P. Grace, L. Yang y S. Prajapat), Springer Nature Switzerland, Cham, **2024**, págs. 18-28.
- [3] M. Abadi, A. Chu, I. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar y L. Zhang en Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, ACM, **2016**.
- [4] H. B. McMahan, E. Moore, D. Ramage, S. Hampson y B. A. y Arcas, Communication-Efficient Learning of Deep Networks from Decentralized Data, **2023**.
- [5] C. Dwork y A. Roth, **2014**.
- [6] J. Verbraeken, M. Wolting, J. Katzy, J. Kloppenburg, T. Verbelen y J. S. Rellermeyer, *CoRR* **2019**, *abs/1912.09789*.
- [7] G. D. Németh, M. Ángel Lozano, N. Quadrianto y N. Oliver, A Snapshot of the Frontiers of Client Selection in Federated Learning, **2023**.
- [8] T. Nishio y R. Yonetani en ICC 2019 - 2019 IEEE International Conference on Communications (ICC), IEEE, **2019**.
- [9] H. Yang, W. Xi, Z. Wang, Y. Shen, X. Ji, C. Sun y J. Zhao, *Information Sciences* **2023**, 645, 119360.
- [10] A. Ignatov, R. Timofte, W. Chou, K. Wang, M. Wu, T. Hartley y L. Van Gool en Proceedings of the European Conference on Computer Vision (ECCV) Workshops, **2018**.
- [11] E. Commission, *ECTS Users' Guide 2015*, Publications Office of the European Union, Luxembourg, **2015**.

- [12] Z. Cataldi, F. Lage, R. Pessacq y R. García Martínez, Ingeniería de software educativo, Archivado el 29 de diciembre de 2013 en Wayback Machine, **2013**.
- [13] E. R. Marsh, *The Academy of Management Journal* **1975**, *18*, 358-364.
- [14] Manfred, Guía Salarial 2024 - Salarios en tecnología [España], Revisado en julio de 2024, **2024**.
- [15] G. Research, Google Colaboratory, Revisado en julio de 2024, **2024**.
- [16] J. D. Team, Project Jupyter: Open source tools for interactive data science and scientific computing, Revisado en julio de 2024, **2024**.
- [17] B. S. on StackOverflow, Backend specs example, Revisado en julio de 2024, **2024**.
- [18] E. D. D. London en On the diagrammatic and mechanical representation of propositions and reasonings, **1880**.
- [19] C. staff, *Coursera* **2023**, Revisado en: 2024-07-30.
- [20] G. C. staff, *Google Cloud* **2023**, Revisado en: 2024-07-30.
- [21] U. B. S. of Information, *UC Berkeley School of Information* **2020**, Revisado en: 2024-07-30.
- [22] DataCamp, *DataCamp Blog* **2023**, Revisado en: 2024-07-30.
- [23] Y. S. Abu-Mostafa, M. Magdon-Ismail y H.-T. Lin, *Learning from Data*, Revisado en: 2024-07-31, AMLBook, **2012**.
- [24] S. Jyothula, *IOP Conference Series: Materials Science and Engineering* **2021**, *1099*, 012028.
- [25] S. H. Do, K. D. Song y J. W. Chung, *Korean Journal of Radiology* **2020**, *21*, 33-41.
- [26] D. P. Kingma y J. Ba, Adam: A Method for Stochastic Optimization, **2017**.
- [27] J. Brownlee, Rectified Linear Activation Function for Deep Learning Neural Networks, <https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/>, Revisado en: 2024-07-31, **2023**.
- [28] Aprende IA, ¿Qué son las Redes Neuronales Artificiales?, <https://aprendeia.com/que-son-las-redes-neuronales-artificiales/>, Revisado en: 2024-07-31, **2023**.
- [29] N. J. Nilsson, *The Quest for Artificial Intelligence*, Cambridge University Press, **2009**.
- [30] Y. LeCun, Y. Bengio y G. Hinton, *Nature* **2015**, *521*, hal-04206682, 436-444.

- [31] Glosser.ca, CC BY-SA 3.0, <https://creativecommons.org/licenses/by-sa/3.0>, via Wikimedia Commons, **2023**.
- [32] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard y L. D. Jackel, *Neural Computation* **1989**, *1*, 541-551.
- [33] D. E. Rumelhart, G. E. Hinton y R. J. Williams, *Nature* **1986**, *323*, 533-536.
- [34] Y. Lecun, L. Bottou, Y. Bengio y P. Haffner, *Proceedings of the IEEE* **1998**, *86*, 2278-2324.
- [35] T. D. Science, Simple Introduction to Convolutional Neural Networks, <https://towardsdatascience.com/simple-introduction-to-convolutional-neural-networks-cdf8d3077bac>, Revisado en: 2024-07-31, **2023**.
- [36] I. Sobel, *Presentation at Stanford A.I. Project 1968* **2014**.
- [37] S. contributor, Valve Sobel, <https://creativecommons.org/licenses/by-sa/3.0>, via Wikimedia Commons.
- [38] MarcT0K, Centralized Federated Learning protocol, via Wikimedia Commons, **2024**.
- [39] J. Konečný, B. McMahan y D. Ramage, Federated Optimization:Distributed Optimization Beyond the Datacenter, **2015**.
- [40] A. Gouissem, Z. Chkirkene y R. Hamila, A Comprehensive Survey On Client Selections in Federated Learning, **2023**.
- [41] J. Konečný, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh y D. Bacon, Federated Learning: Strategies for Improving Communication Efficiency, **2017**.
- [42] S. Wang, T. Tuor, T. Salonidis, K. K. Leung, C. Makaya, T. He y K. Chan en IEEE INFOCOM 2018 - IEEE Conference on Computer Communications, **2018**, págs. 63-71.
- [43] P. Kairouz, H. B. McMahan, B. Avent, A. Bellet, M. Bennis, A. N. Bhagoji, K. Bonawitz, Z. Charles, G. Cormode, R. Cummings, R. G. L. D’Oliveira, H. Eichner, S. E. Rouayheb, D. Evans, J. Gardner, Z. Garrett, A. Gascón, B. Ghazi, P. B. Gibbons, M. Gruteser, Z. Harchaoui, C. He, L. He, Z. Huo, B. Hutchinson, J. Hsu, M. Jaggi, T. Javidi, G. Joshi, M. Khodak, J. Konečný, A. Korolova, F. Koushanfar, S. Koyejo, T. Lepoint, Y. Liu, P. Mittal, M. Mohri, R. Nock, A. Özgür, R. Pagh, M. Raykova, H. Qi, D. Ramage, R. Raskar, D. Song, W. Song, S. U. Stich, Z. Sun, A. T. Suresh, F. Tramèr, P. Vepakomma, J. Wang, L. Xiong, Z. Xu, Q. Yang, F. X. Yu, H. Yu y S. Zhao, Advances and Open Problems in Federated Learning, **2021**.
- [44] L. Xia, P. Zheng, J. Li, W. Tang y Z. Xiangying, *IET Collaborative Intelligent Manufacturing* **2022**, *4*, n/a-n/a.

- [45] Y. Liu, Y. Kang, T. Zou, Y. Pu, Y. He, X. Ye, Y. Ouyang, Y.-Q. Zhang y Q. Yang, *IEEE Transactions on Knowledge and Data Engineering* **2024**, *36*, 3615–3634.
- [46] Y. Shi, H. Yu y C. Leung, *IEEE Transactions on Neural Networks and Learning Systems* **2024**, 1–17.
- [47] S. R. Pandey, V. P. Bui y P. Popovski, Goal-Oriented Communications in Federated Learning via Feedback on Risk-Averse Participation, **2023**.
- [48] Y. J. Cho, J. Wang y G. Joshi, Client Selection in Federated Learning: Convergence Analysis and Power-of-Choice Selection Strategies, **2020**.
- [49] T. Chen, G. B. Giannakis, T. Sun y W. Yin, LAG: Lazily Aggregated Gradient for Communication-Efficient Distributed Learning, **2018**.
- [50] S. Caldas, J. Konečny, H. B. McMahan y A. Talwalkar, Expanding the Reach of Federated Learning by Reducing Client Resource Requirements, **2019**.
- [51] P. Zhou, P. Fang y P. Hui, Loss Tolerant Federated Learning, **2021**.
- [52] L. S. Shapley, *International Journal of Game Theory* **1971**, DOI 10.1007/BF01753431.
- [53] Y. Sarikaya y O. Ercetin, Motivating Workers in Federated Learning: A Stackelberg Game Perspective, **2019**.
- [54] F. Lai, X. Zhu, H. V. Madhyastha y M. Chowdhury, Oort: Efficient Federated Learning via Guided Participant Selection, **2021**.
- [55] J. Kang, Z. Xiong, D. T. Niyato, S. Xie y J. Zhang, *IEEE Internet of Things Journal* **2019**, *6*, 10700-10714.
- [56] P. Blanchard, E. M. El Mhamdi, R. Guerraoui y J. Stainer en Advances in Neural Information Processing Systems, vol. 30, (eds.: I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan y R. Garnett), Curran Associates, Inc., **2017**.
- [57] J. Goetz, K. Malik, D. Bui, S. Moon, H. Liu y A. Kumar, Active Federated Learning, **2019**.
- [58] N. Yoshida, T. Nishio, M. Morikura, K. Yamamoto y R. Yonetani, Hybrid-FL for Wireless Networks: Cooperative Learning Mechanism Using Non-IID Data, **2020**.
- [59] L. Nagalapatti y R. Narayananam, *ArXiv* **2021**, *abs/2110.12257*.
- [60] P. Singhal, S. R. Pandey y P. Popovski, *IEEE Networking Letters* **2024**, *6*, 134–138.
- [61] F. Etro, *Journal of Economics* **2013**, *109*, 89-92.

- [62] H. Wang, Z. Kaplan, D. Niu y B. Li, *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications* **2020**, 1698-1707.
- [63] A. Gouissem, Z. Chkirkbene y R. Hamila, A Comprehensive Survey On Client Selections in Federated Learning, **2023**.
- [64] L. U. Khan, S. R. Pandey, N. H. Tran, W. Saad, Z. Han, M. N. H. Nguyen y C. S. Hong, Federated Learning for Edge Networks: Resource Optimization and Incentive Mechanism, **2020**.
- [65] E. Diao, J. Ding y V. Tarokh, HeteroFL: Computation and Communication Efficient Federated Learning for Heterogeneous Clients, **2021**.
- [66] D. Zeng, S. Liang, X. Hu, H. Wang y Z. Xu, FedLab: A Flexible Federated Learning Framework, **2022**.
- [67] F. Herrera, D. Jiménez-López, A. Argente-Garrido, N. Rodríguez-Barroso, C. Zuheros, I. Aguilera-Martos, B. Bello, M. García-Márquez y M. V. Luzón, FLEX: FLEXible Federated Learning Framework, **2024**.
- [68] U. d. C. Universidad de Granada, Universidad de Jaén, Dasci - Data Science and Computational Intelligence Research Center, <https://dasci.es/es/>, Revisado en: 2024-08-13.
- [69] Suvanjanprasai, MNIST Examples, CC BY-SA 4.0, via Wikimedia Commons, **2024**.
- [70] Y. LeCun, L. Bottou, Y. Bengio y P. Haffner, *Proceedings of the IEEE* **1998**, 86, 2278-2324.
- [71] C. J. B. Yann LeCun, Corinha Cortes, THE MNIST DATABASE of handwritten digits, <https://yann.lecun.com/exdb/mnist/>, Revisado en: 2024-08-13.
- [72] G. Cohen, S. Afshar, J. Tapson y A. van Schaik, EMNIST: an extension of MNIST to handwritten letters, **2017**.
- [73] V. N. Alex Krizhevsky y G. Hinton, The CIFAR-10 dataset, <https://www.cs.toronto.edu/~kriz/cifar.html>, Revisado en: 2024-08-13.
- [74] A. Krizhevsky, *University of Toronto* **2009**.
- [75] D. Macêdo, Correct Normalization Values for CIFAR-10, <https://github.com/kuangliu/pytorch-cifar/issues/19>, Revisado en: 2024-08-13.
- [76] J. Willaert, How To Calculate the Mean and Standard Deviation — Normalizing Datasets in Pytorch, <https://towardsdatascience.com/how-to-calculate-the-mean-and-standard-deviation-normalizing-datasets-in-pytorch-704bd7d05f4c>, Revisado en: 2024-08-13.

- [77] N. Dalal y B. Triggs en IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR), IEEE, **2005**, págs. 886-893.
- [78] D. G. Lowe, *International journal of computer vision* **2004**, *60*, 91-110.
- [79] R. M. Haralick, K. Shanmugam e I. Dinstein, *IEEE Transactions on Systems Man and Cybernetics* **1973**, *SMC-3*, 610-621.
- [80] Z. Zdziarski, The Reasons Behind the Recent Growth of Computer Vision, <https://zbigatron.com/the-reasons-behind-the-recent-growth-of-computer-vision/>, Revisado en: 2024-08-14, **2018**.
- [81] J. Brownlee, Rectified Linear Activation Function for Deep Learning Neural Networks, <https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/>, Revisado en: 2024-08-14, **2020**.
- [82] I. Goodfellow, Y. Bengio y A. Courville, *Deep Learning*, <http://www.deeplearningbook.org>, MIT Press, **2016**.
- [83] T. Yep, torch-summary: Model summary in PyTorch similar to ‘model.summary()’ in Keras, <https://pypi.org/project/torch-summary/>, Version 1.4.5, **2019**.
- [84] R. Kadam, What is Max Pooling in CNN? Is it Useful to Use?, <https://medium.com/codex/what-is-max-pooling-in-cnn-is-it-useful-to-use-6f2d6ff44c6>, Revisado en: 2024-08-14, **2021**.
- [85] K. He, X. Zhang, S. Ren y J. Sun, Deep Residual Learning for Image Recognition, **2015**.
- [86] A. Mao, M. Mohri e Y. Zhong, Cross-Entropy Loss Functions: Theoretical Analysis and Applications, **2023**.
- [87] L. E. Bottou en **1998**.
- [88] S. Ruder, An overview of gradient descent optimization algorithms, **2017**.
- [89] S. J. Reddi, S. Kale y S. Kumar, On the Convergence of Adam and Beyond, **2019**.
- [90] Y. Zhang, C. Chen, N. Shi, R. Sun y Z.-Q. Luo, Adam Can Converge Without Any Modification On Update Rules, **2023**.
- [91] G. van Rossum, Python Programming Language, <https://www.python.org/>, Para este trabajo se ha utilizado Python en su versión 3.10, **1991**.
- [92] J. Gruber, Markdown Syntax, <https://daringfireball.net/projects/markdown/syntax>, Revisado en: 2024-08-14, **2013**.
- [93] W. McKinney, Pandas data analysis tool, <https://pandas.pydata.org/>, Revisado en: 2024-08-14, **2008**.

- [94] T. Oliphant, Numerical Python, <https://numpy.org/>, Revisado en: 2024-08-14, **2006**.
- [95] J. D. Hunter, Matplotlib, <https://matplotlib.org/>, Revisado en: 2024-08-14, **2007**.
- [96] N. Ketkar en *Deep Learning with Python: A Hands-on Introduction*, Apress, Berkeley, CA, **2017**, págs. 195-208.
- [97] B. Settles en **2009**.
- [98] P. J. Acklam, Exp.svg, <http://creativecommons.org/licenses/by-sa/3.0/>, CC BY-SA 3.0, via Wikimedia Commons, **2007**.
- [99] Z. Liu, Y. Chen, H. Yu, Y. Liu y L. Cui, GTG-Shapley: Efficient and Accurate Participant Contribution Evaluation in Federated Learning, **2021**.
- [100] H. Zhao, S. Zhang y E. Garcia-Palacios, *China Communications* **2016**, 13, 89-99.
- [101] P. Liashchynskyi y P. Liashchynskyi, Grid Search, Random Search, Genetic Algorithm: A Big Comparison for NAS, **2019**.
- [102] A. Shamsian, A. Navon, E. Fetaya y G. Chechik, Personalized Federated Learning using Hypernetworks, **2021**.
- [103] M. Rosenblatt, *The Annals of Mathematical Statistics* **1956**, 27, 832 -837.

