INTERNATIONAL DOCTORAL SCHOOL OF THE USC

Marcos
Fernández Criado

PhD Thesis

Federated Learning over sets of heterogeneous devices

Santiago de Compostela, 2025

Doctoral Programme in Information Technology Research

ESCOLA DE DOUTORAMENTO
INTERNACIONAL DA USC

TESE DE DOUTORAMENTO

Federated Learning over sets of heterogeneous devices

Autor
Marcos Fernández Criado

Director: Manuel Ladra González

Titor: Tomás Fernández Pena

PROGRAMA DE DOUTORAMENTO EN INVESTIGACIÓN EN TECNOLOXÍAS DA INFORMACIÓN

SANTIAGO DE COMPOSTELA, 2025

The results presented in this thesis were obtained thanks to a grant from the Xunta de Galicia – Consellería de Cultura, Educación e Universidade, grant ED481A 2021/060, the project grant PID2020-115155GB-I00 from the Agencia Estatal de Investigación (European FEDER support included, UE), and also by the Consellería de Cultura, Educación e Universidade da Xunta de Galicia, through the Competitive Reference Groups (GRC), ED431C 2023/31(European FEDER support included, UE).

The author declares that he has no conflicts of interest related to this doctoral thesis.

Acknowledgements

I would like to acknowledge my unquestionable indebtedness and express my most sincere gratitude to my advisor, Professor Manuel Ladra, for the fundamental role and invaluable support of my PhD study and research, for his dedication, patience, enthusiasm, and immense knowledge. His guidance helped me in very crucial times of research and during the writing of this thesis.

I owe a lot to my family, who encouraged and helped me at every stage of my personal and academic life, and longed to see this achievement come true. Very warm thanks go to my mother and my aunt for their support and understanding.

I also wanted to thank some very special friends that deserve the recognition of having supported me in different moments of this process. To Diego and Nuria, who were the best flatmates I could have wished for, and transformed the beginning of this path into an unforgettable collection of memories. To Sandra and Antía, for those unending conversations over an iced coffee that gave me the strength and the laughs I needed to fulfill this work. To my whole group of friends from Ourense, who heard tirelessly all of my complaints throughout these years and reassured me when I needed it more than anything. To Fran, César, and Sabela, who shared this experience with me, and have been the rock I stood by in so many different occasions. I could not have asked for better workmates than you. Also, to Fernando, for being the guidance I needed, and act as an advisor when it seemed I did not have any.

Contents

Abstract

The objective of this PhD thesis is to face the issue of data heterogeneity when training a Federated Learning model across several devices, or participants.

There exist lot of previous approaches that aim to solve this kind of situation. In contrast with them, this dissertation starts with a bird's eye view of the whole data heterogeneity casuistry. Performing this analysis, data heterogeneity can be rigorously classified, making specific problems more approachable.

Two main branches of problems are identified: the ones caused purely by data skewness across the devices, and the ones were data evolves over time. At the same time, each of these two branches can be further split according to the kind of data probability that is affected. This split is based on mathematical properties of the data probability.

After classifying the data heterogeneity in such a way, two different algorithms are proposed, each of them dedicated to improve previous state-of-the-art results in one of the main branches mentioned before. On the one hand, we present an algorithm designed for handling time-evolving sets of data. On the other hand, we display an algorithm that focuses on identifying conflicting data distributions and manages to reach an FL model that obtains groundbreaking results in this kind of situations.

Resumo en Galego

Nas últimas décadas produciuse unha revolución constante a nivel tecnolóxico na nosa sociedade. A aparición de múltiples dispositivos intelixentes xerou grandes volumes de datos que cómpre procesar sen vulnerar a privacidade. Non estamos a falar soamente dos móbiles, esta revolución inclúe tamén o nacemento dos robots de servizo, ou calquera outro tipo de dispositivo englobado dentro do termo de Internet das Cousas (IoT). Son, en xeral, aparellos de todo tipo equipados con tecnoloxía punta para ofrecer, cada un no seu ámbito particular, o servizo máis eficiente posible. Todos estes trebellos recollen información do seu entorno, cada vez con máis precisión e cada vez máis en tempo real, abrindo unha infinidade de posibilidades de aplicación, en campos tan diversos como a saúde, o deporte, a banca, a educación, as interaccións persoais, etc.

A aprendizaxe automática (ML) é unha rama de investigación que busca desenvolver algoritmos que permitan a todas estas máquinas aprender a desenvolverse en diversas situacións por si mesmas. Dada a crecente cantidade de datos que todos estes aparellos xeran, o procesamento de toda esta información resulta complexa e custosa, e por iso comezaron a desenvolverse alternativas de ML distribuídas, nas que moitos dispositivos captan datos e envíanos a un servidor central, moito máis potente, no que eses datos se procesan de forma máis veloz e eficiente.

A centralización da información, sen embargo, leva ao mesmo tempo unha serie de riscos. A información que cada unha destas máquina recolle é enviada a un servidor central, onde os datos de todos os dispositivos semellantes agréganse. Esta compartición de información provoca riscos para a privacidade dos datos. Ademais disto, o servidor central neste tipo de algoritmos encárgase tanto de almacenar a información como de procesala, e coa inxente cantidade de datos e os numerosos dispositivos que poden enviarlle información, isto pode supoñer unha carga demasiado custosa de manexar.

Por todo isto, nos últimos anos cobrou forza unha nova tendencia dentro da aprendizaxe automática distribuída denominada aprendizaxe federada (FL), na cal os datos recollidos por cada dispositivos procésanse dentro do propio aparello, e o modelo xerado por este e posteriormente enviado ao servidor central, que se encarga de agregar todos os modelos e producir un modelo global axuntando o coñecemento adquirido por todos os participantes. Deste xeito, a carga do servidor central vese reducida enormemente, e os datos mantéñense en cada dispositivo, privados.

Con todo o anterior, a FL permite o adestramento local de modelos mantendo os datos privados, e ditos modelos son posteriormente combinados nun servidor central, para xerar un modelo final que poida ser empregado por todos os dispositivos que participaron no adestramento. Os dous retos principais que teñen que afrontar os algoritmos de FL, de forma xeral, son os seguintes.

En primeiro lugar, a heteroxeneidade dos datos, denominada na literatura como o problema dos datos non-IID (non independentes nin identicamente distribuídos). Esta característica dos datos prodúcese porque cada dispositivo recolle os datos nunhas condicións e nun contexto específico, e polo tanto cada un posúe uns rumbos propios, e distintivos do resto.

Definición 4.2.1. *Dise que un conxunto de datos é IID se a probabilidade dunha mostra mantense invariante a medida que se obteñen outras mostras (é dicir, son independentes), e se ademais, escollida unha mostra arbitrariamente, dita mostra ten a mesma probabilidade de pertencer a calquera dos usuarios que participan no proceso de adestramento. En termos matemáticos, se consideramos a unión dos dataset locais dos usuarios* $D = \bigcup_{i \in \mathcal{N}} D_i \subset \mathcal{X} \times \mathcal{Y}$*, entón* D *é un conxunto de datos IID no contexto da FL se, e só se,*

$$\begin{cases} P_i((x,y),(x',y')) = P_i(x,y) \cdot P_i(x',y'), \\ D_i(x,y) = D_j(x,y), \end{cases}$$

para todo $i, j \in \mathcal{N}$*, e para todo* $(x,y),(x',y') \in \mathcal{X} \times \mathcal{Y}$.

En segundo lugar, o outro gran reto é a non estacionaridade dos datos, é dicir, cambios na distribución dos datos ao longo do tempo. Isto provoca que as predicións obtidas cun modelo queden obsoletas, e sexa necesario adaptar o modelo co paso do tempo, afrontando o que se coñece como Deriva de Concepto, ou *Concept Drift*, e pode definirse máis formalmente do seguinte xeito:

Definición 4.3.1 (Concept drift). *Sexa $S^{[0,t]} = \{(x_i, y_i)\}_{i=1}^{M}$ unha secuencia de datos de tamaño M recollida durante un período de tempo $[0, t]$. x_i representa o vector de características da mostra e y_i a súa correspondente etiqueta. Sexa $\mathcal{D}^t(x, y)$ a distribución de probabilidade da secuencia de datos $S^{[0,t]}$.*

Dise que unha deriva de concepto prodúcese no instante t se a distribución de probabilidade da secuencia de datos é significativamente distinta despois dese intre:

$$\exists\, t > 1 \,:\, \mathcal{D}^{t-1}(x, y) \nsim \mathcal{D}^{t}(x, y).$$

Estes dous retos pódense afrontar dende diferentes puntos de vista, e con estratexias moi diversas. Ambos retos poden interpretarse como un problema de heteroxeneidade dos datos, ben sexa no eixo espacial (entre os diferentes participantes do proceso de adestramento) ou no eixo temporal (cando a distribución de datos muda co paso do tempo).

Ao longo das últimas décadas, estes teñen sido problemas moi coñecidos e afrontados por diferentes investigadores dende diferentes ángulos. Por iso, é importante ter unha visión global de todas as estratexias que xa foron estudadas no seu momento. As solucións ao problema da heteroxeneidade propóñense dende distintos enfoques, sendo un dos máis salientables a personalización dos modelos por parte de cada usuario. Esta variante permite que, unha vez alcanzado o modelo global, cada un dos usuarios continúe co adestramento dese modelo de forma local. Isto permite que cada dispositivo teña un modelo adaptado á súa propia distribución de datos, mellorando a precisión local sen prexudicar o modelo global.

Todos os problemas datos non-IID, tanto no eixo espacial coma no temporal, pódense clasificar segundo a densidade de probabilidade que se vexa afectada en cada caso. Escribindo a probabilidade conxunta das mostras como $P(x, y) = P(x) \cdot P(y|x)$, pódese factorizar a probabilidade dunha mostra como o produto de dúas probabilidades, a probabilidade das características dunha mostra e a probabilidade da etiqueta correspondente a ditas características. Deste xeito, obtéñense 3 escenarios non-IID para cada eixo:

- Hai diferenzas unicamente no termo $P(x)$, que será o que referiremos como cambios no entorno dos usuarios no eixo espacial, ou deriva virtual de concepto no eixo temporal.
- Hai diferenzas unicamente no termo $P(y|x)$, que será o que referiremos como cambios no comportamento dos usuarios no eixo espacial, ou deriva real de concepto no eixo temporal.

- Hai diferenzas en ambos termos, o que denominaremos como deriva total de concepto no eixo temporal.

Cada unha destas probabilidades pode verse afectada de diferentes maneiras. Pódese establecer unha taxonomía estatística para organizar estas estratexias segundo os tipos de diferenzas entre as distribucións de datos, ben sexan diferenzas nas características, nas etiquetas ou na frecuencia das clases. Existen estratexias de aprendizaxe que tentan emendar os problemas ocasionados por todos este tipos de heteroxeneidade, pero na gran maioría dos casos estas estratexias resultan pouco eficientes, e non obteñen resultados comparables aos que se obterían con problemas nos que os datos si son IID. O obxectivo principal da FL no contexto dos datos non-IID é mellorar a calidade das predicións obtidas, alcanzando os mesmos valores de precisión ca no paradigma IID.

Do mesmo xeito, os métodos empregados en aprendizaxe continua para detectar e adaptarse á variación temporal dos datos, coñecida como deriva de concepto, pódense clasificar de maneira análoga. Neste ámbito, distínguense ademais técnicas de detección da deriva e mecanismos para adaptar o modelo cando se detecta un cambio na distribución, polo que a problemática é moi diversa.

Sen embargo, expomos unha perspectiva unificadora para tratar datos non-IID tanto en dimensións espaciais (entre dispositivos) como temporais (ao longo do tempo). Para iso, preséntanse unha serie de condicións necesarias para que os modelos poidan manexar correctamente a variabilidade dos datos: dispoñibilidade de mostras representativas, mecanismos de detección de cambios e capacidade de adaptación. Isto é o que denominamos restricións proporcionadas polo problema que se trata de resolver: Cando os cambios prodúcense na probabilidade $P(x)$ no eixo espacial, non é necesario incluír ningunha restrición adicional. Pola contra, se os cambios prodúcense unicamente na probabilidade $P(y|x)$ no eixo espacial, é necesario ter suficientes mostras etiquetadas de cada usuario no momento de inicio do adestramento. Se a maiores disto, co paso de tempo prodúcense derivas de concepto nalgúns ou en todos os usuarios, é necesario ter periodicamente mostras etiquetadas de cada usuario para establecer métodos precisos de detección da deriva. Con estas restricións, cada unha das posibles casuísticas pode ser resolta co algoritmo axeitado.

Estas restricións e clasificacións están empiricamente avaliadas con experimentos expostos sobre escenarios simulados de heteroxeneidade espacial e temporal. Os resultados amosan que, cando non se cumpran as condicións mínimas establecidas para manexar datos non-IID, os modelos teñen un rendemento moi limitado. Pola contra,

as estratexias que integran mecanismos de personalización e adaptación conseguen unha mellora significativa na precisión.

Unha vez establecidas todas as posibles problemáticas e casuísticas no contexto dos datos non-IID a nivel espacial e temporal, podemos enfocarnos en atopar estratexias para liquidar ditas situacións.

En primeiro lugar, centrámonos nun problema que evoluciona co paso do tempo, e no que se experimenta deriva de concepto. Neste ámbito, presentamos un novo enfoque para afrontar a evolución temporal dos datos na FL, mediante un algoritmo denominado CDA-FedAvg (Federated Averaging con conciencia da Deriva de Concepto). O obxectivo principal deste método é detectar automaticamente cambios nas distribucións de datos que afecten ao rendemento do modelo e adaptar o proceso de aprendizaxe a estes cambios sen perder o coñecemento adquirido anteriormente, acadando un modelo final adaptable e robusto.

A deriva de concepto, no ámbito da FL, poder ter lugar en dous niveis distintos, co que a situación xeral tende a ser máis complicada. Temos por unha banda o nivel local, cando un dispositivo individual experimenta un cambio nos seus datos, mentres que o resto de dispositivos non experimentan ese cambio (polo menos ao mesmo tempo), e o nivel global, cando todo o conxunto de participantes, unha polo menos a maioría, sofre transformacións significativas. Esta distinción é fundamental para deseñar mecanismos de resposta adaptativos porque se a deriva só ten lugar nun dos dispositivos, é moi difícil que o modelo global reflicta esa deriva.

O algoritmo CDA-FedAvg que expomos está composto por tres compoñentes principais, que son a detección da deriva, a adaptación a esa deriva e a agregación federada da deriva.

- Detección da deriva: Cada cliente supervisa o seu rendemento local ao longo do tempo e aplica métricas estatísticas para detectar posibles desviacións nas distribucións dos seus datos. De forma xeral, cando o valor obtido pola función de erro aumentan significativamente nun conxunto de datos representativo, interprétase como un indicio de deriva.

- Adaptación á deriva: Unha vez detectada a deriva, o cliente inicia unha fase de adaptación. Isto implica usar exemplos representativos de datos antigos (almacenados de forma limitada para preservar a memoria) xunto cos novos datos para readestrar o modelo local, facendo que sexa capaz de clasificar os datos do novo concepto sen esquecer como clasificar os datos do concepto previo. Un aspecto clave chegado a este punto é a capacidade dos modelos de redes

neuronais para xeneralizar e aprender grandes cantidades de información sen sobreaxustarse especificamente a un dos conceptos.

- Agregación federada consciente da deriva: Os modelos locais adaptados son enviados ao servidor central, que realiza a agregación tendo en conta se os participantes experimentaron derivas. Isto permite que os modelos que sufriron cambios non prexudiquen a estabilidade do modelo global, e pode axudar a que os usuarios que experimenten a mesma deriva no futuro póidanse adaptar a ela máis facilmente.

Como comprobación empírica das boas capacidades deste algoritmo, realízanse experimentos nos que se amosa que CDA-FedAvg supera claramente a versión estándar de FedAvg en contornas non estacionarios, mentres que mantense á súa par nos contornas nos que non existe deriva. Os experimentos foron realizados sobre a tarefa de recoñecemento de actividades humanas, tendo en conta diferentes posicións do móbil para a captación de datos, e actividades diversas. O novo algoritmo mantén unha alta precisión incluso tras múltiples cambios nas condicións dos datos, demostrando unha notable capacidade de adaptación. En conxunto, CDA-FedAvg supón un paso adiante cara a modelos intelixentes máis resilientes e preparados para contornas dinámicos e cambiantes.

Visto este caso, a continuación centrámonos en problemas de datos non-IID que non evoluciona no tempo, pero que pola contra presenta unha compoñente moi heteroxénea no que concierne aos participantes do proceso de adestramento. Para liquidar estes inconvenientes, proponse unha solución innovadora chamada H&M-FedAvg (Federated Averaging Heteroxéneo e Multidimensional).

Este algoritmo busca resolver o feito de que, en moitos escenarios reais, os datos locais dos dispositivos teñen distribucións moi diferentes, o que provoca que a agregación estándar de modelos (como en FedAvg) sexa prexudicial para a precisión global, pois acada un modelo que sitúase a metade de camiño entre as distribucións locais, pero que non se corresponde realmente con ningunha das distribucións.

A achega principal desta estratexia consiste en introducir unha nova función de erro, que recibe o nome de función de erro da entropía cruzada heteroxénea e multidimensional (Heterogeneous & Multidimensional Cross-Entropy Loss), e que presenta unha adaptación concreta do cálculo do erro a este contexto. Esta función modifica a forma tradicional de calcular o erro dun modelo, integrando tres ideas clave que deben ser tidas en conta para maximizar os beneficios do adestramento:

- Separación do erro por clase: En lugar de calcular un único valor de erro conxunto para todos os datos de adestramento dun usuario, neste caso calcúlase un vector de erros, no que cada compoñente do vector correspóndese co erro cometido sobre unha das posibles clases de clasificación do problema.

- Robustez ante valores extremos: En lugar de usar a media das perdas en cada unha das clase, empregamos a mediana, o que reduce a influencia dos datos con valores atípicos e presenta un valor máis robusto e representativo.

- Penalización proporcional: Penaliza con maior intensidade o feito de clasificar mal unha mostra, ca o feito de clasificar unha mostra ben pero cunha confianza máis baixa. Isto axuda a detectar e corrixir os erros máis críticos. Ademais, a penalización é maior canto máis lonxe estea o modelo de clasificar ben a mostra.

Esta nova función de erro pode utilizarse como unha métrica que permite definir unha distancia precisa entre os modelos dos distintos participantes. A partir desa distancia, o novo algoritmo que expomos agrupa automaticamente os dispositivos en clústeres coherentes, é dicir, con participantes que teñen distribucións de datos similares. Cada clúster adestra un modelo específico, evitando que os participantes con datos moi diferentes prexudiquen o seu rendemento mutuo.

Como engadido a isto, o algoritmo H&M-FedAvg é flexible e modular: permite incorporar diferentes definicións de distancia segundo o dominio do problema ou os obxectivos do sistema. Por exemplo, usando a función da distancia euclidiana favorécese a xeneralización dos clústeres o máximo posible mentres que a función da distancia de Manhattan dálle máis peso a grandes distancias nunha das compoñentes dos vectores, polo que separa aos usuarios de xeito máis drástico. Deste xeito, o algoritmo pode adaptarse tanto a tarefas con clases desbalanceadas como a situacións con estruturas de datos complexas.

Para probar experimentalmente a viabilidade deste algoritmo e a calidade dos seus resultados, empregamos un conxunto de datos coñecido e moi empregado na literatura da FL: O DigiFive. Este conxunto de datos esta composto en realidade por cinco conxuntos distintos formados por imaxes de díxitos, cada un deles coas súas propias particularidades: imaxes reais tomadas na rúa, imaxes en branco e negro, imaxes con diferentes texturas, etc. Cada un destes conxuntos de datos pode ser interpretado como un dominio de datos diferente, aportando xa un primeiro nivel de heteroxeneidade. Como engadido a isto, en cada escenario dos experimentos as etiquetas modifícanse arbitrariamente en certos usuarios ou conxuntos de datos para engadir complexidade ao problema e testar como se comporta o algoritmo.

Nos escenarios expostos nos experimentos, o novo método de FL demostrou unha maior precisión e estabilidade que outras propostas previas, como o FedAvg, e incluso que outras estratexias que tamén realizan clústeres de clientes, como o CFL (Clustered Federated Learning). Ademais, proporciona unha interpretación máis clara da estrutura dos participantes, facilitando a análise posterior dos conxuntos de usuarios formados e dando un sentido lóxico á separación dos participantes en grupos.

En conxunto, este novo algoritmo permite afrontar a diversidade entre clientes na FL, obtendo resultados moi salientables en todas as situacións de heteroxeneidade espacial expostas. Isto é posible combinando unha nova función de erro deseñada en específico para este ámbito, e ao mesmo tempo empregando mecanismos de agrupamento dinámico dos usuarios. H&M-FedAvg representa un avance significativo cara a sistemas colaborativos máis intelixentes, adaptables e respectuosos coas particularidades de cada participante.

En conxunto, axuntando os resultados obtidos cos algoritmos de CDA-FedAvg e H&M-FedAvg, é posible construír modelos de alta precisión en contextos de datos non-IID no eixo temporal e espacial respectivamente. Como engadido final, este dous algoritmos traballan sobre dúas frontes de heteroxeneidade distintas pero son compatibles entre si, polo que sería posible deseñar un algoritmo que incorpore as técnicas e capacidades destes dous, e que polo tanto sexa capaz de obter modelos precisos e axeitados para problemas que presenten unha heteroxeneidade de calquera tipo.

Publications included in this thesis

The results presented in this thesis appear in the following two research articles, and in another one that is currently in the journal revision stage:

Non-iid data and continual learning processes in federated learning: a long road ahead

[1] **M. F. Criado**, F. E. Casado, R. Iglesias, C. V. Regueiro, S. Barro, Non-iid data and continual learning processes in federated learning: A long road ahead, Information Fusion 88 (2022) 263–280.

Title: Non-iid data and continual learning processes in federated learning: a long road ahead

Year: 2022

Author 1: Marcos Fernandez Criado, departamento de Electrónica & Computación, Universidade de Santiago de Compostela.

Author 2: Fernando Estevez Casado, departamento de Electrónica & Computación, Universidade de Santiago de Compostela.

Author 3: Roberto Iglesias, departamento de Electrónica & Computación, Universidade de Santiago de Compostela.

Author 4: Carlos Vázquez Regueiro, CITIC, Computer Architecture Group, Universidade da Coruña.

Author 5: Senén Barro, departamento de Electrónica & Computación, Universidade de Santiago de Compostela.

Journal: Information Fusion.

DOI: https://doi.org/10.1016/j.inffus.2022.07.024.

Publisher: Elsevier B. V. .

ISSN: 1566-2535 (Print), 1872-6305 (Online).

Quality indicators:

Journal Citation Reports (JCR) (2022):

- Journal Impact Factor (JIF): 14.8.
- Quartile: Q1 in categories *Computer science, theory and methods* (2/111) and *Computer science, artificial intelligence* (4/145).

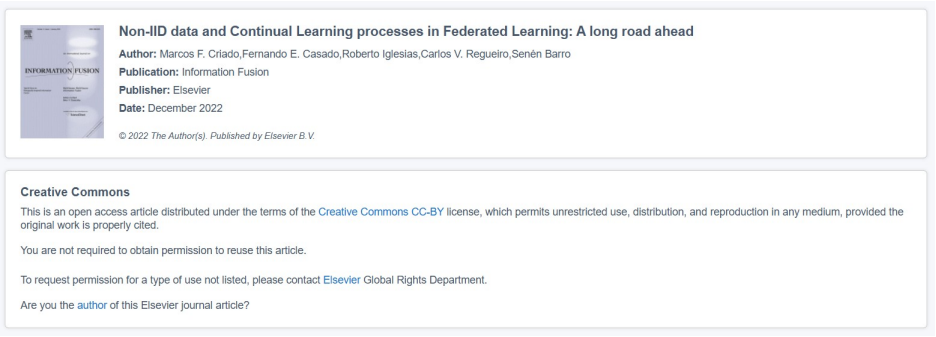
Corresponding chapter: 5.

PhD Candidate Contributions:

Essential. The candidate took part in the design of the research, the conceptualization, the search of related bibliographical references, the methodology, the experiments design, the visualization and analysis of the results, and in the writing of the work.

Reproduction rights:

The publication was partially reproduced in Chapter 5 under Creative Commons Attribution (CC BY) license.



Non-IID data and Continual Learning processes in Federated Learning: A long road ahead

Author: Marcos F. Criado,Fernando E. Casado,Roberto Iglesias,Carlos V. Regueiro,Senén Barro

Publication: Information Fusion

Publisher: Elsevier

Date: December 2022

Concept drift detection and adaptation for federated and continual learning

[2] F . E. Casado, D. Lema, **M. F. Criado**, R. Iglesias, C. V. Regueiro, S. Barro, Concept drift detection and adaptation for federated and continual learning, Multimedia Tools and Applications (2022) 1–23.

Title: Concept drift detection and adaptation for federated and continual learning.

Year: 2022.

Author 1: Fernando Estevez Casado, departamento de Electrónica & Computación, Universidade de Santiago de Compostela.

Author 2: Dylan Lema Pais, CiTIUS (Centro Singular de Investigación en Tecnoloxías Intelixentes), Universidade de Santiago de Compostela.

Author 3: Marcos Fernandez Criado, departamento de Electrónica & Computación, Universidade de Santiago de Compostela.

Author 4: Roberto Iglesias, departamento de Electrónica & Computación, Universidade de Santiago de Compostela.

Author 5: Carlos Vázquez Regueiro, CITIC, Computer Architecture Group, Universidade da Coruña.

Author 6: Senén Barro, departamento de Electrónica & Computación, Universidade de Santiago de Compostela.

Journal: Multimedia Tools and Applications.

DOI: https://doi.org/10.1007/s11042-021-11219-x.

Publisher: Springer US.

ISSN: 1573-7721 (Online), 1380-7501 (ISSN-L).

Quality indicators:

Journal Citation Reports (JCR) (2022):

- Journal Impact Factor (JIF): 2.57.
- Quartile: Q2 in category *Computer science, software engineering* (32/108).

Corresponding chapter: 6.

PhD Candidate Contributions:

Eseential. The candidate took part in the design of the research, the conceptualization, the development of the mathematical component of the work, the experiments design and the subsequent analysis of the results, as well as in the writing of the work.

Reproduction rights:

The publication was partially reproduced in Chapter 6 under Creative Commons Attribution (CC BY) license.

SPRINGER NATURE

Concept drift detection and adaptation for federated and continual learning

Author: Fernando E. Casado et al
Publication: Multimedia Tools and Applications
Publisher: Springer Nature
Date: Jul 17, 2021

Copyright © 2021, The Author(s)

Creative Commons

This is an open access article distributed under the terms of the Creative Commons CC BY license, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

You are not required to obtain permission to reuse this article.
To request permission for a type of use not listed, please contact Springer Nature

CHAPTER 1

Introduction

The advent of the Internet of Things (IoT) has transformed traditional computational paradigms in the last few decades in every possible aspects of our daily lives [3]. Our society has gradually learned to be surrounded by different kinds of devices that constantly facilitate the tasks we make, even when we are not aware of it. This technological evolution enables vast networks of interconnected devices to generate, process, and share data autonomously.

The cutting-edge technology we developed has contributed to the exponential growth of data, necessitating sophisticated approaches to extract actionable insights. Machine learning (ML) stands as a cornerstone of this data revolution, offering robust methodologies to uncover patterns, optimize processes, and predict future phenomena. At its core, the goal of ML is to build intelligent systems capable of learning from data to perform tasks with minimal human intervention. The amount of data collected by our wearables, mobile phones, robots, etc., opens up an immense myriad of application opportunities in domains such as banking, health, education, travel, etc. [4, 5]. However, the increasing ubiquity of data-generating devices has also prompted a reevaluation of centralized ML approaches, which need to aggregate all the data available in the same storage location, paving the way for decentralized paradigms.

Decentralized machine learning represents a paradigm shift in which computation and data remain distributed across multiple nodes or devices, eliminating the need for centralized data aggregation. This architectural shift addresses critical concerns related to privacy, scalability, and latency, which have emerged as significant challenges in the era of big data.

Privacy concerns are paramount in scenarios where sensitive information, such as healthcare records or financial data, is generated and processed [6]. It is of the

utmost importance in applications like this not to aggregate personal information on a central server. In fact, nowadays, digital privacy has become such a primary concern of ML that lots of governments around the globe have been implementing legislation to restrict and control the data that can be collected from consumers at different levels, like the information stored in our phones, the trace left on the internet by the users, and so on. Examples of these legislations include, for instance, the U.S. Consumer Privacy Bill of Rights [7], or the General Data Protection Regulation (GDPR) developed in Europe [8].

Scalability, on the other hand, becomes increasingly critical as the number of IoT devices continues to proliferate, placing unprecedented demands on centralized data centers [9,10]. This poses a challenge for Centralized ML strategies as we know them, and for this reason decentralized ML processes were born, endeavoring to mitigate these issues by enabling local computations while maintaining collaborative learning across the network.

In this context, Federated learning (FL) [11,12] emerges as a pioneering framework within the realm of decentralized ML, introduced to harness the computational capabilities of edge devices while addressing privacy constraints. Originating as a solution to the challenges associated with centralized data storage, FL allows multiple participants to collaboratively train machine learning models without sharing their raw data [13,14]. Typically, the model in this context is a deep neural network (DNN). This approach not only safeguards user privacy but also reduces the dependency on centralized infrastructure. However, while FL has demonstrated remarkable potential, it is not without its challenges [15,16]. The ideal setting for developing an FL model would be to have data available from a high number of participants in the training process, where all of them present the same kind of data samples, collected in similar environments and conditions. This situation is very rarely the one addressed in a real-life scenario. A central issue in FL is precisely the heterogeneity of data across participants, which refers to the non-identical and non-independent distribution of data (non-IID data). This phenomenon, often attributed to varying user behaviors or device capabilities, introduces lots of complexities in model convergence and generalizability.

Furthermore, the problem of non-stationary data presents a significant hurdle in FL. Non-stationarity arises when data distributions evolve over time, which is particularly prevalent in dynamic environments such as IoT networks. This variability necessitates adaptive learning strategies to ensure the trained models remain effective and relevant. For FL frameworks, non-stationary data raises an even bigger obstacle, as not all of the devices participating in the model learning must behave in the same

way. Addressing these challenges requires a nuanced understanding of FL's foundations, as well as innovative methodologies to enhance its robustness and scalability.

Federated Learning has already been successfully applied for lots of real-world applications, but at the same time, it is a recent paradigm with many open issues and lots of open roads to explore and improve. The term Federated Learning is surprisingly new, and back when this research project started in 2020, there was very little information and experimentation done in this area. The foundational work using the name Federated Learning [17] was published in 2016, but this term started spreading around in 2019, and it was then that it became more commonly known, and the number of publications in this area started to grow more and more.

This dissertation aims to contribute to the advancement of Federated Learning by analyzing the open challenges involved and exploring novel techniques to address the well-known threats of data heterogeneity and non-stationarity. By leveraging the principles of decentralized learning and integrating cutting-edge approaches, this research seeks to enhance the practical applicability of FL in real-world scenarios. The subsequent chapters will provide a comprehensive analysis of existing literature, methodological advancements, and experimental evaluations to elucidate the potential and limitations of FL in the context of IoT and beyond.

Structure of the thesis

This manuscript is organized as follows:

In Overview (Chapter 4), we will provide detailed explanations of useful concepts and state-of-the-art research in transversal AI fields, such as Machine Learning, Distributed Machine Learning, and Continual Machine Learning. These settings have been of the utmost importance during the development of this research, and a lot of interesting and useful notions and ideas need to be explained for the work presented in the next chapters.

Next, in Chapter 5: non-IID data in Federated and Continual Learning, we will further discuss the topic of heterogeneous data and provide formal definitions and precise classifications of prior works on these fields. This knowledge is crucial to understand potential issues that may arise in real-life problems and also provide very important insights on how to deal with this kind of inconvenience. Furthermore, some restrictions are stated regarding the necessary information to face the different types of data heterogeneity.

After having discussed the data heterogeneity casuistry, Chapters 6 and 7 present two different algorithms that are able to get accurate results in the presence of non-

IID data. In the first case, Concept-Drift-Aware Federated Averaging (Chapter 6), we design a robust algorithm that is able to detect data heterogeneity over time (Concept Drift) and adapt accordingly. On the other hand, in Chapter 7: Heterogeneous and Multidimensional Federated Averaging, we present an FL algorithm that faces data heterogeneity across the participants of the training process, identifies and understands the cause of data heterogeneity, and reacts to it dynamically.

CHAPTER 2

Hypotheses and Objectives

One of the key aspects that support building Deep Learning models and training processes is the use of huge amounts of data for the training stage. However, one of the main difficulties when performing FL strategies is usually the restricted amount of data samples each participant owns. For this kind of strategy to succeed, it is essential to reach a consensus across all the different models trained. The main objective of this research is to develop FL algorithms that allow both a successful outcome for all of the participants involved in the training process as well as the continuous integration of this knowledge for future models fine-tuning.

The concept of FL was born in 2016 to talk about a technique of distributed ML that favors and enhances the training of a common model for a collection of devices without the necessity of sharing their local, confidential information [13]. The challenges of such a technique are data heterogeneity across these devices and also their data dynamism. It is important to mention, although it will be further explained in future sections, that this data dynamism concerns both the spatial axis (different visions of the same problem based on the different data domains perceived by the participants of the FL training) [18] and the temporal axis (evolution of the data each participant collects as time passes by) [2].

To tackle these challenges in an organized manner, the following step-by-step objectives were brought up:

1. Research on the state-of-the-art works regarding the current state of FL advances and the variety of strategies that have already been developed for taking into account the issues of data heterogeneity and data dynamism.

2. Classification of the FL strategies designed so far. This will be help to identify particular challenges that have not been addressed, and as well as challenges

that have not been identified before. In this stage, it will be very important to identify an underlying pattern in the prior works developed in this field of investigation. The main idea behind this task is to establish useful strategies for dealing with data heterogeneity that might be improved or refined in the future.

3. Local resolution of an ML problem set in a realistic setting, leveraging non-stationary data and thus processing time-series data samples to recognize particular patterns that will improve the model performance. The local model designed in this stage will be the baseline that will be later improved to achieve an incremental strategy for FL. Starting with a local problem, the focus will be placed on the heterogeneity of data in the temporal axis, avoiding the data heterogeneity across different devices that will later be added when the FL scenario is considered.

4. Formulation and design of an FL approach that enables for continual integration and combines the local and global data models created by the devices, allowing to address it data heterogeneity in both spatial and temporal axes. For this, it will be necessary to create an automatic tool to detect the data shift as time progresses and so the global model will also be updated and adjusted accordingly.

5. Publication of the main contributions and interesting results in high-impact journals. In this sense, considering all of the work developed throughout this research, two papers have been published in JCR-Q1 and JCR-Q2 journals, as well as one other paper currently presented for publication. This monograph assembles all the contributions achieved during these past years of work.

CHAPTER 3

General Methodology

Keeping in mind the objectives we just explained, we followed a well-designed road map to achieve them. We tackle a series of working stages based on simple straightforward tasks that conform a cyclic process for each of the objectives presented.

In the first place, each of the tasks we aim to fulfill must begin with a bibliographical review. Nowadays, FL is a very active research field, and papers related to both FL, distributed ML, and data heterogeneity in ML are published constantly. For this reason, it is important to keep track of the progress made and take advantage of it to continue working in the same direction. Having a clear picture of the work done so far will help identify shortages that must be improved.

Once this first step is done, a conceptual model has to be designed. At this point, multiple hypotheses and ideas will be proposed. For each of them, the goal is to determine whether they are reasonable, achievable, and efficient.

The idea behind this analysis is to disprove hypotheses that cannot be tested or carried out for a variety of reasons to avoid spending more time on them. Also, from the remaining hypothesis, we must determine which ones can be realistically accomplished and which will be more profitable in the short and long term. This can be decided based on the time cost they would take, the probability they have to be successful, the different casuistry that needs to be accounted for, and so on.

Afterwards, when one or more proposals have been determined, they must be systematically executed using the FL settings: a bunch of devices with their own sets of data, training a joint model without sharing their private information. It is possible that, due to the nature of the method proposed, some additional restrictions will be implemented along the basics of FL. This can be done to simplify the initial phases of the experimentation or as part of the previous step to check if a proposal is really viable.

This point is usually the one that takes more time to execute, so it is very critical to apply it only to realistic, feasible proposals. The validation phase of these hypotheses should consist of performing multiple experiments, varying the data collected by the devices, repeating the experiments to guarantee statistically significant results, and simulating challenging scenarios to test the capabilities of the approaches proposed.

Finally, once it is clear that we have come upon a useful algorithm, the results achieved must be recorded, again ensuring their statistical significance and enhancing the good properties and assets it provides. At the same time, we must also determine the extension of our model, meaning that we must know not only the problems it can solve but also the ones it cannot to clearly settle the barrier of what is yet to be improved in the future.

Results and Discussion

CHAPTER 4

Overview

In this chapter, we will give the basic concepts to delve into the rest of this PhD dissertation. Machine Learning (ML) is a huge field of research with lots of different aspects, and for the present research we aim to merge learning strategies from different subfields of ML. For this reason, we structure this chapter into the following sections:

- Machine Learning (ML): we will first focus on the basics concepts that started this broad research topic. These notions are then used and reinvented in different ways for investigations in specific tasks, so it is important to have a clear picture of where it all comes from initially. In particular, we will highlight the concepts regarding measuring errors in the models, the well-known loss functions in ML algorithms, and so on.

- Distributed Machine Learning (DML): DML is in a way the predecessor of Federated Learning, setting the basic ideas and initial attempts to perform decentralized ML computations. Federated Learning (FL) will be the main topic throughout the following chapters. For this reason, it is of the utmost importance to settle the basics clearly and establish some common definitions and explain concepts precisely.

- Continual Learning (CL): as was already mentioned previously, the main objective of this work is to provide useful tools for dealing with heterogeneity in the context of ML. One way this heterogeneity might be present is over time, when the circumstances for training a model change, or the way devices collect data shifts, for instance. Knowledge regarding the basic concepts and difficulties concerning CL will be valuable for better understanding future chapters.

4.1 Machine Learning

Machine Learning is a subfield of Artificial Intelligence (AI) focused on developing tools to provide different kinds of machines with the ability to learn how to solve a specific task.

The first reference to the concept of *Machine Learning* was in 1952, by Arthur Samuel. In his research, Samuel showed that it was possible to teach a computer into playing checkers [19]. This investigation was further explored later, by Frank Rosenblatt, who implemented for the first time the forerunner of what we know nowadays as a neural network: the perceptron [20]. In the next decades, the field of ML kept growing, and enlarging. Lots of different approaches to improve the previous results were designed, such as the multilayer perceptron [21], or the method of the nearest neighbor [22]. As time passed, in the 1980s and 1990s the development of technology and the increase in the computer capabilities allowed researchers to investigate more complex calculations, and create new kinds of methods, such as decision trees [23], support vector machines (SVM) [24] and other ground-breaking algorithms.

Another remarkable algorithm was born in 1986, introduced by the researchers Rumelhart, Williams, and Hinton. It is the case of the backpropagation algorithm, which presented a procedure to train multilayer perceptrons, or neural networks [25]. Backpropagation is an algorithm that makes use of well-known mathematical tools, and applies the chain rule along the perceptrons of a neural network to compute the gradient of a predefined loss function. After that, the parameters of the neural network can be updated in a way that reduces the loss, hence improving accuracy. Loss functions will be discussed profoundly in Section 4.1.1.

With the arrival of the 21st century, the increase in the quantity of data collected and available became stronger than ever, receiving the name of Big Data, and the problem of how to deal with such amounts of data arose. There was a necessity of more powerful algorithms in ML to work with it. As a consequence of this, larger and more complex neural networks were designed, forming what it is known as Deep Learning (DL) [26,27]. This kind of techniques, jointly with the availability of large volumes of data lead to a brutal acceleration in the pace of research over the last decades, allowing lots of different DL algorithms to be created, applied, and commercialized. The fields of application are countless: from agriculture [28] to engineering systems [29], going through all possible areas such as entertainment [30], healthcare and diseases detection [31], finance, computer vision, cibersecurity and deep-fake detection, and so on.

In the last decade, the improvement and accessibility of electronic components, jointly with further improvements in communications and data availability, have continued. As a result, there has been an exponential growth of resources generated in terms of data and devices: computers, tablets, smartphones, and any kind of IoT device. With such an overflow of data around us, the problem has shifted from having to use the data available in an efficient way (when there was little data for ML models training), to instead having to select appropriate data among all the accessible resources. It has become more important than ever to make good use of the data available, and train algorithms in a more efficient way.

When ML was first conceived, it focused on developing tools to provide different kinds of machines with the ability to learn how to solve a specific task. This learning must progressively improve with time, based on information on the previous attempts to solve the problem, and at the same time without being designed to produce a specific outcome. The final result of an algorithm is an output created by the initial setting of the task and the way the machine uses the information it collects while solving the problem itself. This information, which is based on the machine interaction with the task, is what we call *Experience*.

Experience can come from an external source, for instance a reinforcement boost that allows the machine to know how its performance was. In this aspect, lots of works have been developed precisely under the name of Reinforcement Learning (RL) [32–34]. This boost, usually called *reward*, provides a measure for the machine to understand how well it acted on the issue it was solving. The final objective of the machine is to get the highest reward possible. There are multiple strategies for determining the value of this reward [35,36], but they fall out of the scope of this PhD dissertation.

On the other hand, experience can also be implemented directly on the learning algorithm, based on data collected by the machine itself while solving the task. In this sense, an ML algorithm is capable of improving autonomously, through a process called *training*, achieving an optimal approach to fulfill the desired task. The result of the training stage is what we call *model*, and it represents the ultimate strategy designed, that aims for the highest accuracy possible when performing on the task it was trained for.

More formally, we next define the set of possible inputs and outputs a machine can register and produce, respectively. These notions will come in handy to understand what an ML model is.

Definition 4.1.1. Let $\mathcal{X}$ be the space of all possible input vectors for a machine, also known as feature space. Let $\mathcal{Y}$ be the set of possible outputs, or categories, a feature vector can be associated with. A data sample is a paired tuple of feature vector and corresponding output (x, y), with $x \in \mathcal{X}$ and $y \in \mathcal{Y}$.

Definition 4.1.2. Let $D = \{(x_i, y_i)\}_{i=1}^{M}$ be a training dataset, consisting of M samples. A ML model is a mapping function that associates each possible feature vector with an output, as follows:

$$\begin{array}{rccc} f: & \mathcal{X} & \longrightarrow & \mathcal{Y} \\ & x & \mapsto & f(x) = \hat{y}. \end{array}$$

Note that when defining data samples as we just did, we are assuming that the corresponding output for a feature vector is known. This is not always the case. ML models can be also designed and trained without the necessity of having these outputs in advance, and in fact, notice that the outputs of the data samples are not used in the previous definition of ML model. According to this casuistic, algorithms in ML can be classified into:

- Supervised Machine Learning [37,38], when all the correct outputs for the samples of the training dataset are known. In this case, the known outputs are called labels. Having information about the labels of the samples is useful to find and extend an accurate model for the whole feature space. Supervised Machine Learning problems are usually split into classification problems [39], or regression problems [40], depending on the nature of the labels.

- Semi-Supervised Machine Learning [41,42], when only a subset of the whole training dataset has its correct output associated. Methods gathered under this framework try to find a model to predict the output given by some labeled inputs, but at the same time can make use of unlabeled samples to help reach a higher accuracy and a better performance.

- Unsupervised Machine Learning [43,44], when the output of each sample from the training dataset is unknown. In this case, the algorithm needs to find relevant information based only on the data samples inputs.

For the rest of this monograph, we will focus on supervised settings, i.e., it will be assumed that all the data samples have a correct output, or label, associated. In

addition, it will also be assumed that the problem to solve is a classification problem, i.e., the labels belong to a set of specific possible categories to choose from. For this reason, unless it is specified otherwise, we will consider $\mathcal{Y} = [0, 1]^C$, where C is the number of possible categories, or classes.

This way of understanding the space of possible outputs, or labels, will be useful for notation purposes and design of algorithms moving forward. From this consideration, an output of a model $\hat{y} \in \mathcal{Y} = [0, 1]^C$ is a vector of C components, varying between 0 and 1, that normally represent the likelihood of an input x to belong to a certain category.

With these definitions, we can already create an ML model, just by setting a mapping to associate each of the input samples x_i with a corresponding output $\hat{y}_i$, and then extending this mapping to the whole set of features $\mathcal{X}$. The next step would be to design a training process so that this mapping can be improved to get more accurate results and a better performance overall. Moreover, we have not made use yet of some of the knowledge available. The labels of the samples are not playing any role in the model creation at the moment, and they hold valuable information of the expected results that the model should get.

4.1.1 Information theory: Cross-Entropy loss function

In Supervised Machine Learning settings, where the correct label for the data samples is known in advance, it is common to use this information to get the most accurate model possible. For this, what the models try to achieve is to match the output obtained when mapping a known feature vector sample x_i using the model function f, with the label for that feature vector, y_i. In other words, $f(x_i) = \hat{y}_i \simeq y_i$.

The best way to achieve this kind of result in our model is to effectively compare the result obtained by a model, with the right value it should obtain. But it is a non-trivial question how to compare these two values, or even harder, how to use and incorporate this comparison in the training process. This is where information theory comes into place.

Information theory originated in the mid-20th century as a response to growing challenges in communication technology. As radio, telegraphy, and telephony became central to modern life, scientists and engineers began seeking a deeper understanding of how information could be transmitted efficiently and accurately over physical channels. However, there was no systematic or mathematical way to define or measure information itself, which made it difficult to address issues like noise, data compression, and transmission limits.

This changed in 1948 when American mathematician and engineer Claude Shannon published his groundbreaking paper, A Mathematical Theory of Communication [45]. In this seminal work, Shannon introduced a formal framework for quantifying information, treating it as a measurable and transmittable quantity based on probability theory. In this way, the amount of information in a message is related to its unpredictability, i.e., the more surprising a message is, the more information it conveys. This led to the introduction of the concept of entropy, a mathematical measure of uncertainty in a set of possible messages, which became a cornerstone of information theory.

Drawing inspiration from earlier work in statistics, thermodynamics, and electrical engineering, Shannon's theory unified a wide range of communication problems under one mathematical umbrella. The impact of his work was profound, laying the foundation for digital communication, data compression techniques like ZIP files and MP3s, reliable data storage, and much more. Its applications spread over fields like molecular biology [46], psychology [47], and of course computation [48]. Over time, information theory has also influenced fields far beyond engineering, including computer science, neuroscience, linguistics, and even physics, making it one of the most important conceptual breakthroughs of the 20th century. One of the key components that led to huge advances in the field of ML was the information entropy.

The information entropy, also known as Shannon entropy, quantifies the average level of uncertainty associated with a variable's potential states or possible outcomes. In a sense, it measures the expected amount of information it would take to describe the state of the variable. According to this definition, the more unexpected a result is, the more information it provides to know it.

Definition 4.1.3. Let $z \in \mathcal{Z}$ be a discrete random variable distributed according to a certain probability $p : \mathcal{Z} \longrightarrow [0, 1]$. The entropy of z is defined as:

$$H(z) := -\sum_{z \in \mathcal{Z}} p(z) \cdot \log p(z).$$

Where $\mathcal{Z}$ represents all possible values of z.

At this point, $H(z)$ quantifies the amount of information a certain state of the variable z provides about its probability distribution p. It would be very useful to apply this concept of entropy for a set of 2 non-independent probability distributions, like the mapping function of a ML model, $\hat{y} = f(x)$, and its corresponding label y, which is also dependent on the variable x.

This concept was also deeply studied and solved [49], using mathematical tools like Kullback-Leibler divergence [50], to achieve the term of cross-entropy, in the following way:

Definition 4.1.4. Let p and q be discrete probability distributions over a common underlying set of events $\mathcal{Z}$. The cross-entropy of the distribution q relative to a distribution p over a common support $\mathcal{Z}$ is given by:

$$H(p,q) := -\sum_{z\in\mathcal{Z}} p(z)\cdot \log q(z).$$

Where $\mathcal{Z}$ represents all possible values of z.

Ultimately, these concepts from information theory were applied into ML and allowed great improvements in the training process of ML models. When training a ML classifier, the goal is to create a mapping function f, called a model, that associates each possible input data sample with a unique output from the set of the C possible classes. Ideally, the output should be the correct class for that sample:

$$\begin{array}{rccc} f: & \mathcal{X} & \longrightarrow & \mathcal{Y} = [0,1]^C \\ & x & \mapsto & f(x) = \hat{y}. \end{array}$$

Depending on the machine learning method employed, the function f will rely on some more parameters. For instance, when using DL strategies, f will vary according to the weights of the corresponding neural network: $f_w(x) = \hat{y}$. However, to keep the notation as straightforward as possible, we will call it simply f, and not f_w.

Creating such a function with only a limited amount of labeled data samples $\{(x_i, y_i)\}_{i=1}^{M}$ is a difficult task, and in general, there is no way of doing it using a single-step formula. In order to perform the training of a model, it is common in the literature to establish a loss function or error function, and try to minimize the empirical risk of the classifier through that loss function to get a higher accuracy and a better overall performance.

Loss functions assign positive real values to a sample (x, y) depending on the correct output of that sample, y_i, and the predicted value obtained by the classifier, $f(x)$:

$$\begin{array}{rccc} \mathcal{L}: & \mathcal{X}\times\mathcal{Y} & \longrightarrow & \mathbb{R}^+ \\ & x\ ,\ y & \mapsto & \mathcal{L}(f(x), y). \end{array}$$

Lot of research has been done trying to achieve the best loss function possible, i.e., the loss function that grants more information for creating an accurate model when

minimized. Resuming with the previous discussion about information theory, and particularly information entropy, in this setting we can compute the cross-entropy of the distribution $f(x)$ relative to the probability distribution of $y(x)$, and using this concept a loss function can be defined: the Cross-Entropy Loss function (CEL) [51]:

Definition 4.1.5. Given a sample (x, y) used for training a ML model f to solve a classification problem consisting of C possible classes, the Cross-Entropy loss of the sample (x, y) is the following:

$$\mathcal{L}_{CE}(x, y) = -\sum_{j=1}^{C} y_j \cdot log(f_j(x)).$$

Notice that, as mentioned in Section 4.1, in the previous definition we are considering y_j as a vector of C components, varying between 0 and 1. As it is a label vector, one of its components is equal to 1 and all of the rest are equal to 0, meaning that only one of the terms of the sum we just defined is non-zero.

After defining the CEL of a single sample, a natural extension of this definition is to define the same concept for a set of samples, which can be expressed as follows:

Definition 4.1.6. Given a set of M samples $D = \{(x_i, y_i)\}_{i=1}^{M}$ used for training a ML model f, the CEL of the model achieved over the dataset D is defined as the sum of the individual losses of each sample:

$$\mathcal{L}_{CE}(D) = -\sum_{i=1}^{M}\sum_{j=1}^{C} y_{i_j} \cdot log(f_j(x_i)). \tag{4.1.1}$$

This formula is used across almost all of the works in DL learning for computing errors in neural networks and train aiming to minimize its value.

It is also very common in the literature of ML algorithms to assume that the data employed in the training process of a model is stationary, unadulterated as time passes, and they also use centralized settings. This means they were created with the aim of being executed using static datasets and a single machine to process all the information. Transforming these works into a multi-device setting would imply to have a bunch of devices sending their local information to a central server, where they would all get stored and used for the model training. As mentioned previously in the Introduction, this approach presents potential problems in terms of scalability, storage capacity, communication between devices and the central server, and, of course, privacy.

Additionally, we live in a constantly changing world, and algorithms should also be able to evolve and account for these changes, meaning handling non-stationary data. Consequently, ML approaches that leverage continual settings and also distributed settings are a much more fitting and natural framework for multi-device settings.

4.2 Distributed machine learning

The origins of Distributed Machine Learning (DML) [52, 53] can be traced back to the increasing computational demands of traditional ML algorithms as datasets grew in size during the late 1990s and early 2000s. It was explained in Section 4.1 that initially ML models were trained on single machines, but as data from internet usage, mobile devices, and enterprise systems exploded, it became clear that single-machine training was neither scalable nor efficient. This became a huge motivation for finding new ways of scaling up algorithms, making them able to cope with the ever-increasing amounts of data and also trying to accelerate the training processes of ML models.

Researchers began exploring ways to distribute the computation across multiple nodes, especially in the context of parallel computing and grid computing. Lots of approaches and proposed methods in these years consist of artificially splitting the data among different devices to aim for that parallelization. The emergence of big data technologies in the 2010s, such as Hadoop [54] and later Apache Spark [55], provided the infrastructure needed to enable distributed data processing. This convergence of big data and ML gave rise to DML systems.

Early DML systems aimed to scale model training across data centers or clusters, relying heavily on data parallelism and model parallelism. However, the design of such systems required careful architectural choices. Based on these systems design, algorithms for DML can be classified into 2 types of dominant settings: peer-to-peer architecture and client-server architecture.

The client-server architecture [56, 57] is the most common and traditional approach used in DML. In this setup a central server (or parameter server) holds the global model, while multiple client nodes (workers) perform the computations (e.g., compute gradients on local data). After this, clients periodically send their local updates to the server, and the server aggregates these updates and sends back the updated global model to the clients so they can continue with the computations and improving the global model more and more. This architecture was popularized by systems such as Google's DistBelief and later TensorFlow [58, 59], where the server controls coordination and consistency.

This approach presents several advantages, like the existence of a centralized server that provides control and easy coordination. At the same time, it is simple to implement synchronization strategies so that client nodes work simultaneously with the same version of the global model. On the other hand, it also presents some drawbacks, like the communication bottleneck at the server, having to wait for clients updates and thus delaying the use of computational resources and ultimately, the training on certain clients. In addition, it presents poor scalability in very large distributed systems.

Peer-to-Peer architecture (P2P) [60, 61] presents, as opposed to the previous one, a setting where there is no centralized server. Each peer (node) contributes both data and computational resources, leveraging the lack of a central orchestrator. Peers share their model updates directly with each other (often through gossip protocols or consensus mechanisms), and in this way communications get more fluent and avoid bottlenecks. P2P models are less common but have seen interest in DML systems and blockchain-based ML initiatives. Examples include systems inspired by BitTorrent or decentralized training for edge devices.

Like its client-server counterpart, this approach provides its own pros and cons. On the bright side, lacking a central server means that there is no central bottleneck in terms of communications. Additionally, as models are computed and shared between all servers constantly, these kinds of strategies have a high fault tolerance. On the contrary, not having a central orchestrator hinders a lot the synchronization task, making it more complex. As a consequence of this, it is difficult to ensure convergence and consistency across all of the nodes. Also, large networks present a high communication overhead.

Whether the architectural configuration of the DML setting is one of the other, each of the nodes perform a fraction of the total computations, and the individual work of each of them is then aggregated into a unified final model.

In some of these approaches, each participant device computes a local model on its own, and the collection models developed is then used to achieve a final result through a consensus. However, most of the methods created in these decades consist of splitting the training of a single model, shared by all the participant nodes, into smaller tasks that can then be solved by a different device.

Initially, this kind of solutions proposes adaptations of already created ML algorithms through some parallelizable optimization approach. It is particularly abundant the use of strategies based on the algorithm of Stochastic Gradient Descent (SGD) [62]. The reason for this is that SDG is a very pragmatic algorithm, prone to be applied to DL strategies, using neural networks, as well as more classic ML proposals, like SVMs.

The basic idea behind SGD algorithm is to minimize a function that represents the loss of the model (for instance, the cross-entropy loss function applied to the model training). The way of proceeding for this minimization is to compute the gradient of the loss function. Recall that, mathematically, the gradient of a function represents the direction that fastest increases its value. After performing this computation, parameters of the model, or connection weights in the case of neural networks, are adjusted in the opposite direction to the gradient. Currently, there exist lots of different variations of the classic SGD algorithm for the distributed framework [63–66].

Lately, the point of view regarding DML has shifted. In the last decade there has been an evolution in the mentality concerning distributed frameworks for the creation of ML models. The main concern regarding the increase in the number of devices and the amounts of data available has turned into a concern about digital privacy, and in that context, the field of DML has progressively transitioned into was we know today as Federated Learning.

4.2.1 Federated Learning

Federated machine learning (FL) [11,12,14,17] is an innovative framework strongly connected with DML that focuses on the training of a ML model across a bunch of devices. Each of the devices that participate in the model creation locally collects their own samples, and those samples are kept private, not shared with the rest. The change in the perspective with DML algorithms is obvious: Whereas in the DML framework the starting point was having all data gathered in a single server and it was split to facilitate the model training, in the FL settings data is spread across the servers from the beginning, by the own nature of the problem that is addressed.

The purpose of FL is to be especially careful about the security and data privacy of local data, and for this reason it further decentralizes computations from the central server, which work is basically reduced to be a coordinator between the different clients. With this approach, bottlenecks produced by the central server in DML algorithms are softened, but in exchange these operations have to be performed locally.

The greater challenge of FL strategies is the heterogeneity it must manage. From the starting point, each of the participants may have different capabilities, computing resources, amount of samples, etc. Moreover, the skew distributions of the data samples among the devices can suppose a big threat in terms of model convergence.

The standard FL approach, firstly proposed by researchers H. Brendan McMahan, Jakub Konečnỳ, and Daniel Ramage [11] in 2015, consists of an alternation between local training rounds performed by each device, followed by the corresponding global

aggregation of the results obtained. Performing training with this approach, the only exchange of information between the clients and the central server is the model parameters, avoiding sharing raw data. This groundbreaking algorithm received the name of Federated Averaging (FedAvg), and it describes a supervised way of training a Deep Neural Network (DNN) in a federated way.

The idea behind FedAvg algorithm is the following: Assume a set of participants $\mathcal{N} = \{1, \dots, N\}$ aiming to train a DML model in a federated way. The model is defined as the set of weights of the neural network connections, $\mathbf{w} \in \mathbb{R}^V$, where V stands for the number of biases and connections between neurons in the network, i.e., the number of parameters to be calculated. FedAvg performs a fixed number of training rounds, R, that alternate local training processes and a global aggregation. The model $\mathbf{w}$ is initialized randomly in the central server, and sent to the participants. In this fashion, all clients begin their local training at a common starting point. For each round, $r \in \{1, \dots, R\}$, a random subset of participants $\mathcal{N}^r \subseteq \mathcal{N}$, of size N_r is selected, and they train the model $\mathbf{w}_G^{r-1}$ using their local data and SGD algorithm. After that, each client $i \in \mathcal{N}^r$ send back to the server their model update, $\mathbf{w}_i^r$. Finally, the central server aggregates the updates received and a new round of training begins:

$$\mathbf{w}_G^r = \sum_{i \in \mathcal{N}^r} \frac{M_i}{M} \mathbf{w}_i^r,$$

where M_i stands for the amount of data used by the participant i on their local training and M is the sum of the quantity of samples M_i of the participants involved in the training in that round. Note that participants with a higher amount of samples influence the global model more than others, which seems a reasonable approach since they have more information about the task beforehand. The general pseudo-code of this algorithm is depicted hereafter in algorithms 1 and 2:

Algorithm 1: Federated Averaging - Server side

Input : Set of clients $\mathcal{N} = \{1, \dots, N\}$, number of participants in each round, N_r, local batch size B, number of local epochs E, number of learning rounds R, and learning rate η

Output: Global model, $\mathbf{w}_G$

Initialization: Random parameter initialization $\mathbf{w}_G^0$
for *r in* $\{1, \dots, R\}$ **do**
  $\mathcal{N}^r \leftarrow$ random subset of S clients
  for *each client i in* $\mathcal{N}^r$ *in parallel* **do**
    $\mathbf{w}_i^r \leftarrow \text{LocalTraining}_i\ (\mathbf{w}_G^{r-1}, \text{B, E, } \eta)$ using SGD
  $\mathbf{w}_G^r \leftarrow \sum_{i=1}^{N_r} \frac{M_i}{M} \mathbf{w}_i^r$
return $\mathbf{w}_G$

Algorithm 2: Federated Averaging - LocalTraining_i

Input : Global model from previous round $\mathbf{w}_G^{r-1}$, local batch size B, number of epochs E, and learning rate η

Output: Local update of the model, $\mathbf{w}_i^r$

Split data into a collection of sets $\mathcal{B}$ of size B
for *e in* $\{1, \dots, E\}$ **do**
  for *b in* $\mathcal{B}$ **do**
    $\mathbf{w}_i^r \leftarrow \mathbf{w}_G^{r-1} - \eta \cdot \nabla \mathcal{L}(\mathbf{w}_G^{r-1}, b)$
return $\mathbf{w}_i^r$;

After FedAvg algorithm was published, FL attracted a lot of attention and the number of works on this field increased exponentially. There has been a lot of research trying to either find better algorithms for FL, or improve FedAvg accuracy or efficiency. In addition, several protective procedures have been explored to increase the privacy robustness and provide additional security to the FedAvg algorithm, using strategies like adding differential privacy [67,68], or encrypting the model updates before sending them to the central server [69,70]. It has also been studied if the transmission of the model parameters $\mathbf{w}$ could lead to privacy leakage [71–73].

All the research performed in the past years has clearly exposed the characteristics of FL strategies: their ability to avoid data leakage, maintaining the privacy of local

data, the improvement in the use of resources with respect to DML classical methods, the convergence speed opposite to centralized approaches, and so on. However, these advantages come at a cost, and FL algorithms normally face cumbersome challenges, namely:

- Unbalanced data: As it was mentioned when explaining FedAvg algorithm, each of the participants in an FL model holds its own data, leading to skewness in the number of samples, and thus provoking that some users influence the model more than others.
- Communication availability: Training FL models on devices require for them to be connected during the whole training process, which can be hard to achieve when working with smartphones, or certain kinds of machines.
- Data heterogeneity: Local data of each client is highly dependent on the device itself: the conditions it is used into, the context it is surrounded by, and the interaction it has with the environment. For these reasons, local data of a single participant is prone to be unrepresentative of the data distribution across the other participants.

The issue of data heterogeneity has drawn lot of attention from the very beginning. It can cause several grave issues, like slowing the model convergence, or impeding the model from converge at all. Data heterogeneity in this context is commonly named *non-IID* data, standing for *non-independent or identically distributed* data.

4.2.2 Non-IID data across participants

Statistical heterogeneity of the data is a major issue that needs to be faced in order to construct and deploy a FL model [13, 74]. It is a well-known issue that has been referenced multiple times in FL research since its origins. To prevent this obstacle it is common in DML to assume that the data of the different participants is Independent and Identically Distributed (IID) [13, 75, 76]. This expectation is satisfied only if data collected by clients does not present significant differences within each other's. However, in most real-life tasks this assumption does not hold: each participant acts differently, hence collects biased data which contrasts to the one collected by another client.

As a starting point, it is important to have a formal definition of IID data. For that, the data probability distributions of the different data owners need to be settled.

Let $i \in \mathcal{N}$ be a client in the FL model training process, with $\mathcal{N} = \{1, \dots, N\}$ being the set of all clients. Let $d = (x, y) \in X \times Y$ be a data sample. Client i collects its own dataset, denoted $\mathcal{D}_i$, which has a data probability distribution $\mathcal{D}_i(x, y)$. Each data sample $(x, y) \in \mathcal{D}_i$ has probability $P_i(x, y)$. The overall data distribution across all participants is a weighted mean of these local data distributions:

$$\mathcal{D}_G(x, y) = \sum_{i \in \mathcal{N}} \frac{M_i}{M} \cdot \mathcal{D}_i(x, y). \tag{4.2.1}$$

With this notation, IID data can be defined as follows:

Definition 4.2.1. Data is declared to be IID if the probability of a data sample remains the same as other samples are drawn, and a randomly selected sample can belong to any of the clients with the same probability. In mathematical terms, if we consider the union of the datasets $D = \bigcup_{i \in \mathcal{N}} D_i \subset \mathcal{X} \times \mathcal{Y}$, then D is an IID dataset for the FL framework if, and only if,

$$\begin{cases} P_i((x, y), (x', y')) = P_i(x, y) \cdot P_i(x', y'), \\ \mathcal{D}_i(x, y) = \mathcal{D}_j(x, y), \end{cases} \tag{4.2.2}$$

for all $i, j \in \mathcal{N}$, and for all $(x, y), (x', y') \in \mathcal{X} \times \mathcal{Y}$.

An important consequence of this definition is the following: If D is an IID dataset, all the participants distributions are equal, and they are also equal to the global distribution:

$$\mathcal{D}_G(x, y) = \sum_{i \in \mathcal{N}} \frac{M_i}{M} \mathcal{D}_i(x, y) = \mathcal{D}_i(x, y) \sum_{i \in \mathcal{N}} \frac{M_i}{M} = \mathcal{D}_i(x, y), \text{ for all } i \in \mathcal{N}.$$

Reciprocally, we claim D is a non-IID dataset if any of the two conditions given in Equation (4.2.2) does not hold.

Notice that the local dataset of a single client is always IID if its samples are independent. Applying this to a centralized setting, data is always identically distributed, as there is just one device involved in the training process. The term *non-IID* in ML implies the existence of more than one participant, and it is mostly used in the DML paradigm. Non-IID data has been extensively studied in the past few years, and there are lots of works aiming to design novel algorithms to tackle it [1]. They will be detailed and reviewed in Chapter 5.

4.3 Continual Learning

Continual Machine Learning, normally referred to as just Continual Learning (CL) [77–79] is an area of artificial intelligence focused on developing systems that can learn continuously over time, integrating new information without forgetting previously acquired knowledge. This capability is essential for real-world applications where data is not static but arrives sequentially, often under constraints that prevent full retraining. The concept of CL dates back to early work on adaptive systems. In the 1980s and 1990s, researchers began to recognize the limitations of traditional ML models, which typically assumed that training and test data were drawn from the same fixed and static distribution. However, real-world environments are dynamic, requiring models to adapt to new tasks and data over time.

One of the earliest discussions of these limitations appeared in McCloskey and Cohen [80] and Ratcliff [81], who introduced the concept of *catastrophic forgetting*. This term describes the tendency of neural networks to completely or substantially forget previously learned information upon learning new data. This phenomenon occurs because weight updates that help the network learn new tasks can interfere destructively with weights representing older tasks.

Over time, various research communities have developed related frameworks and terminology for continual learning, each emphasizing slightly different goals:

- Incremental Learning: refers to the ability of a model to learn from new data or tasks sequentially without retraining from scratch. This approach emphasizes efficiency and scalability, particularly in environments with memory or computational constraints [82, 83].
- Lifelong Learning: emphasizes the development of agents that accumulate knowledge over time and leverage past experiences to learn new tasks more effectively. Lifelong learning highlights not only memory retention but also knowledge transfer and reuse [84, 85].
- Never-Ending Learning: refers to a paradigm in machine learning where a system continuously learns from data over time, without a predefined endpoint. This approach emphasizes scalability and the open-ended nature of learning without a predefined stopping point [86, 87].

While these terms are often used interchangeably, they reflect subtle differences in emphasis: incremental learning focuses on process and efficiency, lifelong learning

on experience accumulation and transfer, and never-ending learning on open-world knowledge discovery. Regardless of the kind of approach used, a central challenge across all forms of continual learning is *catastrophic forgetting*. As mentioned earlier, when a model learns a new task, it may overwrite parameters important for previous tasks. Addressing this issue has been the focus of extensive research [88–90].

Currently, there are a multiplicity of different strategies to overcome catastrophic forgetting, like regularization-based methods that penalize changing parameters from the model that were important for tasks that have already been learned [91], replay-based methods that involve storing past data to rehearse in old tasks while learning a new one [92], or dynamic architecture methods, which grow the network over time to accommodate new tasks without interfering with prior knowledge. Each of these approaches represents a different strategy for maintaining performance across tasks while supporting the continuous acquisition of new knowledge.

CL strategies have been performed successfully for a wide variety of applications in the last few decades, for instance in healthcare [93,94], robotics [95,96], computer vision [97], and autonomous driving [98], among others.

It is remarkable to mention that, from its origins, CL has traditionally been focusing in centralized settings, and it is only recently that works regarding distributed settings in CL have been deployed. This natural extension of CL enables the development of new applications, and also implies new challenges to be addressed.

Unlike for centralized ML tasks, or for the different participants of the training process in a DML or an FL problem, data is not considered a static set of samples in the context of CL. CL considers the source of data for training to be a stream [99]. A data stream S is a possibly unbounded sequence of data collected over a certain period of time, $[0, t]$, which will be denoted as $S^{[0,t]}$. The major contrast between standard datasets and CL streams is that in the latter, data is collected in a particular order, and normally its tendency varies over time. This can be expressed by saying that $S^{[0,t]}$ is detailed by an unknown distribution $\mathcal{D} \sim (\mathcal{D}^1, \ldots, \mathcal{D}^T)$, where each $\mathcal{D}^i$ represents the distribution of the data samples in the stream over a different task.

The goal of CL models is to be able to solve a set of tasks $(\tau_1, \ldots, \tau_T)$ using the data samples from the stream $S^{[0,t]}$. There are two main challenges in the training process of a CL algorithm: being able to accurately predict the outcome of a feature vector for a previous task, that has already being learned and is no longer present in the new samples drawn from the data stream (face the catastrophic forgetting issue), and also detect the tipping point that separates samples from one task and the next one in the data stream (this point is commonly referred to as concept shift of concept drift).

4.3.1 Concept drift: non-IID data over time

Concept drift is a crucial term regarding CL that has been studied extensively in supervised CL literature [100–103]. The origin of this term relies in a research work that explained how what initially can be considered as noisy data may reveal later, as more samples are drawn, as clear useful information [104]. In a formal way, we will describe concept drift as follows:

Definition 4.3.1 (Concept drift). Let $S^{[0,t]} = \{(x_i, y_i)\}_{i=1}^{M}$ be a data stream of size M collected during the period $[0, t]$. x_i represents the feature vector of a sample and y_i its corresponding label. Let $\mathcal{D}^t(x, y)$ be the probability distribution of $S^{[0,t]}$.

A concept drift occurs at timestamp t if the probability distribution of the data stream is significantly different after that timestamp:

$$\exists\, t > 1 \,:\, \mathcal{D}^{t-1}(x, y) \nsim \mathcal{D}^t(x, y).$$

Concept drift can be analogically defined in terms of the probability density function of the samples, $P^t(x, y)$, instead of their data distribution:

Definition 4.3.2 (Concept drift). Let $S^{[0,t]} = \{(x_i, y_i)\}_{i=1}^{M}$ be a data stream of size M collected during the period $[0, t]$. x_i represents the feature vector of a sample and y_i its corresponding label. Let $P^t(x, y)$ be the probability density function of $S^{[0,t]}$.

A concept drift occurs at timestamp t if the probability density function of the data stream is different after that timestamp:

$$\exists\, t > 1 \,:\, P^{t-1}(x, y) \neq P^t(x, y).$$

Note that definitions 4.3.1 and 4.3.2 are very similar to the non-IID data definition from Section 4.2.2, but referring to distribution or probability variations over time instead of across clients.

Using probability density functions is normally more helpful when analyzing a task, since the joint probability density function $P^t(x, y)$ can be factorized as

$$P^t(x, y) = P^t(x) \cdot P^t(y|x).$$

Research works that experiment and handle concept drift usually categorize it in three different classes, according to which factor of this factorization gets modified [100, 102]:

- Virtual drift: it is caused by a difference in the marginal probability of the features in the samples, $P^{t-1}(x) \neq P^{t}(x)$, and $P^{t-1}(y|x) = P^{t}(y|x)$. In this case, the labels of the samples with respect to the features remain the same, but the input data is statistically new to the model, so it is hard to correctly classify it.

- Real drift: it refers to the opposite situation, where feature vectors keep the same probability but the correct label for them has shifted, $P^{t-1}(y|x) \neq P^{t}(y|x)$, and $P^{t-1}(x) = P^{t}(x)$. Models not aware of this drift will classify the new samples in the same way as before the drift, leading to incorrect predictions.

- Total drift: both the marginal density probability of the samples and their label are significantly different, $P^{t-1}(x) \neq P^{t}(x)$ and $P^{t-1}(y|x) \neq P^{t}(y|x)$, so it is really challenging to adjust the model to reflect both changes.

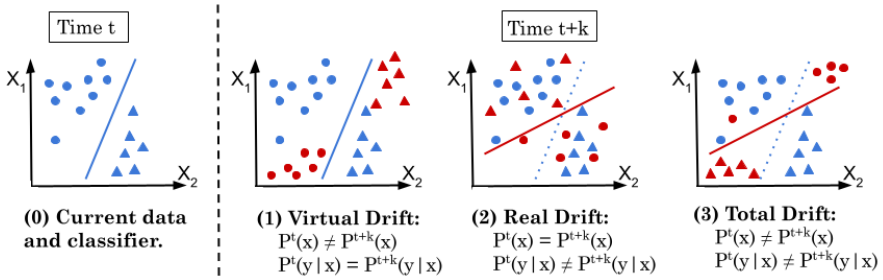


Figure 4.1: Representation of Concept drifts in a two-dimensional input space X with two possible labels $Y = \{\circ, \triangle\}$. On the left of the doted line, we find the data samples received before time $\mathbf{t}$, and on the right there are three possible time-evolving situations. (1): the new data samples observed are situated in new regions, previously unseen. However, data labels correspond with the split made by the classifier previously. (2): the new instances appear in already known regions of the input space, but they are incorrectly classified using the previous model. (3): the two previous situations are combined.

Examples of all of these situations are depicted in Figure 5.4, and the references mentioned there will be further explained in Chapter 5, where there is an extensive

classification of spatial non-IID data (across clients) and temporal non-IID data (over time).

In addition to all of this, the concept drift can be produced in various ways: a drift can be sudden, if a certain timestamp can clearly separate old and new probability distributions, gradual, when the samples belonging to the new concept appear intertwined with samples from the old concept, or incremental, if there is a smooth shift of the data from one concept to another. Furthermore, if several concept drifts occur along time, a concept that was previously outdated may reappear, which is called recurrent concept drift [105–107]. The set of combinations is enormous and complex, and each specific task comes with its own particularities, which must be taken into consideration carefully.

CHAPTER 5

Non-IID data in Federated and Continual Learning

Introduction and Chapter 4 have consisted of a detailed review of the different elements involved in the research constructions that will be presented from here onwards. During the last decades DML and multi-device learning have gained a lot of interest and appeal, since they present clear advantages and adaptability to the context of data nowadays, and the increasing challenges it presents.

The basics of both FL and CL have been discussed, highlighting both the benefits they can provide, and also their difficulties and shortcomings. Precisely regarding these shortcomings, the terms of non-IID data and concept drift have arisen naturally as words crafted for explaining the undesired results initial models in this field have achieved.

In the present chapter, the main objective is to interpret these terms under a common light, formulate a shared ground theory for the data heterogeneity across participants in the FL context and the data heterogeneity over time in the CL context, and simplify the problems they held to the most fundamental, straightforward elements possible so that they become easier to manage and tackle. After that, in the next chapters, different algorithms will be designed to face these kinds of heterogeneity.

In the literature, lots of methods have been presented aiming to face these adversities, for either the FL or the CL paradigm. While most of them present brilliant ideas and very powerful algorithms, in some of them the nature of the heterogeneity they confront is vague and unclear, or unrelated to other similar works created before them. For this reason, this chapter will also focus on establishing similarities between well-

known works in the fields of FL and CL and unify the understanding of heterogeneous samples.

After creating a classification of non-IID data, both throughout devices in FL context and over time for CL settings, some requirements for undertaking each of the situations presented will be exposed, as minimal necessary conditions that should be fulfilled for a problem in each category to be solvable. In case those restrictions are not fulfilled in a real-life scenario, then the models applied to that problem are advocated to produce poorly accurate results.

The most significant contributions of this chapter have been published in the following article, and have been reproduced here under Creative Commons Attribution (CC BY) license: [1] M. F. Criado, F. E. Casado, R. Iglesias, C. V. Regueiro, S. Barro, Non-iid data and continual learning processes in federated learning: A long road ahead, Information Fusion 88 (2022) 263–280.

The Journal Information Fusion had the following Journal Citation Reports (JCR) indicators (2022): Quartile Q1 in the categories *Computer science, theory and methods* (2/111, percentile 1.4) and *Computer science, artificial intelligence* (4/145, percentile 2.4), and an overall Journal Impact Factor (JIF) of 14.8.

The matters of this chapter are disposed as follows: In Section 5.1 the different research works about non-IID data across clients are reviewed and classified extensively, including algorithms and strategies that were not designed for FL originally. Next, in Section 5.2, a similar inspection is done regarding non-IID data over time and CL strategies. Mixing these 2 classifications together to conform the whole variety of non-IID situations, in Section 5.3 a set of restrictions is defined to be able to tackle some of those heterogeneous situations. Finally, some experiments are performed throughout Section 5.4 to show the adversities and poor results algorithms will face if these restrictions are not met.

5.1 State of the art learning strategies for non-IID data in Federated Learning

FL framework is an innovative and recent field within DML, so some of the threats it poses have not been addressed or understood fully. The effect of malicious attacks from a participant of the training process, for instance, have been studied thoughtfully, as it concerns in matters such as data leakage or model convergence [108, 109]. Another difficulty could be the clients' availability, like participants selected to perform the training dropping out, or being incapable of computing a local update due

to factors such as lack of battery or poor connection. To avoid these issues, some specific strategies have been carried out, named Asynchronous FL [110–112]. The main adversity, however, consists of the data heterogeneity [113,114].

Data can be non-IID for many reasons, and in many different ways. For instance, one option is that the clients' local datasets present slight dissimilarities while sharing some characteristics. *Client-level personalization* strategies (Section 5.1.1.1) were designed to handle this situation. A different possibility is that there exists a partition of the participant in the FL process such that each of the groups presents an IID dataset, but when considering the whole sets of data, the mixture of them becomes non-IID. This way of thinking is the reasoning behind the *Group-level personalization* techniques developed that will be further discussed in Section 5.1.1.2.

On the whole, the root cause that makes data non-IID is a significant piece of information to know, that would help deciding the most suitable approach to face it. Digging deeper into the possible motives of disturbances in the joint probabilities is both necessary and useful [13,115]. Like in the case of Concept drift, mentioned in Section 4.3.1, to characterize the nature of non-IID data it happens to be useful to express it in terms of the probability density functions, $P(x, y)$ and $P_i(x, y)$, instead of data distributions, since they can be factorized in two different ways:

$$P(x, y) = P(x) \cdot P(y|x), \tag{5.1.1}$$

$$P(x, y) = P(y) \cdot P(x|y). \tag{5.1.2}$$

With these factorizations, we can better identify which factor represents the devices particularities. However, we are going to focus on the factorization given by Equation (5.1.1), since the conditional probability $P(x|y)$ in Equation (5.1.2) is more controversial and hard to translate into actual real-life situation. This is because of the natural process for training a model, which consists of predicting the label y based on the feature vector x and not the opposite way.

Let $P_i(x, y)$ and $P_j(x, y)$ be the data probabilities of 2 clients, i and j. If those probabilities are the same, then it is clear that both factors need to be equal: $P_i(x) = P_j(x)$ and $P_i(y|x) = P_j(y|x)$. An important clarification to be made here, is that when we assert two probability distributions are equal, in reality what we mean is that they are alike in statistical terms, i.e., a standard hypothesis testing cannot recognize them as different. On the other hand, if the joint probabilities $P_i(x, y)$ and $P_j(x, y)$ are not the same, then according to the previous factorization three possible scenarios can be acknowledged:

i) $P_i(x) \neq P_j(x)$ and $P_i(y|x) = P_j(y|x)$. In a situation like this, where the conditional probabilities of labels with respect to feature vectors match, the goal of both participants is the same, and the way of labeling data is shared. In this scenario, if a feature sample x was collected by either participant i or j, it would have the same label associated. The source of heterogeneity in this case comes from the features themselves: clients own data samples from different domains. An example of this would be training an autonomous car. In certain countries people drive on the right, whereas in other people drive on the left. This way of acting will make the input spaces of the users skew. Nonetheless, in both cases data is gathered with one common objective, and they would act similarly with respect to the general traffic signs.

ii) $P_i(x) = P_j(x)$ and $P_i(y|x) \neq P_j(y|x)$. This sort of scenario takes place when the input spaces that the participants perceive are similar to each other, but the response to that input space is not the same. If a feature sample x was collected by user i, it would have one label, but if it was collected by user j it would probably have a different label. There is a direct conflict with the problem understanding. This could occur again using the task of training an autonomous car, when users encounter a yellow traffic light. In this situation, the correct output for some clients would be to slow the car and stop until it is green again, but for others it would be to continue driving without changes.

iii) $P_i(x) \neq P_j(x)$ and $P_i(y|x) \neq P_j(y|x)$. This casuistic combines the two previous scenes. In this case, users want to train an FL model to perform a shared task, but they perceive significantly unequal data, making their input spaces different, and at the same time they would react in particular ways, or classify their data in dissimilar manners.

To sum up, this provides us with a total of four situations to account for. They are represented in Table 5.1, as well as a collection of works that deal with each of the possible situations. Lots of techniques could fit the cell of IID data, such as the foundational paper FedAvg [17]. However, we will not consider that situation, and we will focus only on the non-IID scenarios. In general, works that aim to solve heterogeneous problems do not pause to consider the scenario they are tackling, nor do they provide some clarification on the types of non-IID data that could be present in such problems. Nonetheless, they are located in Table 5.1 based on the type of non-IID data situation that they are solving, or that they can solve. For all of these

reasons, we have considered two possible classifications of the FL research regarding non-IID data.

Variations in the clients datasets			
No changes (IID data)	Changes in the input space throughout clients $P_i(x) \neq P_j(x)$	Changes in the behaviour throughout clients $P_i(y/x) \neq P_j(y/x)$	Changes in the input space and behaviour throughout clients
OUT OF SCOPE	**[117—123, 129] [141, 146—166]**	**[124—127] [185—187]**	**[16, 128] [136—138]**

Table 5.1: Non-IID learning scenarios in FL, and currently existing strategies that could potentially solve each situation.

One way for classifying non-IID strategies in FL is regarding the level of heterogeneity they consider (on clients or in groups). The majority of FL works that deal with non-IID data explain their approaches as a *Personalization* strategy to increase the accuracy of the model. Personalization can be considered from different angles. In Section 5.1.1 we briefly explain them. Note that, although all of these strategies deal with data heterogeneity, they are unaware of the source of data heterogeneity itself, meaning they do not consider the probability density function or which of its factors is being affected.

Another way for classifying strategies in the Non-IID context, is regarding the factor of $P(x, y)$ decomposition where the data heterogeneity relies on, and classify them using that knowledge. Now that we have explained what non-IID data is, and what subcategories can be identified, this approach seems reasonable. In addition, a classification like this would encompass the previous one, and at the same time it allows us to consider techniques from ML that deal with data heterogeneity but are not strictly using a FL framework. These techniques will be described in Section 5.1.2.

5.1.1 Non-IID data in Federated Learning: Personalization strategies

Let's focus on the first classification we mentioned: Personalization. This technique consists of balancing the general knowledge achieved by the global model obtained with an FL algorithm, taking into account the specificities each client presents. Research in this area tend to leverage the importance of individual particularities instead

of aiming for a unique global model. This is because it is empirically proven that in realistic settings a single global model may not be suitable for all clients at once, performing poorly in some of them [116]. Moreover, clients may have information that contradicts with each other. Using this reasoning of personalization, we must be opened to the idea of computing more than one global model. Personalization postulates itself as an intermediate arrangement between having a common model for all clients, or having a different model for each of them, considering the opportunity of designing a model that absorbs the general knowledge but in addition recognizes uncommon pieces of information.

There are different levels of personalization: using the global model as a baseline for all the participants, it can be refined later individually so that each participant could have its own model. We will refer to this type of approach as *Client-level Personalization*. On the contrary, there could be clusters of participants using a common model, i.e., *Group-level Personalization*. Both of these options have their own strong points, and both of them can accomplish an improvement in the model accuracy. As a drawback for all of them, their computational requirements are usually more demanding since they add new computations in standard FL algorithms.

5.1.1.1 Client-level personalization

Client-level personalization encompasses methods that enable each individual participant to derive a unique model following the training phase. In the context of personalizing a global model trained within the FL environment, a wide array of strategies and concepts have been explored. These personalization techniques can broadly be categorized into two main groups:

A) Firstly, we encounter traditional ML techniques adapted to the FL paradigm with the goal of enhancing the global model, such as Transfer Learning and Multi-task Learning.

B) Secondly, there are methods that focus on concurrently training both global and local models. After obtaining the global model, each client merges it with their private model, trained in tandem, to produce a personalized version.

The first category (*A*) covers solutions like applying Transfer Learning to adjust pre-trained models using public datasets across a range of devices [117, 118]. Other approaches within this category focus on learning a shared data representation at each

client, which can enhance robustness to domain discrepancies and improve the performance of the global model under non-IID data distributions [119]. Regularization-based techniques also fall into this group, where additional penalty terms are introduced into the loss function. For example, *pFedMe* [120] includes a term that discourages excessive divergence between client updates. The loss function in this case forms a Moreau envelope, a mathematical structure that facilitates convergence guarantees and deeper accuracy analysis. Similarly, *pFedAtt* [121] incorporates a regularization element that emphasizes alignment between similar updates, effectively clustering related clients to support convergence in non-IID environments with differing input distributions.

Another notable line of work involves adapting algorithms from Model-Agnostic Meta-Learning (MAML), such as *Reptile*, for federated settings [122, 123]. These methods aim to produce a model that quickly adapts to varying tasks by favoring more generalized data representations. Some researchers also treat the heterogeneity across devices as an indication that clients are solving distinct tasks [124, 125], and apply Multi-task Learning principles within the FL framework. A representative example is MOCHA [125]. Collectively, these approaches offer different interpretations of personalization by leveraging and reimagining well-established techniques.

Regarding the second class of strategies (*B*), they concentrate on maintaining dual-model training for each client. One model is collaboratively built across clients using typical federated procedures, while the other remains private and serves to fine-tune or personalize the final model outcome. These methods differ in both how the global model is formed and how the local model is incorporated. For instance, in [126, 127], clients train a Deep Neural Network (DNN) where the last few layers are kept private and trained individually. The shared layers represent the global model, whereas the private layers act as a personalization mechanism, enabling different outputs even for similar inputs. In [128], training follows a probabilistic process: participants perform local SGD with a predefined probability $p \in (0, 1)$. On the other hand, with complementary probability $1 - p$, they transmit their local models to the server for aggregation. Unlike traditional FL approaches, the global model is not continuously updated, but in the end each client gets a unique model adapted to their data, and it is empirically shown that the accuracy of the models is higher.

Another notable technique, *FedProx* [129], is a variant of FedAvg that allows for controlled divergence in local updates and tracks their dissimilarity throughout training. Once training concludes, clients receive the global model and apply minor modifications based on their update history. Finally, in [18], participants concurrently train both a global and a local model during each communication round. Upon completion

of training, each participant adjusts the received global model using their personalized local model.

Empirical findings across these approaches demonstrate impressive performance gains. For example, [119] shows that their method maintains strong performance on a rotated MNIST dataset, whereas the standard *FedAvg* suffers a considerable drop in accuracy. In [122], the proposed meta-learning method outperforms both a fine-tuning baseline [130] and a k-nearest neighbor classifier [131], achieving over 10% higher accuracy. In another case, [126] utilizes *MobileNet-v1* and *ResNet-34* with personalized layers, testing them on CIFAR-100 and FLICKR-AES [132] under non-IID splits. Their personalized models reach 80% accuracy, significantly outperforming the 60% and 40% achieved by standard FedAvg on CIFAR-100 and FLICKR-AES respectively. Similarly, [129] compares FedProx against FedAvg using the MNIST [133] and Shakespeare datasets. While FedAvg shows slightly better results in the IID scenario, FedProx excels in non-IID conditions and improves convergence in both. Lastly, [18] evaluates their APFL approach on MNIST, EMNIST [134], and CIFAR-10 [135], achieving 89% accuracy in a non-IID context, compared to 32% for FedAvg and 83% for FedAvg with fine-tuning.

5.1.1.2 Group-level personalization

Unlike strategies that aim to produce a unique model for every client, another option is referred to as *Group-level personalization*, which involves organizing clients into clusters and training a separate model for each group. This research direction has only recently begun to gain traction, meaning that it's still evolving and likely has untapped potential. As a result, there are currently only a handful of methodologies available, and all are quite recent.

The central challenge in this domain is determining how to form the client groups in a way that maximizes the benefit of model sharing within each cluster. One group of methods relies on hierarchical clustering techniques to segment the clients [136–138]. These approaches typically use a distance metric based on the clients' weight updates to either merge [136, 137] or split [138] them into clusters. However, there's no formal assurance that clients with similar updates will actually gain from sharing a model. It's possible for clients with very different data distributions to produce similar updates, and likewise, clients generating different updates might still benefit from collaboration. The only defensible claim is that clients with similar update patterns may reach convergence more quickly, though this does not necessarily translate into better model performance.

Another line of research into group-level personalization focuses on the discrepancies between the global data distribution and the local distributions across clients. The key idea in these works [139–141] is that when client data is non-IID, a single global model cannot effectively generalize to each client's data. In such cases, the global distribution might fail to capture the particular characteristics of some clients, suggesting that it should not guide the model training process. These studies propose a method for identifying a global distribution D_Λ that better represents the collective data landscape across clients. Notably, this distribution does not have to match the traditional weighted average distribution (Equation 4.2.1). Clients are then grouped based on private attributes related to D_Λ, rather than their local model updates. This clustering technique is supported by theoretical guarantees, ensuring that organizing clients this way leads to a more beneficial model outcome.

As for experimental evaluations, the methods in [136–138] are tested on common benchmarks such as MNIST [133], FEMNIST (Federated Extended MNIST), and CIFAR-100 [135]. These datasets are partitioned among clients, and label permutations are introduced for some clients to create diverse label distributions, $P(y|x)$. These approaches outperform standard FedAvg in both accuracy and convergence rate. However, they are only compared against FedAvg, which is primarily designed for IID data scenarios. In contrast, [139] demonstrates improved performance over FedAvg in digit classification using the MNIST dataset within a centralized setup. Further evaluations in [140, 141] utilize decentralized frameworks and test on Fashion MNIST [142] and EMNIST (Extended MNIST) [134], achieving accuracy on par with FedAvg. Overall, the standout advantage of group-level personalization methods so far has been their faster convergence. The datasets used across these studies are summarized in Table 5.2.

Article	Datasets used in experiments		
[18]	MNIST*;	CIFAR-10*;	EMNIST.
[117]	MNIST;	CIFAR-10.	
[122]	MiniImageNet;	Omniglot.	
[123]	MNIST*;	CIFAR-10*.	
[124]	CASAS.		
[126]	CIFAR-100;	FLICKR-AES.	
[127]	OTB;	VOT2014	
[129]	MNIST;	FEMNIST;	Shakespeare.
[136]	MNIST*;	CIFAR-100.*	
[137]	MNIST*;	FEMNIST.	
[140]	Fashion MNIST;	EMNIST.	
[141]	Fashion MNIST;	EMNIST.	

Table 5.2: Summary of the datasets employed in the works presented in Section 5.1.1. Datasets marked with an asterisk are modified in different ways, making it impossible to fairly compare each other. Some of the datasets mentioned were not referenced so far: Fashion MNIST [142], Omniglot [143], OTB [144], VOT2014 [145], and Shakespeare [11].

5.1.2 Statistical taxonomy for non-IID strategies

Now that we are familiarized with the algorithms for personalization in FL, it is time to deepen into the other classification mentioned in Section 5.1. Recall that this classification is based purely on the kind of heterogeneity faced by each of the methods proposed. Going back to Table 5.1, we distinguish 2 types of proposals: The ones that deal with changes in $P(x)$, i.e., changes in the input space across clients; and the ones that deal with changes in $P(y|x)$, i.e., changes in the behavior across clients.

5.1.2.1 Changes in the input space throughout clients

Devices can collect data from their particular input domain. Nonetheless, as in the previous sections, we will assume that the domain of each device remains unchanged in the training process, i.e., data will be IID over time. We will explore this particular

topic in Section 5.2. We already established that participants collect data in a separate way from others, and so it is unknown if their input domains are the same. If they are not, we encounter the most studied type of non-IID data by far: non-IID data across clients in their feature domain. Strategies designed to tackle this issue are countless, and pay attention to a variety of realistic problems that can be encountered. We will consider here centralized ML and DML methods that do not necessarily fit into the FL field, but are prone to become FL algorithms through some adaptations. These works are split into 2 different branches: *(i) Domain Transformation* and *(ii) Domain Adaptation*. Both of them, at the same time, also branch into different approaches (see Figure 5.1).

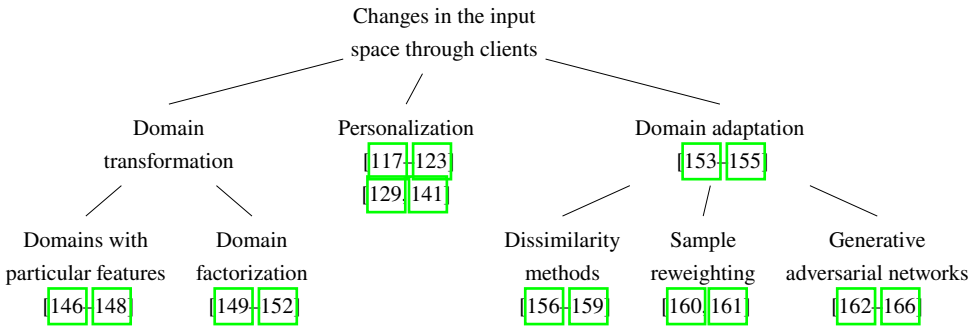


Figure 5.1: Classification of the different approaches that are able to solve the problem of spatial heterogeneity in the input spaces.

Regarding the first category, Domain Transformation methods aim to identify both the unique characteristics and the shared components of data domains, with the goal of constructing a unified input space. After identifying these elements, the individual input spaces are mapped into a common representation. To the best of our knowledge, this technique has only been implemented in centralized systems so far. Nevertheless, it is conceivable that in a decentralized context, each client could independently compute its own transformation, allowing the central server to synthesize a global shared space from the locally estimated ones.

A major obstacle in applying this approach lies in the high dimensionality typical of feature spaces in real-world problems. This often leads to increased computational demands and issues stemming from the curse of dimensionality [167]. For instance,works like [146–148] consider scenarios where each domain possesses a dis-

tinct feature set for representing samples, resulting in cross-domain incompatibilities. They propose methods to derive a unified feature representation. Another strategy, explored in [149–152], focuses on decomposing the feature space with specific structural properties. In particular, [149, 150] partition the space into two orthogonal components: one capturing domain-specific variations and the other representing shared features, which are then utilized separately during the learning process. Meanwhile, [151, 152] divide the input space into multiple low-dimensional subspaces and apply Distance Metric Learning [168, 169] techniques within each of those subspaces.

In contrast, Domain Adaptation methods [153–155] address a different scenario, one that is closely related to Transfer Learning and typically explored within centralized architectures. However, certain studies in this line also consider Federated Transfer Learning [170] within FL environments. Here, clients train models on data from specific domains, which are subsequently used to infer outcomes on other, previously unseen, domains. In this context, training data is drawn from what are known as *Source Domains*, whereas inference is performed on *Target Domains*. A key distinction from Domain Transformation methods is that no training data samples from the Target Domains are available, making it infeasible to directly build an appropriate feature space for them. Other works in this area [156–159] introduce diverse strategies for quantifying the discrepancy between source and target distributions, such as Maximum Mean Discrepancy (MMD) [156] or Moment Matching for Multi-source Domain Adaptation (M^3SDA) [157], and fine-tune the model using unsupervised techniques [158, 159].

Another route for aligning domain distributions involves calculating distance metrics between them and using this information to reweigh data samples, giving preference to those from similar distributions to enhance model performance. For example, [160, 161] use the Kullback-Leibler divergence [50], a widely used information-theoretic distance metric, to implement this reweighting strategy. The Kullback-Leibler divergence is closely related to the cross-entropy definition given in Section 4.1.1 and also to the cross-entropy loss function.

A particularly promising family of Domain Adaptation techniques leverages Generative Adversarial Networks (GANs) [162–166]. These methods train two neural networks simultaneously: one that generates synthetic data across different domains, and another that distinguishes real data from generated samples. The work in [166] stands out for applying GAN-based Domain Adaptation in federated learning settings. These GAN-based methods are often benchmarked against leading approaches such as Deep Adaptation Network (DAN) [171], Deep Domain Confusion (DDC) [172], and Residual Transfer Network (RTN) [173]. GANs consistently outperform these

methods in various benchmarks, including image recognition tasks using the Office-31 [174], Office-Home [175], and VisDA2017 [176] datasets, as well as digit classification with MNIST [133], USPS [177], and SVHN [178].

Article	Datasets used in experiments		
[146]	MNIST + SVHN + USPS.		
[148]	Office-31;	Bing-caltech256.	
[151]	COREL5000;	Trecvid2005[1].	
[152]	MNIST*;	Olivetti FR[2].	
[154]	MNIST + SVHN.		
[155]	Office-31;	ImageNet;	VisDA2017.
[156]	Office-31;	Image CLEF-DA.	
[157]	Digit5;	Office-31.	
[158]	MNIST + MNIST-M + USPS;	VisDA2017.	
[162]	MNIST + SVHN + USPS;	Office-31;	NYUD.
[163]	Office-31;	Image CLEF-DA[3].	
[164]	Office-Home;	VisDA2017.	
[165]	MNIST + SVHN + USPS;	Office-31.	

Table 5.3: Summary of the Datasets employed in Section 5.1.2.1. Asterisks indicate that the datasets have been modified in particular ways, making it impossible to fairly compare each other. Some of the datasets mentioned were not referenced so far: Bing-caltech256 [179], COREL5000 [180], ImageNet [181] and NYUD [182].

In general, drawing direct comparisons among these various approaches is difficult. Each study often employs distinct synthetic datasets, tailored and manipulated to highlight specific forms of heterogeneity (refer to Table 5.3). As such, a standardized comparison across all methods is currently infeasible. Nevertheless, several works report standout findings. In the Domain Transformation category, [148, 152] provide comprehensive evaluations and demonstrate significant improvements in accuracy and error reduction when compared to baseline methods. Among Domain Adaptation methods, the most notable outcomes are reported in [158, 164, 165]. Specifi-

[1] Available at http://www-nlpir.nist.gov/projects/trecvid

[2] Available at http://www.uk.research.att.com/facedatabase.html.

[3] Available at http://imageclef.org/2014/adaptation

cally, [158] introduces the SimNet architecture and evaluates it against DAN, RTN, and a simple baseline across MNIST, Office-31, and VisDA2017, consistently achieving superior accuracy. Similarly, [164, 165] both use the Office-31 dataset and show strong performance improvements over competing approaches.

Beyond the methods discussed so far, other innovative techniques have also been proposed to manage domain shifts. For instance, [183] explores the use of Source and Target Domains by factorizing the input space and identifying a Grassmann Manifold that encompasses all data samples. Training is then conducted on this manifold rather than the original high-dimensional feature space. Finally, several federated personalization techniques described earlier in Section 5.1.1 are also applicable to the domain heterogeneity challenges highlighted here [117, 118, 122, 123, 129, 141].

5.1.2.2 Changes in the behavior throughout clients

Variations in the behavior of clients pertain to inconsistencies in their conditional distributions $P(y|x)$. A divergence of this sort indicates that, for at least some data instances, the appropriate output is not identical across all clients. Next, we present a formal definition of this phenomenon:

Definition 5.1.1. Two clients i, j have different conditional probabilities $P(y|x)$ if there are at least T data samples $\{x_k\}_{k=1}^{T}$ and T outputs $\{y_{k_1}\}_{k_1=1}^{T}$ and $\{y_{k_2}\}_{k_2=1}^{T}$, such that the data samples (x_k, y_{k_1}) and (x_k, y_{k_2}) are owned by participants i, j respectively, $\forall\, k, k_1, k_2 \in \{1, \ldots, T\}$.

To clarify, we must define the notion of "distinct outputs": y_{k_1}, y_{k_2} are considered different if $||y_{k_1} - y_{k_2}|| > 2L$ for a predefined tolerance threshold L (refer to Figure 5.2), which may fluctuate based on the nature of the problem. In classification tasks, $2L$ should be less than 1, implying the margin must be below $1/2$, whereas in regression scenarios, the tolerance is determined by the required precision level.

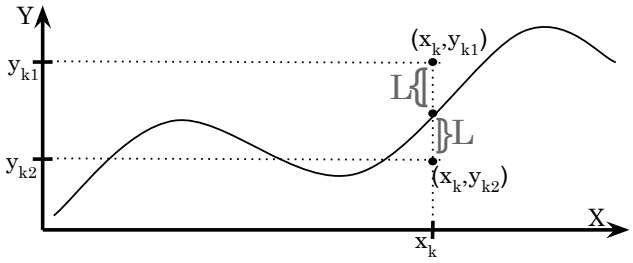


Figure 5.2: One variable regression model. Samples (x_k, y_{k_1}) and (x_k, y_{k_2}) are owned by different clients. If those samples are more than $2L$ units apart, there would not exist any possible output that is simultaneously closer than L units apart for both of them. For this reason, at most one of the outputs would be considered correct.

The principal challenge in this context is that, contrary to prior situations, a single unified model cannot accommodate all user behaviors, as it is incapable of producing different outputs for the same input across clients. Recall that, in ML, a model is formally defined as a function $f : \mathcal{X} \longrightarrow \mathcal{Y}$ that maps each input to a single output [184]. For a given input x_k, a conventional model in a decentralized environment processes it uniformly for every client, yielding a single, universal output. Consequently, the predicted result might be y_{k_1}, y_{k_2}, or another value, most probably between the two, but inevitably there will at least one client that will experience a prediction error exceeding L. Moreover, identifying the presence of varying behaviors among clients necessitates analyzing their respective loss functions, which in turn requires access to a sufficient quantity of labeled data.

To address this issue, it becomes crucial to adopt a model design that allows for individualized variations across users. That is, enabling each client $i \in \mathcal{N}$ to utilize a partially personalized model $\mathbf{w}_i$, differing from others. Several personalization methods explored in Section 5.1.1 can be adapted to handle behavioral heterogeneity (see Figure 5.3). For example, having an appropriate error metric would facilitate the use of *Group-level personalization* techniques, which categorize participants according to their behavioral similarities. This methodology is closely aligned with Cohort-based Federated Learning [185], which explicitly forms clusters of clients with comparable data characteristics.

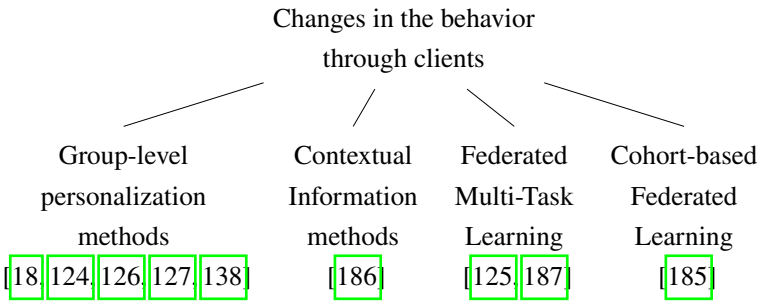


Figure 5.3: Classification of the different approaches that deal with the spatial heterogeneity in the behaviors of clients.

Only a limited number of techniques are explicitly crafted to manage this specific type of non-IID data. One of the most relevant solutions, though not directly framed in the context of FL, is the one introduced in [186]. This method involves appending extra information to the input, such as a task label z. This enables two highly similar inputs x_{k_1}, x_{k_2} to be treated distinctly: $(x_{k_1}, z_1), (x_{k_2}, z_2)$.

Finally, additional research lies at the intersection of FL and Multi-task Learning [125, 187]. The first was previously covered in Section 5.1.1. The latter introduces a multi-task structure merged with the federated framework. It evaluates its approach on the MNIST, EMNIST, and Shakespeare datasets and benchmarks against prominent federated learning techniques such as FedAvg and FedProx.

5.2 State of the art learning strategies for non-IID data in Continual Learning

In this part, we will explore the concept of *Continual Learning (CL)*, which deals with training models incrementally over time. In the conventional ML framework, the goal is to develop a predictive model based on a fixed dataset.

An important aspect to highlight is that training data is generally assumed to be entirely available from the beginning, a premise that often clashes with real-world scenarios, where information is gathered gradually and may evolve. For this reason, it is beneficial to consider CL, a learning paradigm where models are updated continuously with new streams of data, while attempting to retain previously acquired

knowledge. As mentioned in Section 4.3 , this paradigm has been referred to by different terms over time [77], including *Lifelong Learning* [188, 189], *Never Ending Learning* [87, 190], and *Incremental Learning* [97, 191, 192], but all share the same core principle: progressively training a model with data obtained across various time intervals, adapting to novel inputs while attempting to preserve prior insights.

We bring up the CL framework here because we intend to address the temporally dynamic nature of FL problems. Nevertheless, throughout this section, we will reference and briefly summarize studies centered on CL that may not explicitly operate within the FL paradigm. This is due to the fact, as mentioned earlier, that very few studies address FL and CL in combination [193–195]. Still, the methods we review are, in our view, among the most readily transferable to the FL context, where numerous devices cooperate to train a unified global model. We will elaborate on how each method could be adapted to this setting in the corresponding discussions.

Applying CL approaches to model training introduces specific challenges, many of which have already been investigated in recent publications. The most prominent difficulties, as in the case of FL, are tied to the distribution of data. CL has traditionally been framed as a centralized ML approach, so while device-level non-IID data has not been a focus, the data itself can change over time. This presents a challenge, as the model may struggle to converge if the training distribution is in constant flux. Another problematic issue, known as *catastrophic forgetting*, occurs when the model abruptly loses previously learned information if it is no longer present in recent data [196,197]. For this reason, we will concentrate on understanding how data evolves over time and how to respond when these changes are abrupt and unpredictable, a phenomenon commonly referred to as *concept drift* [77, 101].

5.2.1 Concept drift detection

The non-stationary data distribution is caused by changes in data over time. These changes can be seen as variations in the frequencies certain kinds of data appear: a concept has frequency zero if it has not appeared yet in the dataset, and when it shows up its frequency becomes a positive number. This kind of variation, called *concept drift*, is one of the most important CL challenges [77, 105]. We have already discussed it deeply in Section 4.3.1, so here we will directly tackle the possible strategies to face it, and the basic steps needed for it.

When trying to deal with concept drifts, one should notice that not all of them are alike, as data can evolve in multiple ways. Similar to what occurred with non-IID data in FL, it is important to characterize concept drifts to distinguish them. However, in

the case of concept drifts, most of the existing works present a common ground, and base their classification according to which factor from the equation $P(x, y) = P(x) \cdot P(y|x)$ is altered. According to this criteria, we determine three types of shift [100, 192]: (1) virtual, (2) real, and (3) total (see Fig. 5.4):

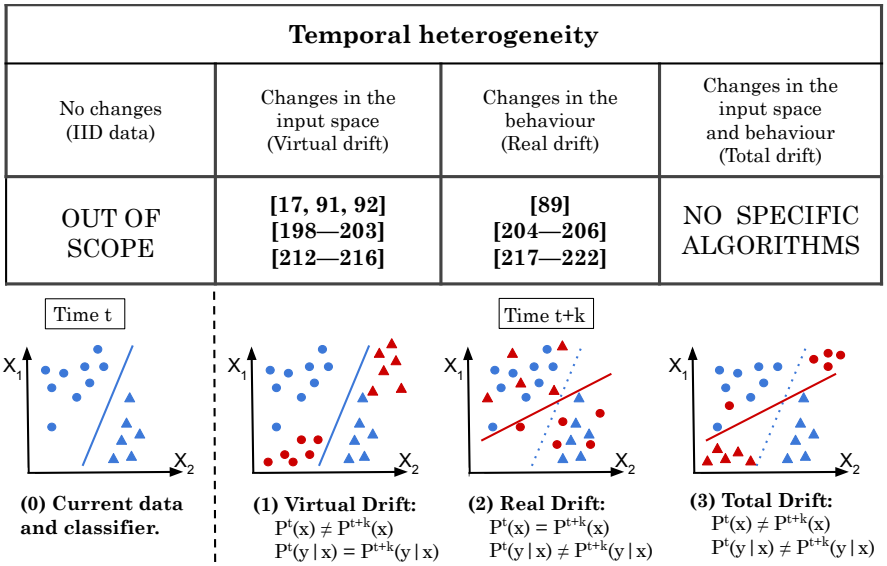


Figure 5.4: Representation of Concept drifts in a two-dimensional input space X with two possible labels $Y = \{\circ, \triangle\}$. Representation of Concept drifts in a two-dimensional input space X with two possible labels $Y = \{\circ, \triangle\}$. This representation is the same as in Figure 4.1, but here the corresponding references of research works in each of the different kinds of concept drift is detailed.

Once we have established a clear understanding of what concept drift entails, we can explore the various techniques proposed to manage it. These approaches are usually divided into three stages. Initially, they must identify changes in the data distribution. Following that, they need to respond appropriately to the identified shifts, ensuring the model adapts to the updated data conditions. Lastly, interpreting the nature of the drift and its potential impact on future model training becomes essential. In this section, we will focus solely on detection mechanisms. The methodologies used to address these changes after detection are analyzed in Section 5.2.2.

A wide range of concept drift detection techniques have been introduced to tackle both virtual and real drift scenarios [2, 198–206]. These methods are generally divided into two broad families: *Data Distribution-based* and *Error Rate-based*. They analyze distinct statistical attributes of the data's input and output to pinpoint moments of distributional change. The majority of these methods are constructed under the assumption that the data resides on a single centralized system. To the best of our knowledge, only [2, 203] have introduced detection techniques tailored for federated learning settings. Nevertheless, we argue that the approaches highlighted in this work are, in principle, adaptable to FL scenarios.

Data Distribution-based strategies are tailored to detect virtual concept drift. To identify this type of drift, one only needs access to the input data patterns, $\{x_i\}_{i=1}^{M}$, or suitable representations of them. As an illustration, the method in [198] directly analyses the input by assessing similarities between features, clustering them, and then checking how new data points are distributed among those clusters to infer a drift. In a federated learning scenario, this method could be localized: each device could independently compute clusters and monitor its own input stream, later sending any drift reports to a central server tasked with managing global decisions.

Alternatively, [199] employs a transformed version of the input data. They define a function $f : X \subset \mathbb{R}^m \to -1, 1$ that divides the dataset into two categories and compare distributions from different time periods to detect changes. If the allocation of data across the two groups significantly shifts over time, this is taken as a sign of drift. For accurate results, the mapping f must satisfy certain properties. One practical way to build such a mapping is through a linear transformation L, followed by a thresholding step based on a constant $A \in \mathbb{R}$. Specifically:

$$
\begin{aligned}
L : X &\longrightarrow \mathbb{R}^m \\
x &\longmapsto w^T x, \quad x \in X, \;\; w \in [0,1]^m. \\
f(x) &= \begin{cases} 1 & \text{If } \|L(x)\| \geqslant A \\ -1 & \text{Otherwise} \end{cases}
\end{aligned}
$$

This design ensures the function meets the required criteria. The threshold A is chosen based on the feature value distribution. Though alternative mappings may also work well, this one is among the most straightforward. To use such methods in federated learning, it is crucial that all participants apply the same mapping f to maintain consistent detection sensitivity across devices.

Another widely adopted approach to detect virtual drift involves employing sliding windows to track historical and current data for comparison [200–202]. This class

of methods partitions the dataset according to the collection time and then contrasts the two partitions using statistical features such as mean, variance, or distances within and between groups. To ensure coherence across devices, it is essential that all clients use the same metric or statistical test. However, synchronization of time splits is not required since drift might occur at different moments for each client. This is the approach employed in [2] within a federated setup.

The other study proposing a method for virtual drift detection in FL settings is [203]. This technique begins by assuming the absence of drift during an initial phase of training, during which clients' updates are statistically profiled. These profiles are then compared against subsequent client updates, and any significant deviations are interpreted as evidence of drift.

Turning now to *Error Rate-based methods*, these focus on identifying real concept drift and tend to be more complex than virtual drift detection. One key distinction is that virtual drift methods often function without label data, while identifying real drift requires labeled examples, as it hinges on detecting changes in the conditional distribution $P(y|x)$ through prediction error. Certain strategies [204, 205] also use sliding windows in this context, but with performance metrics that prioritize label-related information. When the model's error rate experiences a marked increase, a real drift is inferred. Therefore, the effectiveness of these methods depends strongly on how model performance is evaluated [206].

There are multiple ways to quantify model errors. One of the most widely adopted metrics in classification tasks is the cross-entropy loss. Considerable effort has gone into analyzing whether it reliably reflects actual prediction error [207, 208]. Moreover, various modifications and extensions of cross-entropy have been suggested [209–211]. While these alternatives have shown promising results, they still suffer from certain drawbacks, such as vulnerability to imbalanced data or an inability to trace error sources. For the purpose of detecting real drift, having loss metrics that offer traceability and robustness is especially valuable.

5.2.2 Concept Drift adaptation

After detecting a concept drift, the next step is to respond appropriately to it. In the next subsections, we will explore available solutions for managing temporal non-IID data, categorize these methods according to the already mentioned types of concept drift 4.3.1, and, where applicable, compare their empirical performance.

5.2.2.1 Virtual Concept Drifts

Virtual drift detection relies solely on examining shifts in input data distributions. Correspondingly, countermeasures designed to maintain model accuracy focus exclusively on the inputs. These techniques are generally categorized into *Memory-based* and *Regularization-based* approaches, see Figure 5.5.

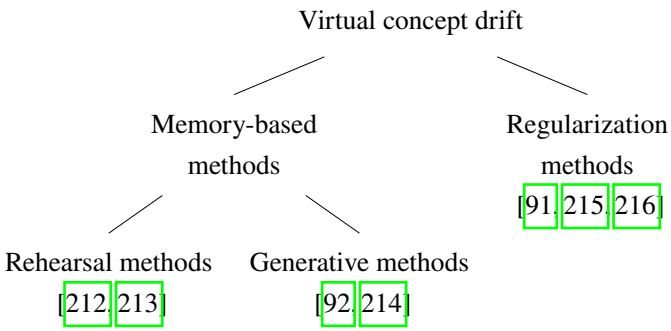


Figure 5.5: Classification of the different techniques able to deal with the temporal heterogeneity in the input space of data.

Memory-based approaches [92, 212–214] emphasize maintaining a selection of past examples, enabling the model to be trained on both new and stored data, thereby mitigating forgetting. In [212], the proposed CLEAR technique stores past training examples and reuses them during later updates. This significantly helps in maintaining performance across time. The ER method introduced in [213] adopts a similar tactic, albeit with smaller memory buffers, reused frequently alongside new samples. They report no significant negative impact on generalization or overfitting.

Other variants within this category include generative models that create synthetic examples based on past data [92, 214]. This technique aims to sidestep overfitting to stored samples. For instance, [92] utilizes a GAN to synthesize new data points after drifts occur. However, concurrently training both the GAN and the main model is computationally demanding. To address this, [214] proposes embedding the generative capabilities directly into the primary model using feedback connections.

In contrast, *Regularization methods* [91, 215, 216] restrict how the model's parameters evolve, effectively "freezing" weights important for prior tasks. The EWC method introduced in [91] evaluates the importance of weights using the Fisher In-

formation matrix and discourages changes to those weights during subsequent training. While effective, it faces scalability challenges as tasks accumulate. A follow-up by [215] refines EWC to reduce computational overhead. Likewise, [216] approximates the weight importance via a Laplacian, aiming for more efficient updates.

Article	Datasets used in experiments
[91]	MNIST*
[92]	MNIST + SVHN
[213]	MNIST* + CIFAR-10*; CUBS
[214]	MNIST*
[216]	MNIST* + SVHN + CIFAR-10

Table 5.4: Summary of the datasets employed in the works presented in Section 5.2.2.1. Asterisks indicate that the datasets have been modified in particular ways, making it impossible to fairly compare each other.

Regarding the reported experimental outcomes, we observe a recurring issue: the considerable variation among datasets (refer to Table 5.4) hinders consistent comparison across different methodologies. Despite this, the majority of the works previously discussed evaluate their proposals against EWC [91], which serves as a common benchmark. In [91], the authors utilize the MNIST dataset and apply a random pixel permutation across all images to simulate multiple input distributions. This method results in what is known as Permuted MNIST, allowing the simulation of numerous domains from a single dataset. In [215], the P&C approach is assessed alongside EWC, both per task and on average, demonstrating that P&C slightly surpasses EWC by mitigating catastrophic forgetting. A similar trend appears in [212], where the CLEAR algorithm is evaluated against both EWC and P&C, consistently outperforming EWC and achieving results close to those of P&C. Additional studies such as [213, 216] also adopt Permuted MNIST, managing to improve EWC's mean accuracy and outperform several alternative methods. Finally, generative approaches like the one proposed in [92] have shown effectiveness in addressing catastrophic forgetting, with their experiments on MNIST and SVHN indicating minimal performance drops when switching datasets.

5.2.2.2 Real Concept Drifts

This scenario considers cases in which users may change their tasks throughout the training period. Previously, in Section 5.1.2.2, we emphasized the importance of enabling different clients to produce varying outputs from similar inputs. Here, we instead aim to model systems where a single client may require different outputs at different times. Addressing this temporal variability typically involves one of two main strategies: *Contextual Information Methods* and *Architecture-based Methods*, as illustrated in Figure 5.6.

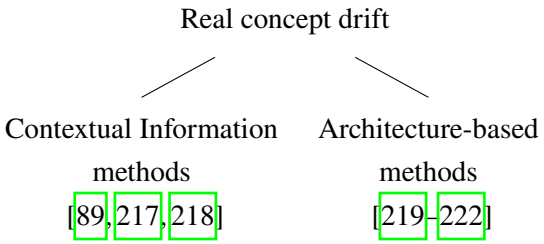


Figure 5.6: Taxonomy of techniques addressing temporal diversity in data behavior.

The *Contextual Information methods* closely align with the concepts introduced in Section 5.1.2.2 [186], where an auxiliary variable z is added to each input, representing a task or domain label. This extra context is intended to resolve both multi-task and multi-domain problems in various settings, not limited to the one originally described. Relevant works include [89, 217, 218]. In [89], the authors argue that task-related information influences learning and introduce task identifiers for personalization. Similarly, [217, 218] make use of contextual cues to map data samples to specific tasks, functioning effectively as task-specific subnetworks. When a new task emerges during training, these models expand their layers, dedicating new neurons exclusively to the new task, thereby avoiding interference with previous tasks.

In contrast, *Architecture-based methods* target deep continual learning by structurally modifying the model to accommodate new tasks while preserving previous ones. A common solution in this category involves masking techniques. For example, [219] introduces binary masks over weights in a pre-trained model to deactivate specific parameters, enabling adaptation to new tasks without overwriting earlier knowledge. This process can be iteratively applied to several tasks. An alternative method by the same authors [220] involves pruning a trained model by setting some

connections to zero, briefly retraining it, and later reactivating the pruned connections for training on new tasks, while freezing the original weights. However, this strategy faces limitations in terms of scalability due to the finite number of available connections. Another variation [221] proposes the use of ternary neuron-level masks, where each neuron can be marked as trainable, frozen, or inactive. This allows reusability of neurons across tasks and supports network expansion to maintain performance across all previously seen tasks.

A distinct and more modular approach is proposed in [222]. This method begins by training a deep neural network for the first task. When introducing a new task, a separate network is instantiated, with connections established from the original network to each new layer. This leverages previously acquired knowledge while isolating learning paths to prevent interference. Though particularly effective for related tasks, this method may suffer when tasks conflict.

Article	Datasets used in experiments		
[89]	MNIST*;	CIFAR-10*	
[217]	MNIST*;	CIFAR-100	
[218]	MNIST*		
[219]	ImageNet;	CUBS;	Oxford102Flowers
[220]	ImageNet;	CUBS;	Oxford102Flowers
[221]	ImageNet;	CUBS;	Oxford102Flowers
[222]	Atari games [223]		

Table 5.5: Overview of the datasets utilized in the experiments described in Section 5.2.2.2. Asterisks indicate datasets that have undergone specific modifications.

There is notable variability in how these methods evaluate performance. While some works focus on accuracy, others emphasize error rates or the extent of forgetting. For example, [89] applies pixel permutation to MNIST, similar to [91], and modifies label assignments to simulate varying behaviors. They report improved outcomes over EWC and LwF [224], though they also highlight the artificial nature of this simulation. Interestingly, the methods introduced in [219–221] all conduct their experiments using the same trio of datasets: ImageNet [181], the CUBS dataset [225], and Oxford102Flowers [226] (refer to Table 5.5). This kind of dataset consistency across different approaches is, as discussed throughout this work, highly unusual.

5.3 Addressing Federated and Continual non-IID data

From the insights given in Section 4.3, we can interpret concept drift within Continual Learning (CL) as the temporal analogue of *non-IID data* encountered in Federated Learning (FL). In essence, time-dependent distributional shifts cause the statistical variability we observe in continual scenarios. Although any transformation over time of a single dataset should theoretically be regarded as identically distributed, since only one dataset is involved, the subsets gathered at different moments by a single client i, say D_i^t and D_i^{t+k}, can be regarded as two distinct datasets, with $D^t \subsetneq D^{t+k}$.

From this point of view, considering data as non-identically distributed over time is reasonable. Returning to the decomposition $P(x, y) = P(x) \cdot P(y|x)$, we encounter a situation directly analogous to the one described in Section 5.1: when the data available to a client does not shift over time (i.e., it remains IID), then both marginal and conditional distributions remain invariant, that is, $P_i^t(x) = P_i^{t+k}(x)$ and $P_i^t(y|x) = P_i^{t+k}(y|x)$. Otherwise, we can observe the three scenarios depicted in Figure 5.4. From this point onward, we use the term *temporal non-IID data* to refer to intra-client distributional drift over time (see Section 4.3.1), while *spatial non-IID data* denotes inter-client statistical differences in collaborative training setups (see Section 5.1).

In practice, datasets exhibit a wide range of variations. Within federated contexts, each client typically gathers its own dataset under specific, potentially unique conditions, which naturally leads to statistical divergence. These variations may arise from differences in the observed inputs, where $P_i(x) \neq P_j(x)$, or from distinct labeling patterns, such that $P_i(y|x) \neq P_j(y|x)$ (see Table 5.1). Relying solely on traditional FL solutions won't suffice if we want the model to accommodate client-specific characteristics. Additionally, learning and inference often occur over extended periods, implying that the model must remain adaptable. Over time, both the input features and corresponding labels may change, i.e., $P^t(x) \neq P^{t+k}(x)$ and $P^t(y|x) \neq P^{t+k}(y|x)$, as illustrated in Figure 5.4.

Overall, we can enumerate four types of scenarios each for spatial and temporal dimensions of data variation, resulting in 16 total combinations. These combinations, along with various solutions tailored to handle them, are presented in Table 5.6. Note that even the IID case is included to ensure full coverage of potential configurations.

Each of these cases may demand distinct methods for effective handling. For example, if the environment introduces temporal changes in input features (for one or multiple participants), memory-based techniques could be helpful to retain and generalize from past knowledge, thereby preventing catastrophic forgetting. However,

		Spatial heterogeneity			
		No changes (IID data)	Changes in the input space throughout clients $P_i(x) \neq P_j(x)$	Changes in the behaviour throughout clients $P_i(y\|x) \neq P_j(y\|x)$	Changes in the input space and behaviour throughout clients
Temporal heterogeneity	No changes (IID data)	OUT OF SCOPE	**[117—123, 129] [141, 146—166]**	**[124—127] [185—187]**	**[16, 128] [136—138]**
	Changes in the input space over time, $P^t(x) \neq P^{t+k}(x)$ (Virtual Concept Drift)	**[17, 91, 92] [198—203] [212—216]**	NOT		
	Changes in the behaviour over time, $P^t(y\|x) \neq P^{t+k}(y\|x)$ (Real Concept Drift)	**[89] [204—206] [217—222]**	ADDRESSED		
	Changes in the input space and behaviour over time (Total Concept Drift)	NO SPECIFIC ALGORITHMS	SO FAR		

Table 5.6: Spatial and Temporal heterogeneity learning scenarios, and the strategies that could potentially solve each situation. Strategies that deal with changes in both the input space and the behavior are placed only in the last row/column, and not in the previous ones.

although effective for such input-driven drift, these approaches generally fall short when tackling behavioral shifts or inter-client data divergence. Therefore, this section aims to determine which methodologies align best with each type of distributional change. Until now, we have briefly covered some FL personalization approaches and introduced drift detection methods, without delving into response mechanisms.

It's worth mentioning that if we manage to solve the scenarios corresponding to the first row and column of Table 5.6, then we can resolve any combined situation by applying the relevant row-wise and column-wise strategies, provided they're compatible. Facing both federated and continual dynamics in tandem is a novel scenario that has started been considered very recently [193–195], and has a growing interest and research advances nowadays [227–229]. This situation leaves a wide open area for future work that has yet to be fully explored.

In order to implement FL or CL across the various settings illustrated in Table 5.6, the data must adhere to certain prerequisites. This section outlines these constraints which are encapsulated in Table 5.7.

Analyzing all the situations shown in Table 5.6, some can be addressed through advanced methodologies without requiring any further assumptions, while others may necessitate meeting specific conditions not typically imposed by conventional FL nor CL. As discussed in Sections 5.1 and 5.2, managing fluctuations in the marginal input distributions $P(x)$, whether across space or over time, can be accomplished without needing extra data [146, 151]. In other words, unsupervised learning methods may also prove effective in such contexts. However, identifying shifts in the conditional distribution $P(y|x)$ generally requires access to labeled examples [186, 219], since these shifts can only be quantified by observing the model's prediction errors.

More concretely, we introduce three requirements that must be met to address certain use cases, labeled as Constraints 1P-MT, AP-1T, and AP-MT in Table 5.7:

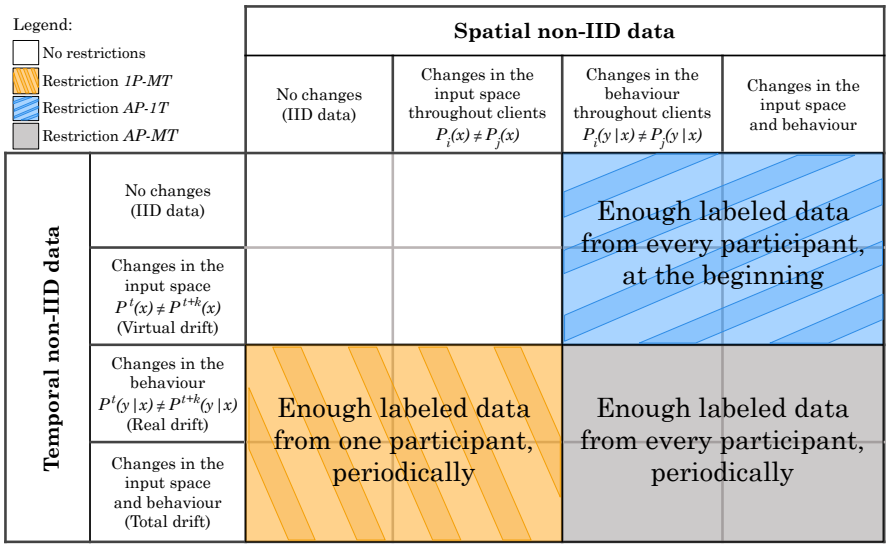


Table 5.7: Required restrictions for the non-IID learning scenarios.

i) If client behavior evolves over time but remains identical across devices, i.e., data corresponds to cells labeled as *Constraint 1P-MT* (one participant, multiple time steps) in Table 5.7, then it becomes necessary to acquire sufficient labeled samples from at least one user periodically. Since participant behavior is uniform across clients in this case, knowing the behavior of just one client suffices. In the event of a Real Concept Drift, these labeled samples allow the system to identify the drift and respond accordingly.

ii) In contrast, if the data fits the cells labeled as *Constraint AP-1T* (all participants, one time step) in Table 5.7, then each client has a distinct conditional distribution that remains unchanged over time. In this scenario, labeled data from every client is required at the beginning of the training phase in order to characterize their initial behavior. Since this behavior does not vary with time, further labeled data will no longer be necessary later on.

iii) Finally, when conditional distributions differ across both spatial and temporal dimensions, which corresponds to cells marked as *Constraint AP-MT* (all participants, multiple time steps) in Table 5.7, then labeled samples from every participant must be gathered periodically, allowing us to detect both skew data distributions across clients and also concept drifts over time, and act accordingly. This requirement assumes more information than the other two, meaning any context in Table 5.7 can also be resolved if this condition holds. Nevertheless, assuming such information is available in practical applications might be unrealistic.

The variety of heterogeneity that can be managed without any supplementary assumptions is, by far, the most thoroughly investigated in the existing literature (as illustrated by the number of references in Table 5.6). Scenarios requiring additional conditions have received comparatively less attention, not because such problems are rare, but because crafting solutions under these stricter requirements is inherently more challenging.

The spatial and temporal components can be treated independently, as they call for distinct sets of techniques, and they represent different sources of variation. In real-world scenarios, both axes might be present, and thus the model employed must include mechanisms capable of addressing each type of heterogeneity.

Approaches for dealing with spatial and temporal heterogeneity are orthogonal in general, meaning they do not interfere with one another. For example, using sliding windows to identify distribution shifts over time, and applying domain adaptation techniques per client to accommodate divergent feature spaces, are two strategies that can be used simultaneously without conflict. As a result, when confronted with scenarios exhibiting both spatial and temporal non-IID characteristics, each dimension of heterogeneity can be tackled separately using the necessary information, meaning the constraints previously described.

5.4 Results and Discussion

To robustly demonstrate that the constraints outlined in Section 5.3 are indeed necessary, and that the associated heterogeneity cannot be effectively managed without them, we present a series of experiments in this section. The goal is to showcase how different manifestations of non-IID data degrade model quality and hinder performance. We organize the experiments into two categories: those involving spatial non-IID distributions (Section 5.4.1) and those involving temporal non-IID distributions (Section 5.4.2).

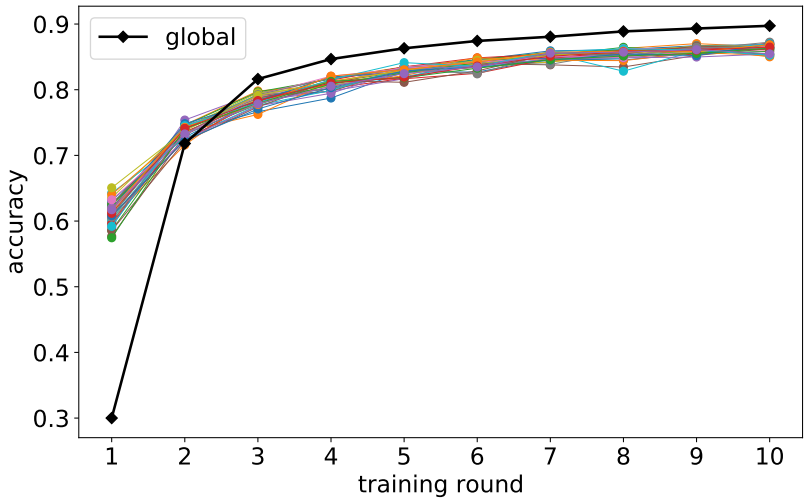
All the experiments utilize the *Digit-five* dataset, which aggregates MNIST, MNIST-M, SVHN, USPS, and Synthetic datasets [154]. This dataset is chosen due to the substantial visual variety among its inputs, allowing each constituent dataset to act as a distinct domain. This approach lets us simulate domain shift in $P(x)$ across clients and time without manually modifying the distributions. Each domain is limited to 60,000 samples, giving a combined total of 300,000 instances. These are evenly spread across 50 clients, with each client receiving 6,000 data points.

We conducted experiments in two distinct settings: one characterized by spatially non-IID data and the other by temporally non-IID data. Each setup required different considerations regarding the data organization and processing strategies to accurately represent the respective phenomena. These details are elaborated in Sections 5.4.1 and 5.4.2. Across all experiments, the model used is a basic Convolutional Neural Network (CNN) composed of four convolutional layers followed by three fully connected layers. Neural networks are the most common approach in FL strategies as they are very scalable and have very good properties regarding data privacy. Using neural networks, the only piece of information shared by the participants are the parameters of the network, which are not sufficient to retrieve the original data employed to train the network.

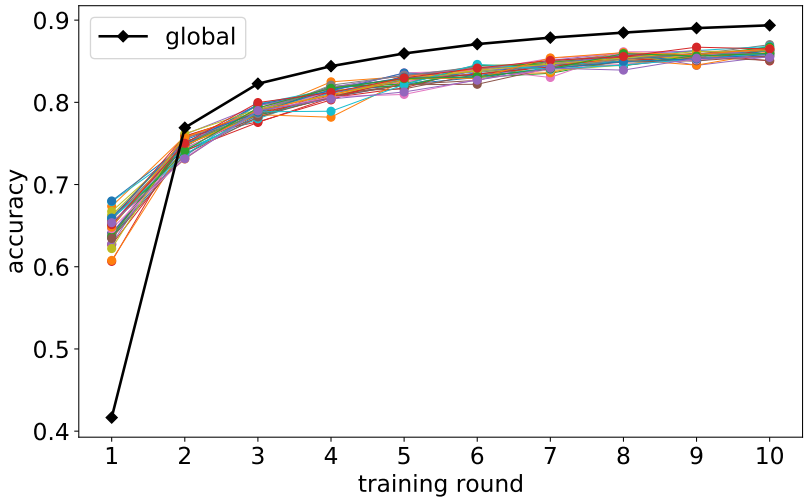
Additionally, each experiment was repeated multiple times to ensure statistical validity and to rule out any anomalous outcomes.

5.4.1 Spatial non-IID scenarios

This setting features inter-client data variation while maintaining temporal consistency. We introduce four plausible cases to illustrate how disparities in client data distributions impact the efficacy of FL algorithms.

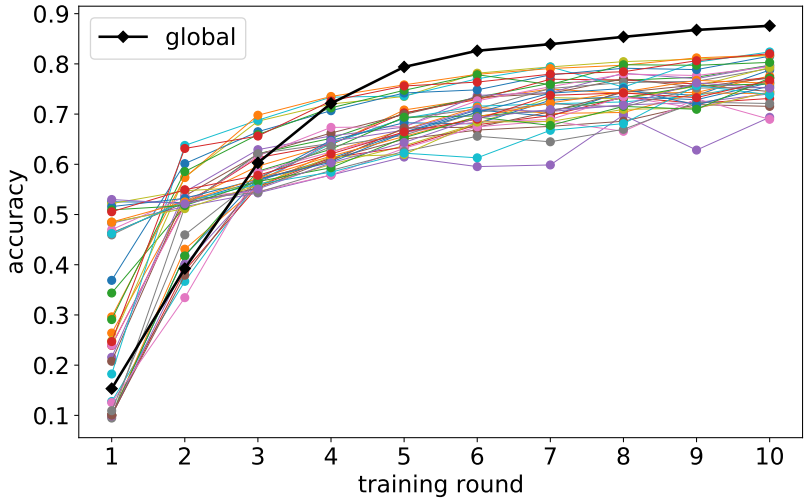


(a) Spatial baseline – FedAvg.

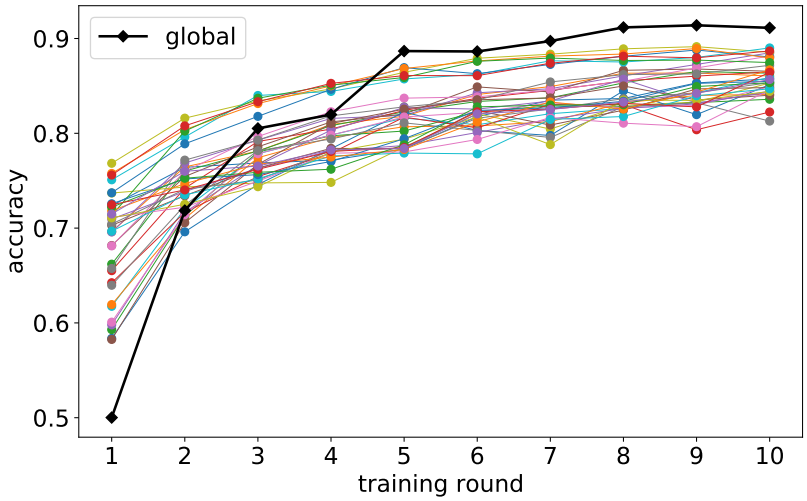


(b) Spatial baseline – FedProx.

Figure 5.7: Performance of FedAvg and FedProx across scenario A. The thick black line indicates global model accuracy, while the others reflect the accuracy achieved by individual clients.

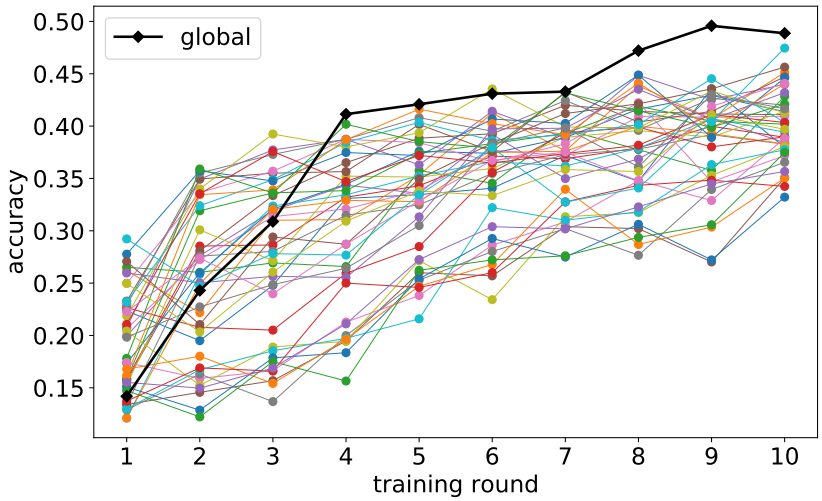


(a) Input heterogeneity with balanced test – FedAvg.

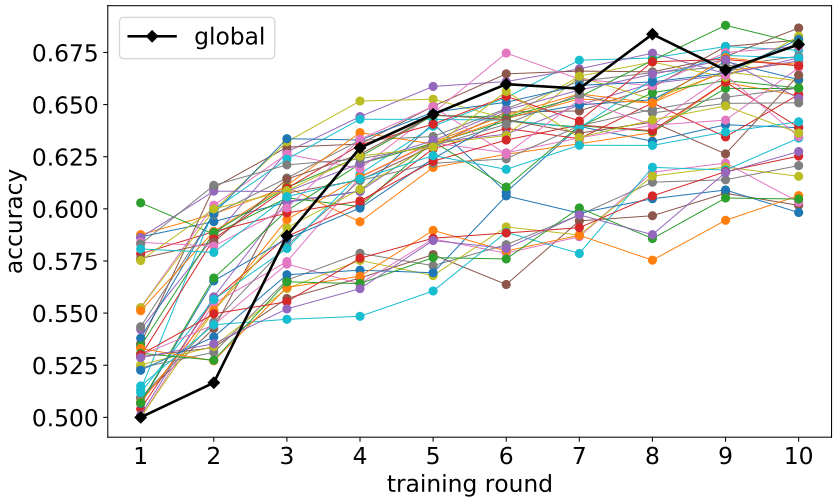


(b) Input heterogeneity with balanced test – FedProx.

Figure 5.8: Performance of FedAvg and FedProx across scenario B. The thick black line indicates global model accuracy, while the others reflect the accuracy achieved by individual clients.

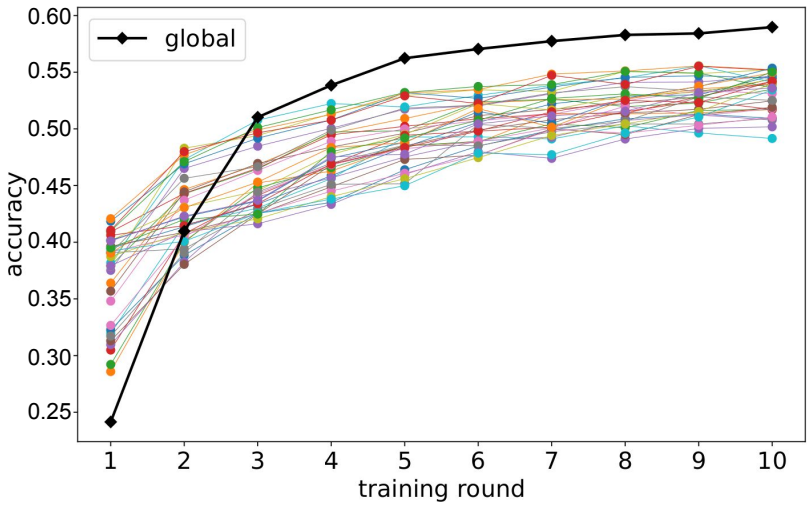


(a) Input heterogeneity with domain bias at test time – FedAvg.

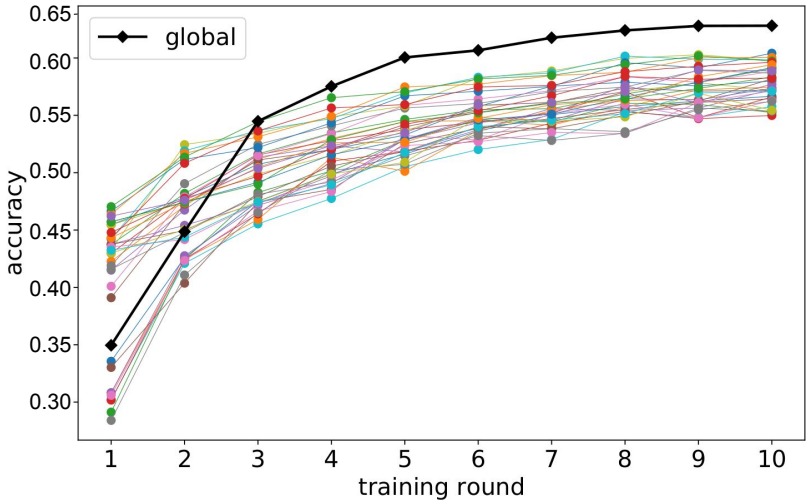


(b) Input heterogeneity with domain bias at test time – FedProx.

Figure 5.9: Performance of FedAvg and FedProx across scenario C. The thick black line indicates global model accuracy, while the others reflect the accuracy achieved by individual clients.



(a) Label inconsistency – FedAvg.



(b) Label inconsistency – FedProx.

Figure 5.10: Performance of FedAvg and FedProx across scenario D. The thick black line indicates global model accuracy, while the others reflect the accuracy achieved by individual clients.

The models evaluated include *FedAvg* and *FedProx*, the latter being specifically tailored to handle cross-client $P(x)$ variations [129]. We selected 35 clients for training, leaving the remaining 15 for evaluation purposes.

Scenario A. Spatial baseline. This reference scenario assumes each client holds 1,200 examples from each domain, producing an entirely IID setup where all of the participants should be able to identify all of the possible data domains. While unrealistic, it serves as a control case to compare with the non-IID ones. As seen in Figures 5.7a and 5.7b, both *FedAvg* and *FedProx* perform very well under this condition, achieving 90% accuracy.

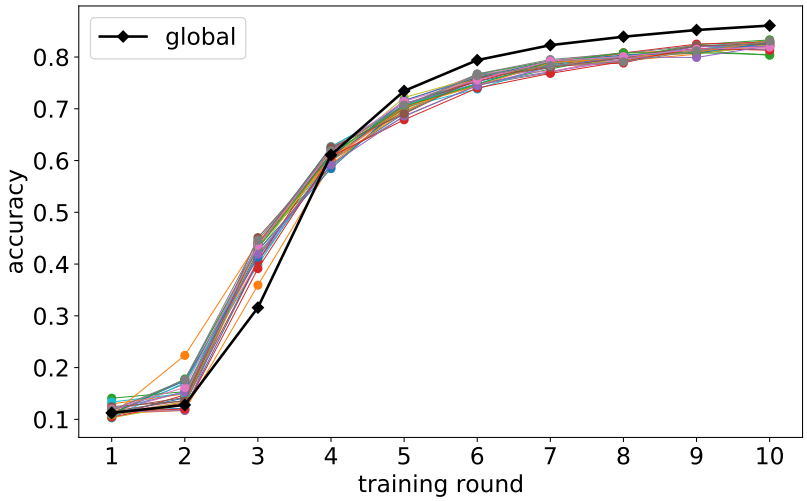
Scenario B. Input heterogeneity with balanced test. Here, each client receives data exclusively from a single domain, and test clients are randomly chosen. Figures 5.8a and 5.8b show that both algorithms yield performance close to the baseline, although they require a few more training rounds to converge. This result indicates their robustness to moderate input heterogeneity.

Scenario C. Input heterogeneity with domain bias at test time. This scenario mimics domain adaptation by selecting only MNIST clients for testing, none of which contributed to training. As shown in Figures 5.9a and 5.9b, model accuracy drops significantly due to the unseen domain. *FedProx*, however, outperforms *FedAvg* (68% vs. 48%), demonstrating its enhanced capability to generalize under such shift.

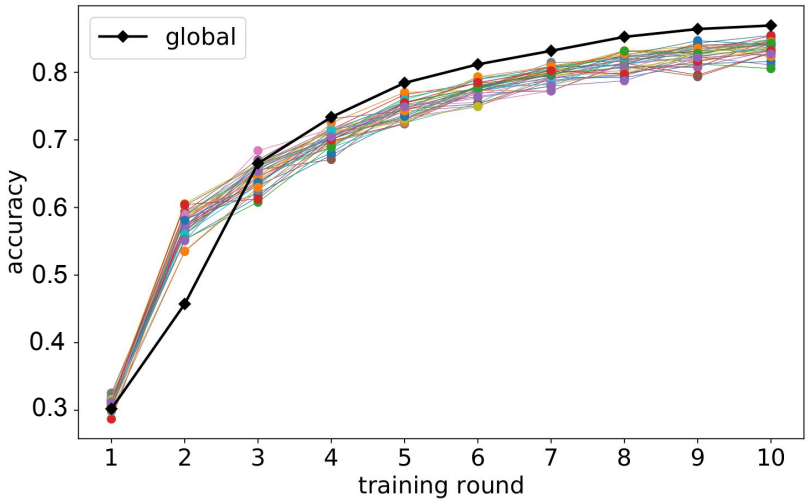
Scenario D. Label inconsistency. In this case, client datasets remain domain-specific as in Scenario B, but labels in the SVHN dataset are systematically altered. Thus, 10 clients exhibit a different labeling scheme, with 3 of them selected for testing. Figures 5.10a and 5.10b show similar results for both methods (63% and 64% respectively), reflecting that neither is equipped to address label-based heterogeneity.

5.4.2 Temporal non-IID scenarios

In this experimental setup, data evolves over time, but its statistical properties and distribution remain consistent across all clients, and changes occur simultaneously for all of them. Although this form of temporal shift is more regular and manageable than what typically arises in real-world applications, it is still sufficient to undermine the performance of federated models, as we demonstrate below. As in prior sections, we compare two algorithms: *FedAvg* and *CDA-FedAvg*. It is worth recalling that *CDA-FedAvg* [2] is specifically designed to cope with temporal variations in $P(x)$, while *FedAvg* does not support CL and assumes full data availability from the outset, an assumption violated in this scenario. A complete description of CDA-FedAvg, as well as deep analysis of its design and capabilities, can be found in Chapter 6.

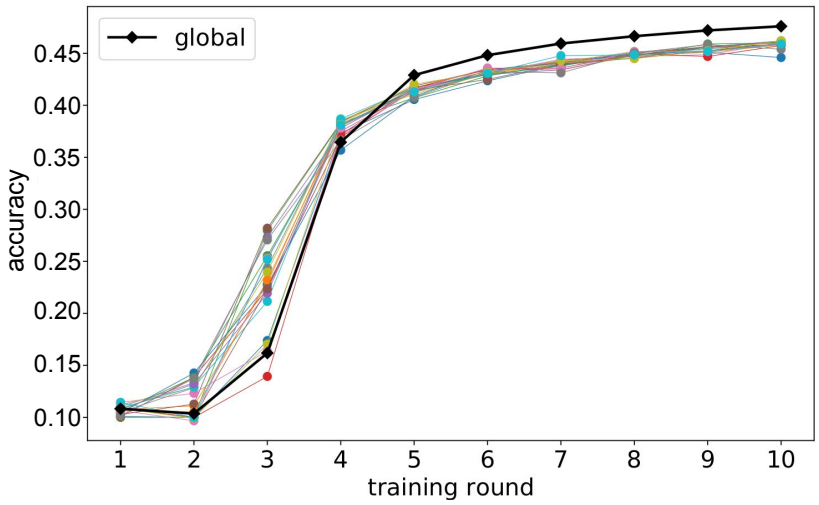


(a) Temporal baseline – FedAvg.

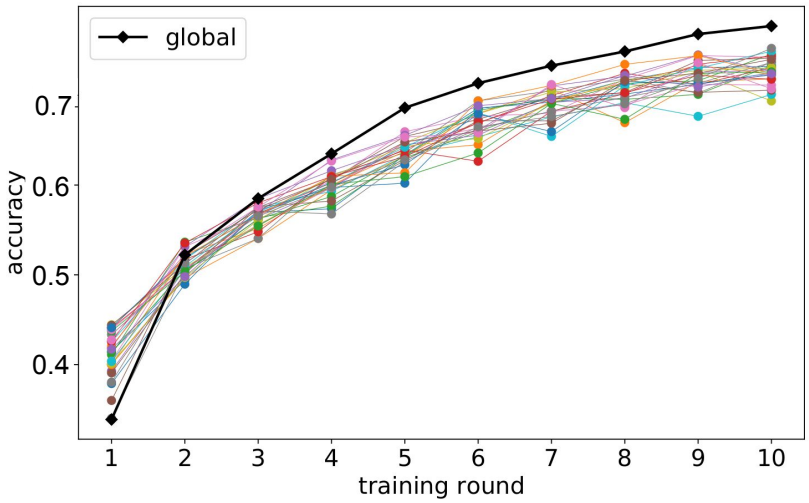


(b) Temporal baseline – CDA-FedAvg.

Figure 5.11: Performances of FedAvg and CDA-FedAvg in the temporal non-IID Scenario A. The black thick line represents the global model accuracy, whereas the other ones represent the accuracy of the model of each client.



(a) Virtual drift – FedAvg.



(b) Virtual drift – CDA-FedAvg.

Figure 5.12: Performances of FedAvg and CDA-FedAvg in the temporal non-IID Scenario B. The black thick line represents the global model accuracy, whereas the other ones represent the accuracy of the model of each client.

To apply *FedAvg* in this dynamic context, we adopted a straightforward adaptation: each client chosen for training uses 800 data samples from its local dataset in each round. These instances are then stored so that, during the next participation, the client trains on those previous 800 examples along with an additional batch of 800 new samples. To simulate the memory constraints of real-world devices, we imposed a cap of 5000 data samples per client. Once this limit is exceeded, the oldest data points are discarded to make room for new ones. This emulates the practical limitations of edge devices, which often cannot retain all observed data. The same limits and policies are applied to *CDA-FedAvg*, although this method retains samples only when a concept drift is detected, thereby managing memory usage more judiciously.

Scenario A. Temporal baseline. The first case examined is a reference scenario. Each client holds 1,200 examples from each of the 5 domains, randomly mixed so that every training batch of 800 samples reflects the overall domain distribution. This yields a full IID situation, as each client sees a representative blend of all domains in every round. As shown in Figures 5.11a and 5.11b, both *FedAvg* and *CDA-FedAvg* reach an accuracy of 84% under this setting. Nonetheless, this is slightly inferior to the results in the Spatial baseline (Figures 5.7a and 5.7b) due to the sequential processing of data and the imposed storage limit of 5000 samples.

Scenario B. Virtual drift. In this scenario, all clients experience identical virtual drifts over time. Each one receives 1200 samples per domain, but unlike the baseline, these are arranged in domain-specific blocks rather than being shuffled. For instance, the first 1200 instances come from MNIST, followed by 1200 from SVHN, and so on. As seen in Figure 5.12a, *FedAvg* struggles significantly under this configuration, with accuracy dropping to 48%. In contrast, *CDA-FedAvg* successfully mitigates the adverse effects of this drift (Figure 5.12b), attaining a notably higher accuracy of 79%.

It is important to remark that although we are aiming for the more general setting possible, each problem in FL has its own particularities and data should be analyzed carefully before deciding which algorithm fits best to get an accurate model. The general results achieved in these experiments cannot be extrapolated to every other task that we solve in the context of FL or CL.

CHAPTER 6

Concept Drift Aware Federated Averaging

As mentioned in Section 4.2, in recent decades the proliferation of smart technologies, such as smartphones, wearables, IoT devices, and service robots, has transformed daily life, enabling diverse applications across sectors like healthcare, education, and finance. These devices continuously generate large volumes of data, which can be leveraged to train adaptive machine learning models. A promising strategy for this is collaborative learning, wherein a global model is constructed from partial data contributions of distributed participants. This can be done via centralized (server-based) or decentralized approaches such as FL [17, 230, 231].

Centralized approaches face scalability and privacy issues, as we already explained in previous chapters. In addition, adaptability is limited for this kind of ML strategies. For this reason, FL has received more and more attention recently. It enables privacy preservation and is scalable. However, the adaptability of FL remains underexplored, especially regarding temporal dynamics. Continual Learning (CL) [79, 232] addresses this by adapting models to evolving data while retaining useful past knowledge, balancing stability and plasticity [233].

This chapter details a new method, Concept-Drift-Aware Federated Averaging (CDA-FedAvg), an extension of the FedAvg algorithm [17], tailored for continual single-task scenarios with non-IID and temporally shifting data. We validate CDA-FedAvg through experiments on human activity recognition (HAR) using smartphone data from ten users. Results show that CDA-FedAvg outperforms standard FedAvg in terms of accuracy, adaptability, and efficiency in storage and communication.

Research in FL has largely focused on non-IID data [76,129,234], demonstrating that FedAvg can converge for some cases of heterogeneous client distributions, though more slowly and performing worse than in optimal conditions. Some approaches explore local model personalization or track client-specific adaptations [18]. However, few address temporal non-stationarity. FedWeIt [195] proposes weight decomposition for inter-client knowledge transfer under continual settings, improving accuracy and communication efficiency, though it does not explicitly detect drift.

In the present chapter, FedAvg is redesigned to address concept drift through explicit detection mechanisms, allowing the model to determine what to learn and when [235]. We focus on the realistic case of a single task with temporally evolving data, contributing a method better suited to adaptive, privacy-preserving, and distributed learning systems.

The most significant contributions of this chapter have been published in the following article, and have been reproduced here under Creative Commons Attribution (CC BY) license: [2] F . E. Casado, D. Lema, M. F. Criado, R. Iglesias, C. V. Regueiro, S. Barro, Concept drift detection and adaptation for federated and continual learning, Multimedia Tools and Applications (2022) 1–23.

The Journal Information Fusion had the following Journal Citation Reports (JCR) indicators (2021): Quartile Q2 in the categories *Computer science, theory and methods* (42/109) and *Computer science, software engineering* (48/110), and an overall Journal Impact Factor (JIF) of 2.577.

The chapter is organized as follows: Section 6.1 establish the necessary background for understanding the Concept Drift mathematically and the difficulties it presents. After that, Section 6.2 presents in detail the Concept Drift Aware FedAvg (CDA-FedAvg) algorithm. Finally, Section 6.3 presents a use case of this algorithm in a real case scenario regarding Human Activity Recognition.

6.1 Concept drift at local and global levels

In typical FL scenarios, the training process consists of multiple iterations involving localized model training and central parameter aggregation. This was already explained in Section 4.2.1, but we briefly mention the most important key points here: In each round r, every client $j \in \mathcal{N} = \{1, \ldots, N\}$ and the server s carry out two separate operations: optimization of the local parameters of each client, $\mathbf{w}_j^r$, and optimization of the global parameters, $\mathbf{w}_G^r$. Initially, the server sets the model's parameters (weights and biases). At the start of each iteration, a random subset of clients,

$\mathcal{N}^r \subseteq \mathcal{N}$ of cardinality $J = |\mathcal{N}^r|$ is sampled. The server distributes the current global model to those selected clients.

During the local update phase, each client $j \in \mathcal{N}^r$ applies Stochastic Gradient Descent (SGD) on its own dataset and returns the modified parameters to the server. The server then consolidates the updates from all participating clients in $\mathcal{N}^r$ into a unified model. Typically, this aggregation is done using a mean function, although specific methods may vary. In the case of FedAvg, this involves a weighted mean:

$$\mathbf{w}_G^r = \sum_{j=1}^{J} \frac{M_j}{M} \mathbf{w}_j^r,$$

where M is the total number of data points in the round, and M_j denotes the number of samples belonging to client j. The revised model is then sent back to the clients, a new group is chosen, and the process repeats. Algorithms 1 and 2 from Section 4.2.1 present the pseudocode for FedAvg.

However, as we demonstrate in Section 6.3, applying FedAvg directly in practical contexts often results in catastrophic forgetting, since client data is typically non-stationary and non-IID. Such evolving data distributions indicate the presence of concept drift. Without any external cues, the model must autonomously detect and react to these changes to maintain its effectiveness.

Concept drift is a key concern in CL. It was defined formally in Definitions 4.3.1 and 4.3.2 for the general case, but in the present work it is important to differentiate between a local concept drift in one participant, and a global concept drift among all of them.

It is important to note that a drift occurring on a single client does not always impact on the overall FL model. If only one device undergoes a distributional change, and it is not significant enough to alter the collective distribution, the global model may remain unaffected, leading to a performance decrease on the client that experienced the drift. Hence, in our FL setting we will refer to the following types of concept drift:

Definition 6.1.1 (Local Concept drift). Let $j \in \mathcal{N}$ be one of the participants of an FL process. Let $S_j^{[0,t]} = \{(x_i, y_i)\}_{i=1}^{M_j}$ be the data stream of samples collected by participant j, which has size M_j, collected during the period $[0, t]$. x_i represents the feature vector of a sample and y_i its corresponding label. Let $P_j^t(x, y)$ be the probability density function of $S_j^{[0,t]}$.

A local concept drift occurs at timestamp t for the participant j if the probability density function of that clients' data stream is different after that timestamp:

$$\exists\, t > 1 \,:\, P_j^{t-1}(x, y) \neq P_j^t(x, y).$$

Definition 6.1.2 (Global Concept drift). Let $\mathcal{N}$ be the set of participants of an FL process. Consider the data stream $S_G^{[0,t]} = \{(x_i, y_i)\}_{i=1}^{M}$, conformed by all data samples collected by any client $j \in \mathcal{N}$ during the period $[0, t]$. x_i represents the feature vector of a sample and y_i its corresponding label. Let $P_G^t(x, y)$ be the probability density function of $S_G^{[0,t]}$.

A global concept drift occurs at timestamp t if the global probability density function of all clients is different after that timestamp:

$$\exists\, t > 1 \,:\, P_G^{t-1}(x, y) \neq P_G^t(x, y).$$

Note that the data stream $S_G^{[0,t]} = \{(x_i, y_i)\}_{i=1}^{M}$ is unknown for every single client and also, for the central orchestrator of the FL process, since data in FL framework cannot be shared. For this reason, detecting a global concept drift can be a problematic task.

Having a single participant experiencing a local concept drift may not have an impact on the joint data distribution of all participants. Conversely, detecting a global drift implies that at least one local drift has been identified. While we define local and global drift in terms of distribution shifts, it is critical to clarify that we do not observe the true distributions directly. Instead, we estimate these changes using empirical distributions derived from observed samples.

A client's local drift may go unnoticed by the global model for two reasons: either the change is exclusive to that client, or other clients haven't yet experienced or detected the shift. In such situations, the global model may underperform on that specific client, making an adaptive mechanism desirable. Nonetheless, in our experimental setup, we assume synchronous drifts across all clients, rendering local and global drifts effectively equivalent.

A global concept drift typically causes a drop in model accuracy. Therefore, as we show in Section 6.2, it is necessary to enhance the FedAvg algorithm to include drift detection and adaptation capabilities, allowing it to autonomously respond to dynamic data conditions.

6.2 CDA-FedAvg algorithm

Algorithms 3 and 4 describe our proposed approach. Algorithm 3 presents the pseudocode for the server-side operations, whereas Algorithm 4 illustrates the client-side procedure. In contrast to FedAvg, our method is asynchronous, meaning there is no fixed order in which events or communications between the server and the clients occur. Owing to the mechanisms for detecting and adapting to concept drift, each participant has sufficient independence to determine when to initiate training and which data to utilize for that task, with the server merely coordinating the overall workflow. As shown in Algorithm 3, the server begins by initializing the global model and disseminating it to all clients (lines 1–2). Afterwards, it regularly checks for any local updates from one or more participants (lines 4–5), which prompts a global aggregation step to refresh the global model (line 6). Each time the model is globally updated, the server must broadcast the new version to all participants to ensure consistency (line 7).

Algorithm 3: Concept Drift Aware Federated Averaging - Server side

Input : Set of clients $\mathcal{N} = \{1, \dots, N\}$ of size $J = |\mathcal{N}^r|$.

Output: Global model, $\mathbf{w}_G$.

1 Initialization: Random parameter initialization $\mathbf{w}_G^0$
2 Send_model_to_clients($\mathbf{w}_G^0, \mathcal{N}$);
3 **while** *true* * **do**
4 Listen for client updates for all $j \in \mathcal{N}$
5 **if** $\exists\ j \in \mathcal{N}$: *new update* $\mathbf{w}_j^t$ *is received* **then**
6 $\mathbf{w}_G^t \leftarrow \sum_{j=1}^{J} \frac{M_j}{M} \mathbf{w}_j^t$;
7 Send_model_to_clients($\mathbf{w}_G^t, \mathcal{N}$)

* Given the continual nature, the stop criteria are not determined and should be decided ad-hoc for each particular implementation.

It is important to note that this framework allows any number of clients to send updates at arbitrary times, permitting varying participation frequencies across them. As a result, the global model may become more tailored to the behavior of the most active clients. To avoid overfitting the model to specific participants, it may be beneficial to impose constraints on participation levels, thereby regulating the number of updates each client can submit.

Clients are in charge of performing local learning, including handling concept drift if it arises. Research concerning learning under concept drift typically distin-

guishes three core stages [102]: **drift detection** (identifying whether a drift has occurred), **drift interpretation** (determining when and how it happens), and **drift adaptation** (responding to the detected change). Our approach concentrates on detecting and adapting to global drift. To achieve this, each client (see Algorithm 4) continuously collects new data from its surroundings. This incoming data is analyzed to detect emerging patterns (drift detection) and to update the model accordingly (drift adaptation). The issue of understanding the drift in depth is outside the scope of this paper. Our concern lies only in detecting changes between two points in time, without delving into the specific nature or origin of the drift.

Algorithm 4: Concept Drift Aware Federated Averaging - Local training

Input : Minimum amount of data to train L, local batch size B, number of rounds R, number of epochs E, learning rate η, sensitivity to change λ, padding Δ, and maximum size for the sliding window N_{max}.

Output: None.

$\mathcal{Q} \leftarrow \{\varnothing\}$;
$\mathcal{L} \leftarrow \{\varnothing\}$;
$\mathcal{L} \leftarrow \text{DriftAdaptation}(\mathcal{L}, L, B, R, E, \eta)$;
while *true* * **do**
  if *new data sample* x_i *is collected* **then**
    $[\hat{y}_i, q_i] \leftarrow \text{predict}(\mathbf{w}_G^t, x_i)$;
    $\mathcal{Q} \leftarrow \mathcal{Q} \cup q_i$;
    if $|\mathcal{Q}| \geqslant N_{max}$ **then**
      $\mathcal{Q} \leftarrow \mathcal{Q} \backslash \{q_1\}$;
    $r \leftarrow \text{random}(0, 1)$;
    if $e^{-2q_i} \geqslant r$ **then**
      $d \leftarrow \text{DriftDetection}(\mathcal{Q}, \lambda, \Delta, N_{max})$;
      if d *is true* **then**
        $\mathcal{Q} \leftarrow \{\varnothing\}$;
        $\mathcal{L} \leftarrow \text{DriftAdaptation}(\mathcal{L}, L, B, R, E, \eta)$;

* Given the continual nature, the stop criteria are not determined and should be decided ad-hoc for each particular implementation.

Regarding the mechanisms for detection and adaptation, we propose that each client manages two distinct memory buffers: a short-term memory and a long-term memory. The short-term memory, denoted as $\mathcal{Q}$, temporarily holds the most recently acquired data samples. This recent data is retained for a limited duration and used to evaluate whether a drift has taken place. The long-term memory, denoted as $\mathcal{L}$, maintains representative samples from all previously observed concepts. This persistent memory is retained over extended periods and supports local training and retraining of the model to ensure all concepts are adequately learned. The client's procedure is outlined in Algorithm 4: Initially, both the short-term and long-term memories are empty (lines 1–2), and the model has not yet undergone local training. Consequently, the first batch of data collected by a client is assumed to belong to the original concept. These data are saved in the long-term memory and used to perform the initial local update (line 3). Subsequently, the client continues to gather new data, stores it in the short-term memory, and analyses it to detect possible drift (lines 5–12). Only when the drift detection method confirms the presence of drift (line 13) is the new concept's data transferred to the long-term memory, prompting additional rounds of local training (line 15). In the next two subsections, we elaborate on the two essential components of the client-side process: drift detection (Section 6.2.1) and drift adaptation (Section 6.2.2).

6.2.1 Drift detection

Drift detection refers to the collection of methods and strategies designed to identify and assess concept drift by pinpointing moments or intervals of change in the data's underlying distribution. In our scenario, detecting concept drift signals that the federated model no longer serves as an effective predictor for all clients and thus needs to be updated. We opt for a data distribution-based technique to perform the drift detection. This choice is motivated by the ability to detect virtual concept drift without requiring labeled data. In the following paragraphs, Algorithm 4 is detailed:

We adopt a detection mechanism inspired by the CUSUM algorithm, tailored to beta distributions [236], following the principles introduced by Haque et al. [237]. This procedure runs on each device whenever a new data point (x_i, y_i) is received (line 5). Since this method does not require labels to identify drift, we only need the new feature vector x_i. For each incoming sample, a metric is calculated to assess the divergence between the new and past data distributions. Any suitable metric proposed in previous studies may be used. We selected the model's confidence as our metric, given that our experiments are focused on a classification problem (Section 5). We

define confidence for a data sample (x_i, y_i) as the maximum conditional posterior probability assigned by the classifier. Specifically, we use the federated model to classify x_i, and we get the highest probability $P(c_k|x_i)$, where $c_k \in C = \{c_1, \ldots, c_C\}$ is one of the C possible classes (line 6). The resulting confidence q_i is then recorded in the client's short-term memory (line 7), which functions as a sliding window with maximum capacity equal to N_{max}. Our goal is to detect changes in the distribution of the confidence values stored in the short-term memory $\mathcal{Q}$.

In contrast to the original method described in [237], which uses a dynamic window that resets after each drift detection without bounding its size, we employ a fixed-size sliding window with an upper limit. This prevents unbounded growth in the absence of drift. When the size N_{max} is reached, adding a new entry to $\mathcal{Q}$ involves removing the oldest one (lines 8–9). This setup allows us to identify various types of drift: abrupt, recurring, gradual, and even incremental, provided the window is large enough to encapsulate sufficiently distinct data.In our case, since we are focusing on abrupt concept drifts (Section 5), we set $N_{max} = 1000$.

The core of our detection method is encapsulated in Algorithm 5, which is invoked at line 12 of Algorithm 4. However, since executing Algorithm 5 after every new confidence value could become computationally expensive, we limit its invocation. Specifically, it is triggered probabilistically, with a likelihood of e^{-2q_i}, for each confidence value q_i (line 11). This means that higher confidence values reduce the probability of running the drift detector, and vice versa.

Now, for the detection process, detailed in Algorithm 5, the window $\mathcal{Q}$, of size N, is divided into two sub-windows at every candidate point $k \in (\Delta, N - \Delta)$. Let $\mathcal{Q}_a$ and $\mathcal{Q}_b$ be the two segments, with $\mathcal{Q}_a$ containing the most recent confidence values. Each sub-window must contain at least Δ samples to maintain valid statistical properties. A decrease in model confidence typically accompanies concept drift, so we focus exclusively on identifying downward changes. If m_a and m_b are the mean confidence values in $\mathcal{Q}_a$ and $\mathcal{Q}_b$ respectively, a change point is considered if $m_a \leqslant (1 - \lambda) \times m_b$, with λ being the sensitivity threshold (line 7). In our experiments we employ the values $\lambda = 0.05$ and $\Delta = 100$, that are widely accepted in the literature.

The confidence values in $\mathcal{Q}_a$ and $\mathcal{Q}_b$ are assumed to follow distinct beta distributions, though the exact parameters are unknown. Algorithm 5 estimates these parameters (lines 9-10) using the method of moments [238], based on the sample means and variances. Next, the sum of the log-likelihood ratios s_k is computed within the inner loop (lines 11–12), where $f(q_i|\hat{\alpha}, \hat{\beta})$ is the beta distribution's probability density function with the estimated parameters $\hat{\alpha}$ and $\hat{\beta}$, evaluated at each confidence $q_i \in \mathcal{Q}$.

Algorithm 5: Drift detection method

Input : Sliding window $\mathcal{Q}$, sensitivity to change λ, padding Δ, and maximum size for the sliding window N_{max}.

Output: Boolean indicating whether a drift is detected.

1 $s_f \leftarrow 0$;
2 $T_h \leftarrow -\log(\lambda)$;
3 $N \leftarrow |\mathcal{Q}|$;
4 **for** $k \leftarrow \Delta$ *to* $N - \Delta$ **do**
5 $m_b \leftarrow \text{mean}(q_1, \dots, q_k \in \mathcal{Q})$;
6 $m_a \leftarrow \text{mean}(q_{k1}, \dots, q_N \in \mathcal{Q})$;
7 **if** $m_a \leqslant (1 - \lambda) \times m_b$ **then**
8 $s_k \leftarrow 0$;
9 $[\hat{\alpha}_b, \hat{\beta}_b] \leftarrow \text{estimate}(q_1, \dots, q_k)$;
10 $[\hat{\alpha}_a, \hat{\beta}_a] \leftarrow \text{estimate}(q_{k+1}, \dots, q_N)$;
11 **for** $i \leftarrow k + 1$ *to* N **do**
12 $s_k \leftarrow s_k + \log\left(\frac{f(q_i|\hat{\alpha}_a, \hat{\beta}_a)}{f(q_i|\hat{\alpha}_b, \hat{\beta}_b)}\right)$;
13 $s_f \leftarrow \max(s_f, s_k)$;
14 **if** $s_f > T_h$ **then**
15 true
16 **else**
17 false

This probability indicates how likely a value q_i is under the assumed beta distribution, and is defined as:

$$f(q_i|\alpha, \beta) = \begin{cases} \dfrac{q_i^{\alpha_i - 1}(1 - q_i)^{\beta_i - 1}}{B(\alpha, \beta)} & \text{if } 0 < q_i < 1, \\ 0 & \text{otherwise,} \end{cases}$$

where

$$B(\alpha, \beta) = \int_0^1 q_i^{\alpha - 1} \cdot (1 - q_i)^{\beta - 1} dq_i.$$

The variable s_k serves as a dissimilarity score at iteration k in the outer loop (lines 4–13). The more distinct the probabilities corresponding to $\mathcal{Q}_a$ and $\mathcal{Q}_b$, the

higher the value of s_k (line 12). Let k_{max} be the index at which the score s_k reaches its maximum within the range. A change is signaled at point k_{max} if the highest score $s_{k_{max}}$ exceeds a predefined threshold T_h (lines 14-17). As in the original method, we set $T_h = -\log(\lambda)$.

6.2.2 Drift adaptation

Once a concept drift has been identified, the model must continue its training process with awareness of the detected change. Failing to do so can result in catastrophic forgetting. In the context of neural networks, as is the case in our work, the most prevalent strategies to mitigate this issue are (1) regularization-based techniques, (2) rehearsal strategies, and (3) generative replay mechanisms [77]. Within CL, regularization involves safeguarding the weights that are critical to previous concepts from being altered. Rehearsal, or replay methods, rely on storing actual data samples as a memory of earlier tasks. Generative replay, on the other hand, involves training generative models to replicate the original data distribution, enabling the system to synthesize past samples while learning from new ones. Among these, rehearsal approaches have demonstrated superior performance, as they effectively preserve memory over extended periods [239].

In centralized cloud environments, a key limitation of these methods lies in their requirement to maintain a separate storage of raw data. This straightforward approach to retaining knowledge does not align well with data privacy concerns and can incur substantial storage costs. Nevertheless, in a federated environment, this issue is mitigated since each client can locally retain a small portion of its past data. Consequently, we propose using a basic rehearsal method, detailed in Algorithm 6. This algorithm is triggered whenever a new drift is detected (using the detection strategy described in Section 6.2.1). Its role is to handle the long-term memory at the client side and leverage this memory to locally retrain the model.

Formally, the long-term memory of a client j at time t is denoted as a dataset $\mathcal{L}_j = \{\mathcal{L}_0^j \cup \mathcal{L}_u^j \cup \ldots \cup \mathcal{L}_v^j\}$, which aggregates representative samples from each previously detected concept $\mathcal{K} = \{\kappa_0^j, \kappa_u^j, \ldots, \kappa_v^j\}$, where $0, u, v, \ldots$ correspond to the time indices when each concept shift was identified.

At the outset of Algorithm 6, the long-term memory does not yet include any examples corresponding to the newly observed concept κ_{new}^j. Therefore, the first action is to gather a sufficient number of samples $\mathcal{L}_{new}^j \subset \mathcal{L}^j$ of the concept κ_{new}^j (line 1). Any data collected after the most recent change point k_{max}, as detected by Algorithm 5, is immediately eligible for inclusion in $\mathcal{L}_{new}^j$.

In continual classification scenarios, the instances from different classes can arrive in any sequence. To ensure that the rehearsal dataset remains at least approximately balanced across classes, we define a heuristic for selecting data stored in the long-term memory. We impose a minimum data requirement, L, such that for each class $c \in C$, there must be at least $\frac{L}{2C}$ examples in the long-term memory associated with a given concept, where C denotes the total number of classes. Formally, client j continues gathering data until this is satisfied:

$$\text{For all } j \in \mathcal{N}, c \in C, \kappa_u \in \mathcal{K} : |\{(x_i, y_i) \in \mathcal{L}_u^j : y_i = c\}| \geqslant \frac{L}{2C}.$$

Algorithm 6: Drift adaptation method

Input : Sliding window $\mathcal{Q}$, sensitivity to change λ, padding Δ, and maximum size for the sliding window N_{max}.

Output: Boolean indicating whether a drift is detected.

$s_f \leftarrow 0$;
$T_h \leftarrow -\log(\lambda)$;
$N \leftarrow |\mathcal{Q}|$;
for $k \leftarrow \Delta$ *to* $N - \Delta$ **do**
  $m_b \leftarrow \text{mean}(q_1, \ldots, q_k \in \mathcal{Q})$;
  $m_a \leftarrow \text{mean}(q_{k1}, \ldots, q_N \in \mathcal{Q})$;
  if $m_a \leqslant (1 - \lambda) \times m_b$ **then**
    $s_k \leftarrow 0$;
    $[\hat{\alpha}_b, \hat{\beta}_b] \leftarrow \text{estimate}(q_1, \ldots, q_k)$;
    $[\hat{\alpha}_a, \hat{\beta}_a] \leftarrow \text{estimate}(q_{k+1}, \ldots, q_N)$;
    for $i \leftarrow k + 1$ *to* N **do**
      $s_k \leftarrow s_k + \log\left(\frac{f(q_i|\hat{\alpha}_a, \hat{\beta}_a)}{f(q_i|\hat{\alpha}_b, \hat{\beta}_b)}\right)$;
    $s_f \leftarrow \max(s_f, s_k)$;
if $s_f > T_h$ **then**
  true
else
  false

In our evaluation, we use $L = 1400$, which represents a sizeable dataset, but ensures that the memory remains robust over time. We also assume that clients have the capacity to retain data for an unlimited number of detected concepts. Once a sufficient quantity of data related to the new concept is collected, the client can proceed with local model updates (lines 3–11). This training is conducted over a limited number of iterations R. In our tests, we use $R = 5$, although this value may vary depending on the specific application. After each training round, the updated local model $\mathbf{w}_j^{t+1}$ is transmitted to the server, where it participates in global aggregation (line 15).

6.2.3 Advantages of using CDA-FedAvg algorithm

CDA-FedAvg not only supports continual learning while preventing catastrophic forgetting, but it also addresses two essential yet often neglected questions in federated learning: what information should be learned, and at which point in time. As previously noted, a straightforward implementation of FedAvg lacks any built-in policy for selecting which data to retain or discard. Typically, unless constrained by task-specific requirements or implementation limitations, devices will preserve all gathered data. Moreover, FedAvg imposes no ceiling on the number of training iterations, which can, in theory, go on indefinitely. Although each round only involves a random subset of devices, meaning not all participate continuously, it is still possible to apply restrictions, such as allowing training only when the device is charging or connected to Wi-Fi, especially relevant for mobile devices [14, 240]. Still, these constraints do not consider whether further training is genuinely necessary.

Unrestricted training without intelligent criteria leads to excessive bandwidth usage. Indeed, many studies in the federated learning domain highlight communication overhead as the primary limitation [17, 195]. Our method addresses this by incorporating a concept drift detector and long-term memory, enabling the identification of relevant versus outdated data. This allows for computational efficiency and reduces the complexity of the algorithm, in particular, it saves a lot of time-consuming actions regarding the communication costs between the sever and the participants.

6.3 Results and Discussion

To evaluate CDA-FedAvg, we selected a challenging real-world scenario: Human Activity Recognition (HAR) using smartphone data. Specifically, we employed the dataset introduced by Shoaib et al. [241], which comprises sensor readings for seven

activities: sitting, standing, walking, jogging, climbing stairs, descending stairs, and cycling. A distinctive feature of this dataset is that each activity was recorded with the phone placed in five different body positions (See Figure 6.1): (1) attached to a belt, (2) in a pocket on the left, (3) in a pocket on the right, (4) strapped to the upper right arm, and (5) worn on the right wrist. The fifth location is especially important in our study, as it later enables analysis of CDA-FedAvg's adaptability to environmental or contextual changes.

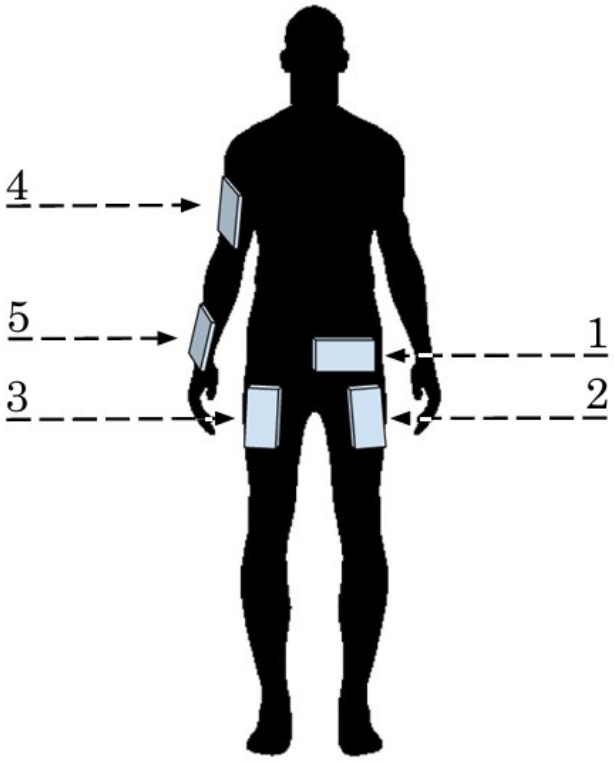


Figure 6.1: General view of the different positions of the mobile phone in the data collection stage.

Data was collected from 10 male volunteers aged between 25 and 30. Each participant performed all seven activities for roughly 20 minutes (4 minutes per each placement of the mobile phone). All trials took place indoors within the same building, except for biking, which occurred outdoors. For consistency, phones were held

vertically in the pockets, wrist, and arm positions, and horizontally on the belt. The dataset includes readings from the smartphone's inertial sensors: accelerometer, gyroscope, and magnetometer, sampled at 50 Hz, a rate sufficient for distinguishing physical movements.

The classification task involves identifying the user's current activity among the seven, framed as a multi-class prediction problem. Sensor placement data is later used to simulate concept drifts. For the federated learning setup, we designated 9 users as clients running both FedAvg and CDA-FedAvg, while the remaining one served as the test subject. We performed cross-validation by rotating the test participant in a 9 to 1 split, resulting in 10 iterations.

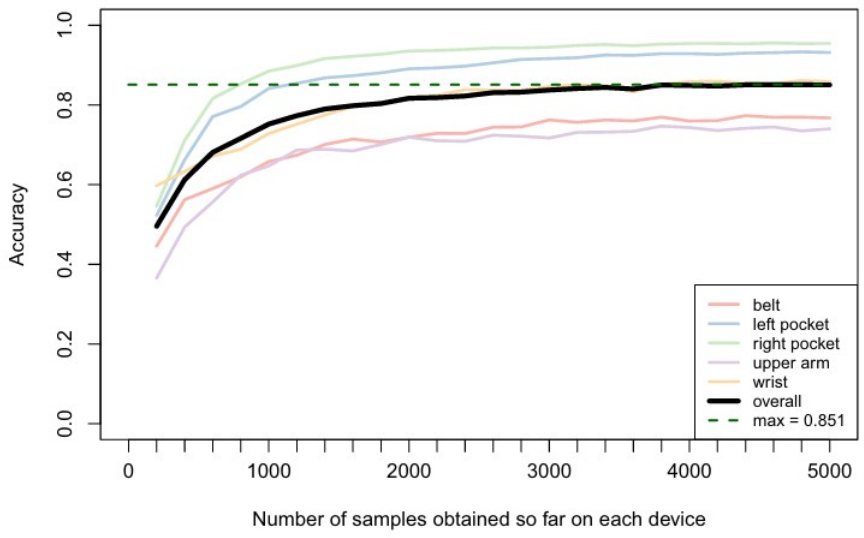


Figure 6.2: Average results on testing data for FedAvg algorithm in an IID setting. The mean accuracy is represented by a black thick line.

Phone position	Average accuracy
Belt	0.767 (±0.158)
Left pocket	0.932 (±0.092)
Right pocket	0.955 (±0.047)
Upper arm	0.740 (±0.112)
Wrist	0.858 (±0.058)
Overall	0.851 (±0.041)

Table 6.1: Final results of FedAvg in an IID setting after 5000 samples were processed by every client.

Given the small client count, all 9 users contributed to training in each federated round, rather than selecting a random subset. Raw data was segmented into windows of 124 samples (approximately 2.5 seconds). Only accelerometer and gyroscope data were used, yielding a 6-channel input comprising readings across three axes each for acceleration (a_x, a_y, a_z) and angular velocity $(\omega_x, \omega_y, \omega_z)$. Each participant contributed 5000 time windows, 1000 per phone location. Thus, every experiment involved 45000 training samples and 5000 testing samples.

For the model, we employed a Convolutional Neural Network (CNN), due to its proven success in processing inertial data. Nonetheless, CDA-FedAvg is compatible with other architectures, including feed-forward networks, recurrent models like LSTMs, or hybrid solutions. In addition, neural networks are the most widely used approach in FL strategies for their very good properties regarding data privacy.

Training an FL model using a neural networks architecture, the participants of the training process only share with the central orchestrator the parameters of the network, which are not sufficient to retrieve the original data employed to train the network. This ensures privacy of the local data and avoids data leakage.

For simplicity, we assumed synchronized data collection across all clients, with identical sampling rates.

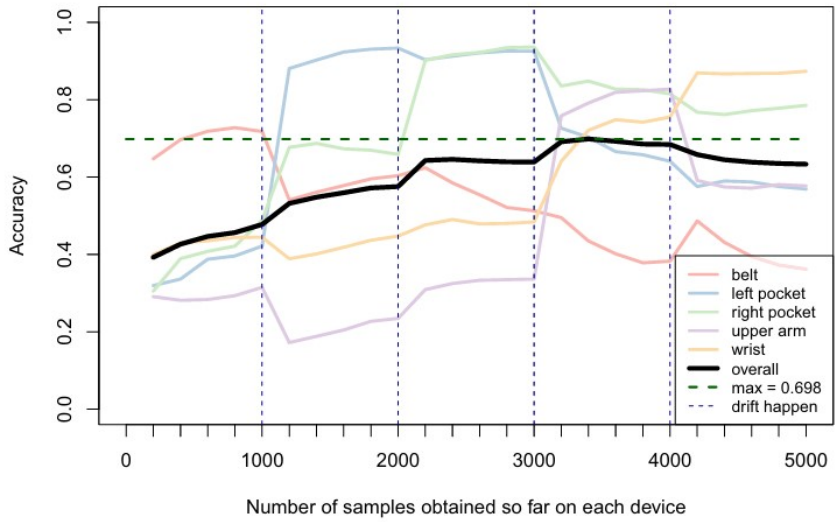


Figure 6.3: Average results on testing data for FedAvg algorithm in a non-IID setting. The mean accuracy is represented by a black thick line.

Test set	Belt	Left pocket	Right pocket	Upper arm	Wrist	Overall
User 1	0.339	0.735	0.624	0.586	0.778	0.612
User 2	0.357	0.578	0.736	0.653	0.763	0.629
User 3	0.491	0.372	0.776	0.616	0.760	0.613
User 4	0.349	0.669	0.640	0.406	0.873	0.587
User 5	0.345	0.656	0.811	0.716	0.973	0.700
User 6	0.344	0.539	0.768	0.465	0.926	0.608
User 7	0.319	0.557	0.926	0.649	0.921	0.674
User 8	0.365	0.541	0.870	0.528	0.924	0.646
User 9	0.144	0.642	0.799	0.567	0.950	0.620
User 10	0.495	0.425	0.910	0.637	0.835	0.660
Average	0.352	0.579	0.783	0.579	0.875	0.633

Table 6.2: Final results of FedAvg in a non-IID setting after every client has processed 5000 data samples.

To establish a reference point, we initially conducted an FL experiment under the simplified and unrealistic assumption that data is collected in an identical and independent manner over time and across all users. To simulate this stationary condition (IID in the temporal dimension), we randomly shuffled each user's dataset (5000 samples) without considering the phone's placement. We then ran the standard FedAvg algorithm for 25 communication rounds, involving local training and server aggregation. This process was repeated ten times, each with a different participant designated as the test user. On average, this setup resulted in test accuracy exceeding 85%. Figure 6.2 illustrates the average performance progression across these runs. The plot shows aggregated results: the thick black curve represents overall accuracy, while the thin colored lines reflect accuracy when tested on data from each specific phone location. Table 6.1 presents the final accuracies after all clients had acquired 5000 samples, broken down by phone position and averaged over the ten repetitions.

While the results in Figure 6.2 appear promising, they do not reflect real-world conditions, where data is often non-IID and evolves over time. To explore this, we conducted a second experiment in which data for all users was sorted by phone position, creating a deliberately non-stationary scenario involving four distribution shifts. Each transition between device placements introduces a change in the data distribution, i.e., a concept drift. The data was organized identically across all users: starting with the belt, followed by the left pocket, right pocket, upper arm, and finally the wrist. Within each segment (1000 samples per position), the data was randomly permuted. Although this does not fully represent realistic user behavior, it helps isolate and examine the impact of distribution shifts during training. As before, we used the standard FedAvg method with the same configuration and ran it for 25 rounds.

Figure 6.3 displays the averaged results from the 10 trials. Dashed vertical lines indicate when distribution changes occur: every 1000 samples per user. The plot shows that although the model improves overall, it experiences forgetting: as it starts adapting to the left pocket data around iteration 1000, performance on the earlier concept (belt data) drops. The final mean accuracy falls to approximately 63%. Table 6.2 summarizes results from all ten runs once each client has gathered their full dataset.

Lastly, we repeated this experiment using our proposed approach, CDA-FedAvg, replacing the standard FedAvg. In this configuration, the training process is aware of concept drift events. Unlike the earlier experiments, a single plot cannot illustrate the average accuracy trend across all ten trials, as drift detection timing varies between executions. Thus, Figures 6.4 and 6.5 show two representative examples. The complete results across all ten runs are reported in Table 6.3.

Table 6.3: Final results of FedAvg in a non-IID setting after every client has processed 5000 data samples.

Test set	Belt	Left pocket	Right pocket	Upper arm	Wrist	Overall
User 1	0.842	0.725	0.629	0.586	0.773	0.612
User 2	0.796	0.568	0.741	0.653	0.758	0.629
User 3	0.844	0.362	0.781	0.616	0.755	0.613
User 4	0.946	0.659	0.645	0.406	0.868	0.587
User 5	0.982	0.646	0.816	0.716	0.968	0.700
User 6	0.786	0.529	0.773	0.465	0.921	0.608
User 7	0.595	0.547	0.931	0.649	0.916	0.674
User 8	0.831	0.531	0.875	0.528	0.919	0.646
User 9	0.304	0.632	0.804	0.567	0.945	0.620
User 10	0.758	0.415	0.915	0.637	0.830	0.660
Average	0.769	0.569	0.788	0.579	0.870	0.633

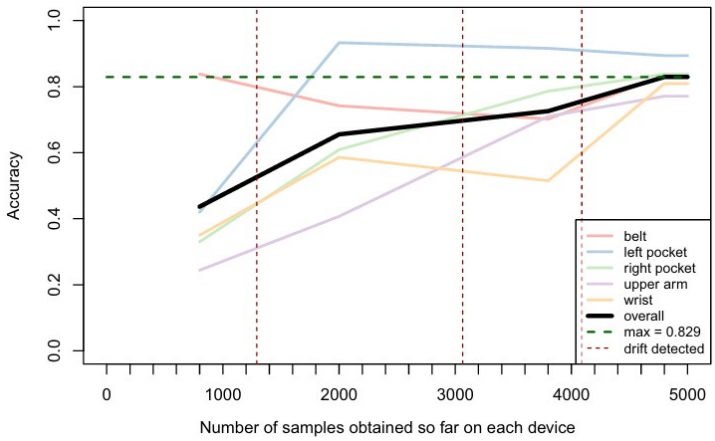


Figure 6.5: Average results on testing data for CDA-FedAvg algorithm in a non-IID setting, using all the participants data for training except for participant 3.

In this scenario, training does not commence immediately. Instead, the initial training round is delayed until each client accumulates a sufficient amount of data from

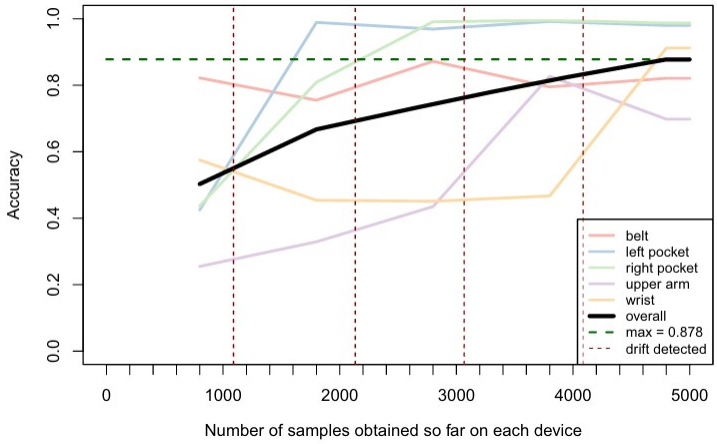


Figure 6.4: Average results on testing data for CDA-FedAvg algorithm in a non-IID setting, using all the participants data for training except for participant 8.

the initial concept (in this case, the belt position) in its long-term memory. We set the number of local training rounds per concept to 5. Consequently, with 5 distribution shifts, the total number of training rounds remains 25, matching the configuration of the prior experiment. The shifts in data distribution occur at the same sample counts as before (i.e., at iterations 1000, 2000, 3000, and 4000). However, in this case, the vertical dashed lines mark the moments when a drift is actually detected by at least one client using our drift detection mechanism. We observe that all distribution changes are identified shortly after their theoretical occurrence. In certain runs, such as the example depicted in Figure 5, the second shift goes undetected. This outcome is reasonable, considering that the transition between the left and right pocket represents a subtle change, as the data distributions are quite similar. If higher sensitivity to such subtle shifts were required, one could simply adjust the value of the λ parameter in Algorithm 5.

As indicated in Table 6.3, our approach yields a final test accuracy of approximately 82%. This result is significantly closer to the baseline scenario shown in Table 6.1. Thus, we demonstrate that CDA-FedAvg can effectively respond to evolving, nonstationary data distributions while maintaining knowledge of earlier concepts.

CHAPTER 7

Heterogeneous & Multidimensional Federated Averaging

In Section 4.1.1 we have extensively discussed the origins of information theory, paying special attention to the term of entropy, and formally defining both the Cross-Entropy loss function of a data sample (See Definition 4.1.5) and the Cross-Entropy loss function of a batch of data (See Definition 4.1.6). In this chapter, we will focus on leveraging and taking advantage of the good properties this error metric owns by definition, and we will use it for creating a new algorithm to tackle data heterogeneity over the participants of a FL process.

Non-IID data has already been discussed and classified during Chapter 5, so with this new algorithm we intend to provide a solution for all kinds of data heterogeneity across participants, that at the same time can be implemented along with the CDA-FedAvg algorithm explained in Chapter 6. By doing so, we would have a complete algorithm to tackle all kinds of data heterogeneity among the clients of an FL training framework, and also to tackle local domain data heterogeneity over time.

The key component of the FL setting we focus on is the statistical heterogeneity of the data employed for training a model. Although all participants in FL processes have a common task and objective, they gather data in different environments. In real-life applications, this non-IID data occurs all the time due to how data is collected. Training in these conditions leads to undesired results or prevents the model from converging at all.

To tackle these statistical adversities, lots of different approaches have been developed (See Chapter 5) [1]. We will focus on a branch of strategies called *clustering strategies*, which are the closest ones to our approach. They gather clients with simi-

lar data distributions in groups and then create a pseudo-global model for each group. We review and discuss most of the existing strategies for FL with clustering.

Some clustering strategies are based on model weights. For instance, Sattler et al. [136] compare the weights of the client's networks using the cosine similarity, and then clients are separated into two groups recursively until there is no need to split them further. Clients that remain together form a group and train an FL model without the intervention of clients from another group. Tian et al. [242] perform a similar strategy, with the difference that clusters are recalculated after each round of training to guarantee they are right. Briggs et al. [138] start considering that each client is a group by itself. Then, after some rounds of training, client's updates are compared and grouped together when suitable. These three proposals employ network weights to decide whether clients should belong to the same cluster or not. Methods that work with the neural network weights are opaque and lack explainability due to their abstract nature. Our approach brings some light into understanding how and why the clusters are formed.

However, this is not the only kind of approach. For instance, Xie et al. [243] propose a strategy where several global models are trained using data from random subsets of clients, and clients are associated with one of the models based on the distance between them and their local model. The global models are then updated using data from the clients associated with them. This process is repeated until convergence. Another similar approach is the one presented by Ghosh et al. [244]. They propose an algorithm that starts with one random model for each cluster. Clients are then grouped in a cluster based on which model produces the lowest loss for them. This work is similar to ours in the sense that it employs the loss function as a metric to create the clusters. As a drawback, these methods require the number of clusters to be known in advance.

There are some more works that perform clustering strategies to improve the results of federated learning. Caldarola et al. [245] develop a strategy where clients are clustered based only on their input domains, using a teacher-student classifier. Another alternative, based on Generative Adversarial Networks (GAN), was carried out by Kim et al. [246]. Clusters are calculated by a GAN that aims to minimize the variance between patterns in the same group. After that, a different model is trained for each of the groups. These last two works are based on the input space distributions of the clients $P(x)$, but do not take into account their conditional distribution $P(y|x)$.

Finally, another FL work performs a clustering of clients, and then in each round, clients from the same cluster are selected to train to accelerate the model conver-

gence [247, 248]. These works are also included in FL with clustering, although their purpose of doing clusters of clients is not to improve the final accuracy of the models.

The most significant contributions of this chapter are in review state, pending to be published in the journal Information Sciences.

The rest of the Chapter is organized as follows: Section 7.1 explains the process to achieve the Heterogeneous and Multidimensional Cross-Entropy Loss, a novel loss function that will be useful to group the participants of the training process according to their similarities. Next, Section 7.2 details the algorithm of the learning strategy Heterogeneous and Multidimensional Federated Averaging, which forms clusters of participants to increase the accuracy of the global model and trains a different global model for each of the clusters, aiming for a more personalized outcome. Finally, Section 7.3 presents some experiments to empirically show how this algorithm improves the result of other well-known algorithms, such as FedAvg, FedProx, and CFL (Clustered Federated Learning) algorithms.

7.1 Heterogeneous & Multidimensional Cross Entropy Loss

In this section, we introduce a novel loss function derived from the Cross-Entropy Loss, which we term the *Heterogeneous and Multidimensional Cross-Entropy Loss*. This loss function offers several meaningful and advantageous characteristics, making it a valuable tool for clustering purposes. Our goal is to cluster clients based on the similarities of their data distributions, considering both $P(x)$ and $P(y|x)$.

A large portion of existing clustering approaches in Federated Learning (FL) utilize neural network parameters or gradient information to compute similarities between clients, grouping those with analogous patterns. While these strategies are effective, they suffer from a key limitation: the mapping between model parameters and output behaviors is often opaque. Clients with significantly distinct weights might still produce similar predictions, and vice versa, due to the non-convex nature of the optimization landscape [249, 250].

Consequently, we propose leveraging the loss function itself as a metric for clustering. By doing so, we avoid enforcing homogeneity in local model weights across clients. Additionally, this method introduces greater clarity and interpretability into the grouping mechanism. The ultimate aim of clustering in FL is to build models tailored to the unique data distribution of each client. Nevertheless, these personalized models may not generalize effectively to unseen data. This presents a critical trade-off: should we favor broadly applicable models that underperform for certain

clients, or develop highly customized models that may fail in other contexts? The answer depends on the specific application scenario, as client priorities may differ. Our approach offers flexibility for clients to opt for a more generalized model or a specialized one according to their preferences.

To achieve this, we suggest using the model's loss function as the basis for the clustering process. However, the scalar output from the standard Cross-Entropy loss (See Equation (4.1.1)) encapsulates only a coarse assessment of the client's performance across its dataset. This singular value lacks granularity and may fail to capture nuanced issues, such as class imbalance or mislabeled entries. To overcome these limitations, we propose a new metric that maintains the advantages of Cross-Entropy from an information-theoretic perspective but delivers a more detailed view of model behavior across distinct data segments.

We construct a metric represented by a vector rather than a single value, thereby enriching the information available for clustering. Furthermore, we outline three essential properties that this metric must satisfy.

The first desirable trait is the ability to *identify where client-specific errors are concentrated*. For instance, some clients may consistently classify samples from particular classes incorrectly, possibly due to skewed class distributions or model bias. To address this, we formulate a class-wise error function, resulting in a vector that quantifies the error for each individual class.

Let $\mathcal{N} = \{1, \dots, N\}$ denote the set of clients collaborating in training a Federated Learning model. Each client possesses a local dataset D_i of size M_i, and we assume that $D_i \cap D_j = \emptyset$ for all $i, j \in C$, i.e., client datasets are mutually exclusive and private.

Recall that y_k is a one-hot encoded vector with C entries, only one of which is set to 1 to indicate the true label. Therefore, we define, for each class $s \in C$, and each batch $X_B = \{x_k, y_k\}_{k=1}^{B} \subset D_i$ from client i, the following error component:

$$\ell_{is}(X_B) = - \sum_{\substack{(x_k, y_k) \in X_B \\ y_{k_s} = 1}} y_{k_s} \cdot \log(f_s(x_k)),$$

where f represents the global model. Although the samples are filtered to include only those belonging to class s, we retain the y_{k_s} term to maintain visual and structural consistency with the traditional Cross-Entropy formulation. It does not affect the result and could be omitted.

Each ℓ_{is} quantifies the model's error for client i specifically on samples from class s. By computing all such components for $s \in \{1, \dots, C\}$, we form a complete

error profile for client i across classes. This can be organized into a vector $L_i = (\ell_{i1}, \ell_{i2}, \ldots, \ell_{iC})$, and notably:

$$\mathcal{L}_{CE_i}(D_i) = \sum_{s=1}^{C} \ell_{is}(D_i).$$

Thus, the total Cross-Entropy loss is the sum of the vector components, each corresponding to a class-specific error. This decomposition provides a more insightful and discriminative representation for clustering.

The second critical property we enforce is *penalization of misclassification*. In particular, if two samples share the same confidence in the correct class, but only one is correctly predicted, they should not contribute equally to the loss. It is intuitive that incorrect predictions deserve higher penalties, especially as the predicted probability diverges from the ground truth.

To model this, we introduce a penalty function $H_s(x)$ for incorrect classifications. This function should exceed one for errors and increase as the model's confidence in the true label decreases. It is defined as:

$$H_s(x) = \frac{1}{1 - \left(\max_{j=1,\ldots,C} (f_j(x)) - f_s(x) \right)}.$$

Observe that for correctly classified instances, the maximum confidence aligns with the true class, i.e., $\max_j f_j(x) = f_s(x)$, leading to $H_s(x) = 1$. This ensures that correct predictions are unaffected, while incorrect ones are penalized. Incorporating this factor, the error vector for client i becomes:

$$\begin{aligned} L_i(D_i) &= (\ell_{i1}(D_i), \ell_{i2}(D_i), \ldots, \ell_{iC}(D_i)), \\ \ell_{is}(D_i) &= - \sum_{\substack{(x_k, y_k) \in D_i \\ y_{k_s} = 1}} H_s(x) \cdot \log(f_s(x_k)). \end{aligned} \tag{7.1.1}$$

The final aspect we must consider is *statistical normalization*. Clients with disproportionately large quantities of data from specific classes will naturally accumulate more error in those categories, regardless of model performance:

$$||\{(x_k, y_k) \in D_i | y_{k_s} = 1\}|| \gg ||\{(x_k, y_k) \in D_i | y_{k_t} = 1\}|| \Rightarrow l_{is}(D_i) \gg l_{it}(D_i).$$

This phenomenon can bias the clustering process based on data imbalance rather than inherent distributional differences. While in some contexts this may be acceptable, for example a client may not care about underrepresented classes, it generally undermines the goal of constructing well-generalized models.

To address this, we apply a measure of central tendency, such as the mean or median, to dampen the effects of data quantity disparities. We opt for the median, as it offers greater robustness against outliers. Accordingly, we finalize our error metric as:

$$\begin{aligned} \overline{L_i}(D_i) &= (\overline{\ell_{i1}}(D_i), \overline{\ell_{i2}}(D_i), \dots, \overline{\ell_{iC}}(D_i)), \\ \overline{\ell_{is}}(D_i) &= \underset{(x_i,y_i)\in D_i}{\text{median}} (\ell_{is}(D_i)). \end{aligned} \tag{7.1.2}$$

This formulation defines the *Heterogeneous and Multidimensional Cross-Entropy Loss* (H&M-CEL), which serves as the core of our proposed client clustering technique in the Federated Learning framework.

7.2 H&M-FedAvg algorithm

Our proposed methodology for structuring the Federated Learning (FL) training process, referred to as H&M-FedAvg, is presented below. Further implementation details are included in Algorithm 7. As a first step, we define a fixed number of total training iterations, denoted by R. We then perform an initial sequence of training steps, denoted as $\hat{r}$ where $\hat{r} < R$, by applying the conventional *FedAvg* algorithm. This initial phase (lines 2–7) serves to stabilize the model so that it can produce predictions that broadly reflect the global data distribution $P(x, y)$, enabling us to compute reliable and representative error vectors for each participating client.

Subsequently, each client evaluates its own error profile (lines 9–10) using the global model $\mathbf{w}_G^{\hat{r}}$ and its local dataset. Once these individual error vectors are computed, the central server applies the DBSCAN clustering algorithm [251] (line 11), a density-based technique that does not necessitate predefining the number of clusters. The particular distance metric used to group clients is determined by the objective of the model: whether it seeks generalization or specialisation. We elaborate further on this selection below.

As a result of this clustering step, the server obtains a set of F distinct client clusters characterized by similar error patterns (line 12). For each of these clusters, the server generates a replica of the current global model $\mathbf{w}_G^{\hat{r}}$ (lines 13–14), yielding

F identical models, namely $\{\mathbf{w}_{G,t}^{\hat{r}}\}_{t=1}^{F}$. In the subsequent training stages, the server delivers to each client the version of the global model corresponding to its assigned cluster. Once updates are received from the selected clients within each group, they are aggregated into the cluster-specific model (lines 15–21).

Algorithm 7: H&M-FedAvg.

Input : N clients, each holds M_i samples. Number of training clients per round, n, number of rounds before clustering, $\hat{r}$, and total number of rounds R.

Output: Optimal models for the clusters, $\mathbf{w}_{G,1}^{R}, \mathbf{w}_{G,2}^{R}, \dots$

1 Initialization: Random parameter initialization $\mathbf{w}_G^0$
2 **for** *r in* $\{1, \dots, \hat{r}\}$ **do**
3 $C_r \leftarrow$ random subset of n clients;
4 **for** *each client* i *in* C_r *in parallel* **do**
5 $\mathbf{w}_i^r \leftarrow$ update $\mathbf{w}_G^{r-1}$ using SGD;
6 $M \leftarrow \sum_{i \in C_r} M_i$;
7 $\mathbf{w}_G^r \leftarrow \sum_{i \in C_r} \frac{M_i}{M} \mathbf{w}_i^r$;
8 **if** $r = \hat{r}$ **then**
9 **for** *each client* i **do**
10 $e_i \leftarrow$ vector of errors, with H&M-CEL (Eq. 7.1.2);
11 $D \leftarrow \text{DBSCAN}(\{e_i\}, \epsilon, MinPts)$;
12 $F \leftarrow max(D)$;
13 **for** t *in* F **do**
14 $\mathbf{w}_{G,t}^{\hat{r}} \leftarrow \mathbf{w}_G^{\hat{r}}$;
15 **for** r *in* $\{\hat{r}+1, \dots, R\}$ **do**
16 **for** t *in* F **do**
17 $C_{r,t} \leftarrow$ random subset of $\lceil n/F \rceil$ clients from cluster t;
18 **for** *each client* i *in* $C_{r,t}$ *in parallel* **do**
19 $\mathbf{w}_i^r \leftarrow$ update $\mathbf{w}_{G,t}^{r-1}$ using SGD;
20 $M \leftarrow \sum_{i \in C_{r,t}} M_i$;
21 $\mathbf{w}_{G,t}^r \leftarrow \sum_{i \in C_{r,t}} \frac{M_i}{M} \mathbf{w}_i^r$;
22 **return** $\mathbf{w}_{G,1}^{R}, \mathbf{w}_{G,2}^{R}, \dots, \mathbf{w}_{G,F}^{R}$;

Upon completion of all R training rounds, the central server retains the final cluster-specific models: $\mathbf{w}_{G,1}^R, \mathbf{w}_{G,2}^R, \ldots, \mathbf{w}_{G,F}^R$ (line 22). Each client receives the final version of the global model that corresponds to the cluster it belongs to.

The key distinction of this approach from the traditional FedAvg algorithm [17] is the clustering mechanism, which we now describe in greater detail. The DBSCAN algorithm relies on two essential parameters: the minimum number of clients that can constitute a cluster, denoted *MinPts*, and a threshold parameter ϵ, representing the maximum distance two clients can be apart to be considered part of the same group.

We set *MinPts* to 1. This decision is crucial because choosing a larger value (e.g., $MinPts > 1$) might result in clients that are outliers being grouped under the same label, -1, which indicates they do not belong to any actual cluster. These clients would then proceed to collaboratively train a model, which would be incorrect. Setting *MinPts* to 1 ensures that such outlier clients will be treated individually, each forming a separate cluster and continuing with their own localized training. While these clients won't leverage collaboration, this is acceptable since their data distributions are not comparable to others in the system.

Determining an appropriate ϵ value can be effectively handled using the elbow technique. This involves computing the distance from each client to its closest *MinPts* neighbors, ranking and visualizing these distances, and then identifying the inflection point: the location where the curve bends most sharply. This inflection provides a natural choice for ϵ. It is important to emphasize that this selection is sensitive to the type of distance metric adopted, and the outcome will differ accordingly.

When the training objective is to obtain a highly generalizable model, the focus should be on variations in the conditional distribution $P(y|x)$ [1]. In this case, we employ a distance function that captures the largest per-class deviation between clients, known as the Manhattan distance. Conversely, if the goal is to tailor models to distinct client domains (i.e., particularization), differences in the input distribution $P(x)$ are also relevant. For this, the Euclidean distance is a better fit, as it encapsulates the overall magnitude of discrepancies.

Formally, given two error vectors $\overline{L_i} = (\overline{\ell_{i1}}, \ldots, \overline{\ell_{iC}})$, and $\overline{L_j} = (\overline{\ell_{j1}}, \ldots, \overline{\ell_{jC}})$, we define the distance metrics for generalization and particularization as follows:

$$dist_g(\overline{L_i}, \overline{L_j}) = \max_{k=1,\ldots,C} ||\overline{\ell_{ik}} - \overline{\ell_{jk}}||,$$

$$dist_p(\overline{L_i}, \overline{L_j}) = \left(\sum_{k=1}^{C} (\overline{\ell_{ik}} - \overline{\ell_{jk}})^2 \right)^{1/2}.$$

The $dist_g$ metric (Manhattan distance) is especially useful for identifying large class-specific error gaps. Our empirical findings indicate that this metric groups clients with similar conditional distributions $P(y|x)$, even when their $P(x)$ differs. On the other hand, the $dist_p$ metric (Euclidean distance) enables more fine-grained differentiation and can separate clients based solely on differences in $P(x)$. This proves advantageous in scenarios where models are deployed under consistent domain assumptions. Alternatively, $dist_g$ excels when facing unpredictable domains, fostering broader generalization. These behaviors will be explored in more detail in our experimental analysis.

In the following section 7.3, we highlight some properties of the H&M-FedAvg and empirically show its performance in realistic scenarios with heterogeneous clients and datasets that present differences in $P(x)$ and $P(y|x)$. In addition, we compare the results obtained with those obtained under the same circumstances by other learning algorithms.

7.3 Results and Discussion

In this section, we evaluate the performance of our algorithm across a variety of heterogeneous environments and benchmark it against several well-established Federated Learning (FL) techniques. All experiments utilize the *Digit-five* dataset, a widely adopted benchmark for digit classification tasks. This composite dataset integrates MNIST [133], MNIST-M [252], SVHN [178], USPS [177], and Synthetic Digits [154]. As illustrated in Figure 7.1, the input images differ significantly across the sources, showcasing a diverse set of visual features. Each dataset within Digit-five represents a separate domain, corresponding to a distinct input distribution $P(x)$. For example, while some datasets consist of grayscale digits, others contain colored and texture-rich images. Additionally, in order to introduce heterogeneity in the label distribution $P(y|x)$, we manually alter the labels of a subset of samples in specific experiments. Overall, we craft a variety of settings that exhibit varying degrees of non-IID behavior, affecting both $P(x)$ and $P(y|x)$.

To ensure fair comparisons and maintain balance across domains, we fixed the sample count at 60000 per dataset, resulting in a total of 300000 data instances. These were evenly distributed among 40 clients, with each client receiving 7500 data points. However, the specific allocation strategy varied depending on the experimental scenario, allowing us to simulate different levels and types of heterogeneity.

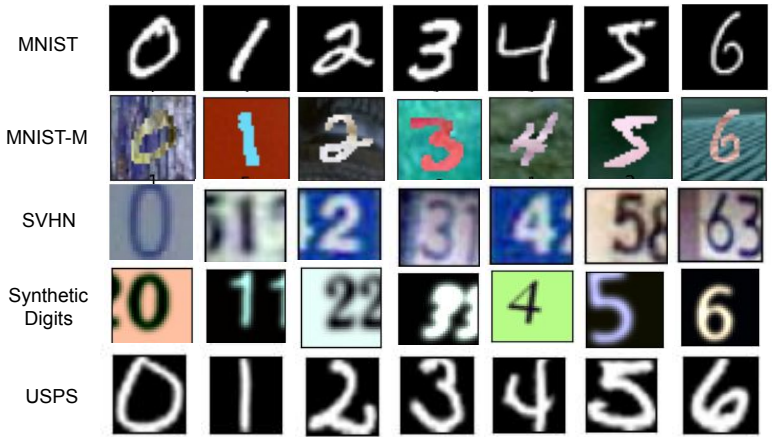


Figure 7.1: Samples of digits 0 to 6 from each dataset that constitute Digit-five.

Our evaluation includes two main objectives: assessing the clustering quality and measuring the accuracy of the learning process itself. For the clustering component, we compute each client's error vector using the H&M-CEL metric and then apply the DBSCAN algorithm to those vectors. Since the classification task involves 10 classes, the resulting error vectors are 10-dimensional. To visualize this high-dimensional data, we perform Principal Component Analysis (PCA), extract the top two principal components, and project them into a two-dimensional plot.

We further compare our model's predictive performance within each cluster to that of several alternative methods, namely FedAvg [17], FedProx [74], and CFL (Clustered Federated Learning) [136]. As defined in Algorithm 7, we set the number of rounds before clustering to $\hat{r} = 4$, and the total number of rounds to $R = 10$. For CFL, we also perform clustering every 4 rounds to mirror our setup and ensure a fair comparison. All competing approaches use the same Convolutional Neural Network (CNN) architecture, which is also employed during the training of each cluster-specific model. As mentioned in previous chapters, training an FL model using a neural networks ensures privacy of the local data and avoids data leakage.

The experimental design includes three distinct non-IID scenarios, in addition to one baseline IID setting, in order to showcase the adaptability and robustness of H&M-FedAvg. Each non-IID setting explores a different dimension of heterogeneity among the clients: variation in input distributions (Scenario A), divergence in both

inputs and labels (Scenario B), and fully incompatible data distributions (Scenario C). Each scenario is explained in greater detail below. To increase statistical reliability and mitigate randomness, we repeated every experiment five times. Table 7.1 summarizes the mean classification accuracy and variance for all methods and scenarios.

		Baseline scenario	Scenario A	Scenario B	Scenario C
FedAvg		**93.2% (± 0.5%)**	**81.4% (± 1.2%)**	**55.4% (± 0.8%)**	**33.1% (± 1.2%)**
FedProx		**93.3% (± 0.3%)**	**83.7% (± 0.9%)**	**57.8% (± 0.6%)**	**37.4% (± 0.8%)**
Clustered Federated Learning	**Mean**	**93.3% (± 0.5%)**	**88.2% (± 1.3%)**	**86.3% (± 1.4%)**	**81.8% (± 1.1%)**
	Cluster 1	93.3% (± 0.5%)	91.1% (± 0.5%)	89.3% (± 0.5%)	84.7% (± 0.3%)
	Cluster 2	-	89.5% (± 0.6%)	84.7% (± 0.9%)	82.9% (± 0.7%)
	Cluster 3	-	87.5% (± 0.3%)	84.5% (± 0.8%)	82.4% (± 0.6%)
	Cluster 4	-	86.7% (± 0.6%)	-	77.2% (± 0.3%)
H&M-FedAvg (Euclidean distance)	**Mean**	**93.2% (± 0.4%)**	**91.6% (± 0.9%)**	**90.8% (± 0.8%)**	**87.9% (± 1.1%)**
	Cluster 1	93.2% (± 0.4%)	92.9% (± 0.4%)	94.6% (± 0.3%)	91.8% (± 0.4%)
	Cluster 2	-	91.6% (± 0.3%)	89.1% (± 0.5%)	91.3% (± 0.6%)
	Cluster 3	-	91.3% (± 0.4%)	88.7% (± 0.6%)	90.8% (± 0.4%)
	Cluster 4	-	90.6% (± 0.6%)	-	86.9% (± 0.3%)
	Cluster 5	-	-	-	81.1% (± 0.7%)
H&M-FedAvg (Manhattan distance)	**Mean**	**93.2% (± 0.5%)**	**81.9% (± 1.0%)**	**87.6% (± 1.0%)**	**87.3% (± 1.2%)**
	Cluster 1	93.2% (± 0.5%)	81.9% (± 1.0%)	90.4% (± 0.6%)	91.1% (± 0.5%)
	Cluster 2	-	-	83.8% (± 0.7%)	90.6% (± 0.4%)
	Cluster 3	-	-	-	86.2% (± 0.4%)
	Cluster 4	-	-	-	81.3% (± 0.6%)

Table 7.1: Results obtained by the different learning algorithms. The accuracies shown are expressed in terms of the mean and the variance in the 5 executions. Blank spaces indicate that a cluster does not exist in a particular scenario.

The **baseline scenario** simulates a fully IID environment, where every client receives data that follows the same statistical distribution. Specifically, each client is assigned 1500 training samples from each of the five domains, totaling 7500 data points per client. In this setting, both the input distributions $P(x)$ and the conditional distributions $P(y|x)$ are consistent and unaltered across clients. Thus, the joint distribution $P(x, y)$ is effectively identical for all participants:

$$P_i(x, y) \sim P_j(x, y) \text{ for all } i, j \in \{1, \ldots, N\}.$$

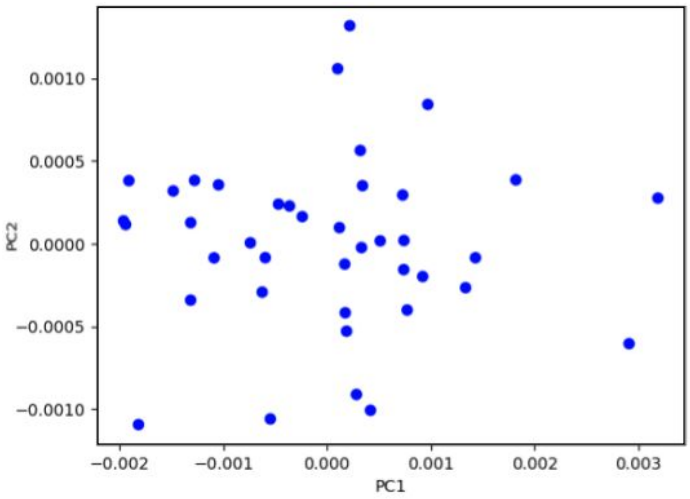


Figure 7.2: Clustering results using either one of the distance functions.

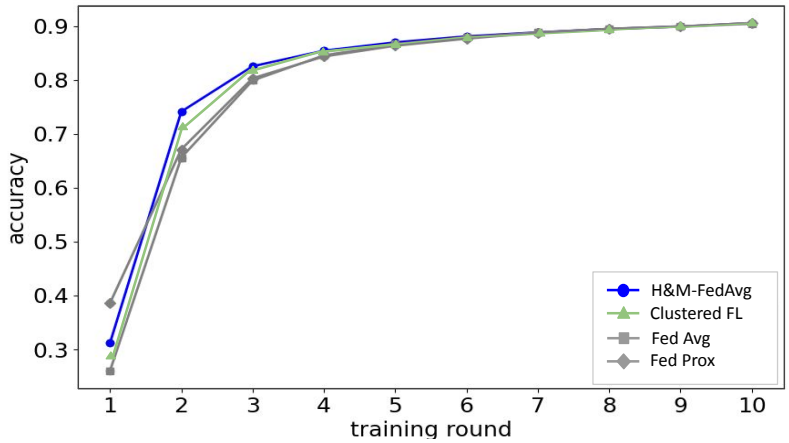


Figure 7.3: Accuracies obtained on the baseline scenario.

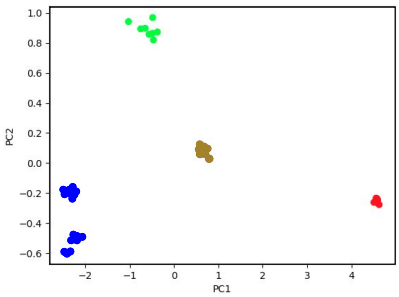
For this reason, our clustering algorithm detects only one cluster, regardless of the distance function employed in clustering. All clients should remain together after the clustering split (See Figures 7.2 and 7.3). Since our method performs FedAvg for

each of the clusters detected when working with only one cluster, our algorithm will behave exactly as FedAvg. We can observe that the accuracies obtained are almost the same for all of the learning algorithms: around 93% (See also Table 7.1).

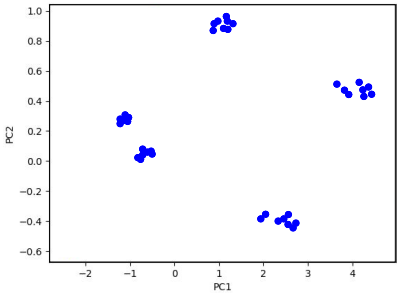
In Figures 7.2 and 7.3, the results of the experiments in the Baseline scenario can be observed. Figure 7.2 represents the points of the error vectors used for clustering, whereas Figure 7.3 shows the accuracies of the different learning algorithms.

For **Scenario A**, we designed skew data distributions as follows. As in the baseline scenario, we have not modified the data labels in this case. Each client will have 7500 samples, all belonging to the same domain, leading to statistically different $P(x)$ distributions between the clients. There will be eight clients owning data samples from MNIST, another eight from SVHN, and so on.

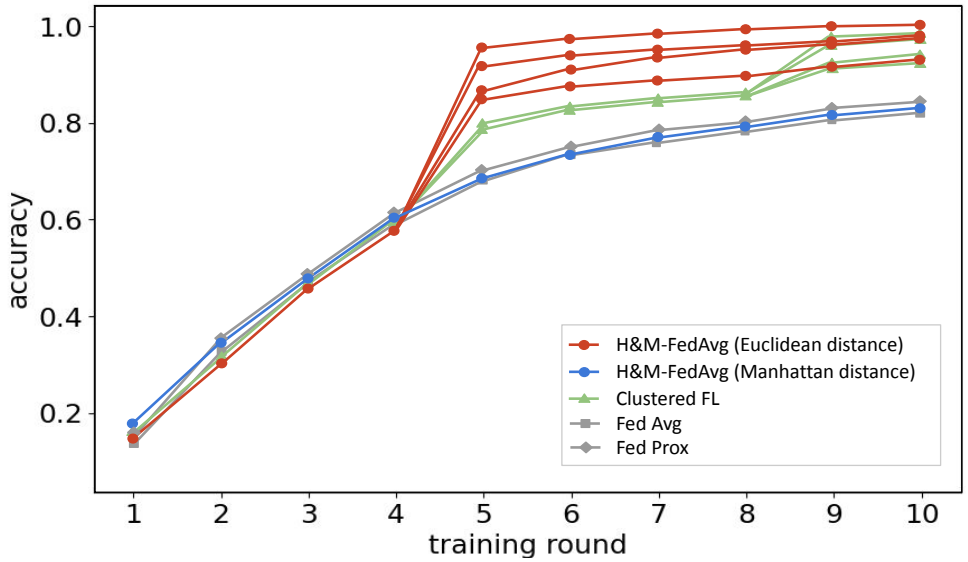
As a result, we ended up with a total of 5 joint global distributions, each of them represented by eight clients. However, those distributions are not contradictory in the sense that the goal for all of them is to label their samples in the same way. For this reason, if we train our models running the H&M-FedAvg with the Manhattan distance, the expected result is to have just 1 cluster. On the other hand, if we run the H&M-FedAvg with the Euclidean distance, the clustering process splits the clients into groups according to their input domain distribution. The results achieved in this scenario can be seen in Figure 7.4.



(a) Clustering results using the Euclidean distance function.



(b) Clustering results using the Manhattan distance function.



(c) Accuracies reached by all of the learning algorithms when performing on Scenario A.

Figure 7.4: Scenario A results. On top, we see the points of the error vectors used for clustering. At the bottom, the accuracies of the different clusters and learning algorithms.

In Figures 7.4a and 7.4b, we can see the two ways our algorithm has computed the clusters: Euclidean distance on the left and with Manhattan distance on the right. For

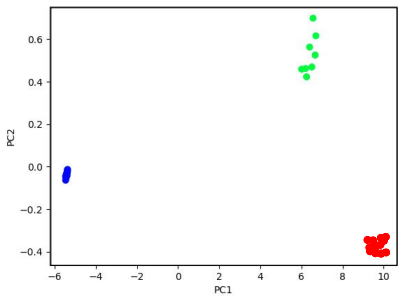
the first, our method detects four different groups of clients. We notice that there are two different clouds of points, corresponding to clients that own samples from MNIST and USPS, which remain together after the split because they are very similar, both presenting textureless black and white images. Concerning the generalization method, we can observe that only 1 group of clients is formed, and the training process is totally analogous to the one of FedAvg.

The accuracies achieved are shown in both Figure 7.4c and Table 7.1. Notice that since CFL only splits the clients in 2 each time it performs clustering, it takes more time to determine the four groups and for that reason, its accuracy is lower than H&M-FedAvg with Euclidean distance. It is also important to note that these methods get higher global accuracy because they compute more global models, which learn to classify different subsets of the whole data. In contrast, the other methods compute a unique more general model. Each strategy could be more suitable depending on the circumstances of the problem being solved.

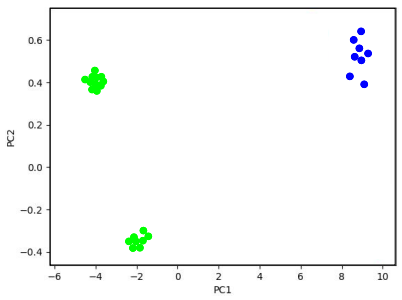
Scenario B is designed to explore unbalanced distributions. We have set a majority group, formed by 24 out of the 40 clients, which hold data samples from MNIST-M, Synthetic digits, and USPS, equally distributed. On the other hand, eight clients own only samples from MNIST for both training and testing, without altering their labels. The remaining eight clients have SVHN data, with all their labels shifted.

The purpose of this experiment is to show that clients who do not match the distribution of the majority are going to get poor results, and the global accuracy will drop. That drop will be more notorious for changes in the conditional distribution $P(y|x)$. However, when using clustering strategies, this drawback can be overcome. In this case, the correct split if we are looking for better local performance would be to split the clients into 3 clusters according to the three domains they perceive. On the contrary, if we want more adaptable models, the right split of clients would be in 2 groups, corresponding to the two ways of labeling the data samples.

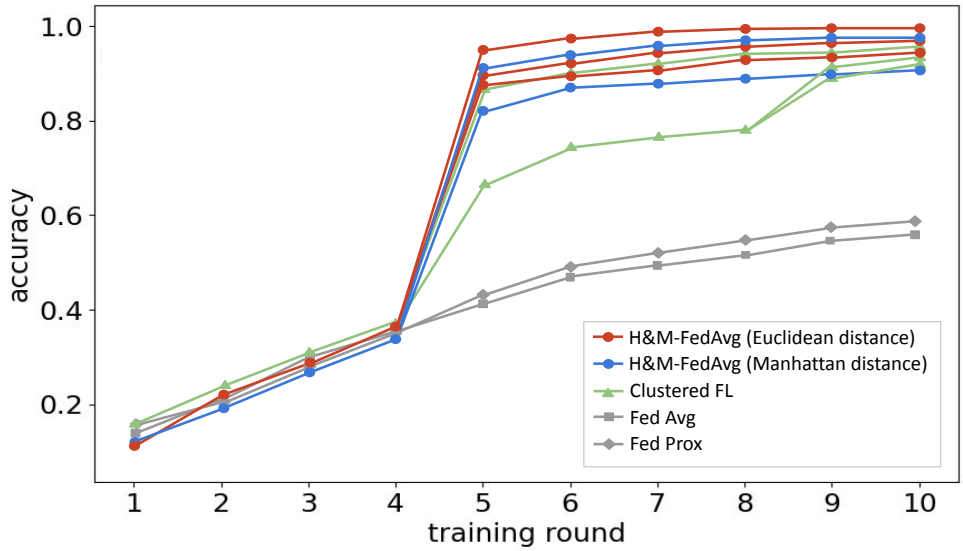
The results obtained with this display are shown in Figure 7.5. Concerning the clustering, we can see that both of our methods behave as expected, finding the proper clusters in each situation (Figures 7.5a and 7.5b). CFL behaves as H&M-FedAvg with the Manhattan distance and finds the same groups, thus splitting clients into three groups.



(a) Clustering results using the Euclidean distance function.



(b) Clustering results using the Manhattan distance function.



(c) Accuracies reached by all of the learning algorithms when performing on the Scenario B.

Figure 7.5: Scenario B results. On top, we see the points of the error vectors used for clustering. At the bottom, the accuracies of the different clusters and learning algorithms.

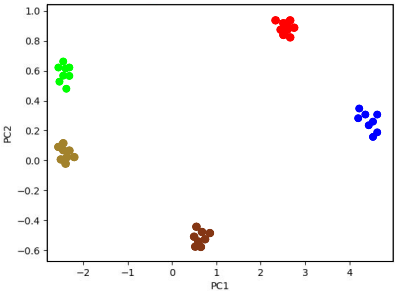
This scenario presents more of a challenge if we pay attention to the accuracy of each algorithm (Shown in Figure 7.5c). Both FedAvg and FedProx drop below 60%, caused by the labeling mismatch between distributions. The clustering strategies

achieve around 90% precision (See Table 7.1). CFL gets 86.3%, which is slightly improved by both of our H&M-FedAvg algorithms. They get 87.6% and 90.8% of accuracy.

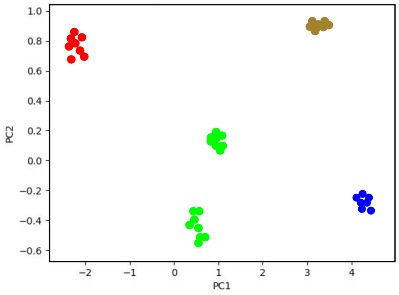
Lastly, we designed **scenario C**, which presents a higher degree of heterogeneity in $P(y|x)$. For this scenario, we shifted all of the labels of the samples belonging to MNIST, MNIST-M, and Synthetic Digits in 3 different ways, 1 for each of them. Next, each client is given samples from only 1 of the domains. This data division among the clients leads to completely skewed distributions both in $P(x)$ and $P(y|x)$, except for the clients that own SVHN and USPS samples, who share the same $P(y|x)$, but even for them the data distribution of $P(x)$ is very different, as these two domains present very distinctive samples.

Under these circumstances, standard FedAvg and FedProx will produce a dreadful model, which cannot properly classify the data. In fact, the results depicted in Figure 7.6c show that their accuracy is below 40%. On the contrary, algorithms that split clients into clusters can find a series of global models appropriate for each distribution. Using the Manhattan distance for clustering (i.e. aiming to generalize), our method detects four groups of clients, representing the four different ways of labeling our data. Our other approach, using the Euclidean distance, finds 5 clusters of clients as expected.

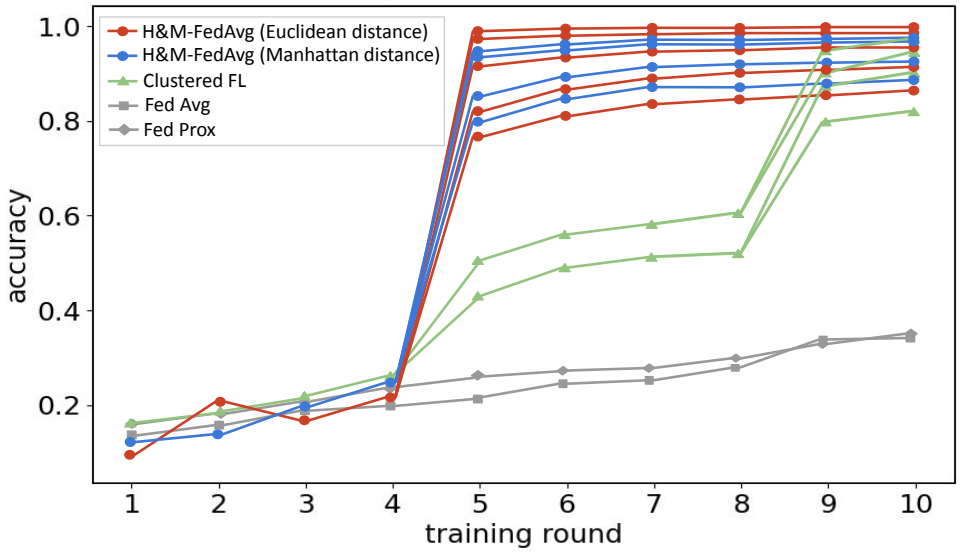
The results obtained in scenario C, for both clustering and model performance, can be observed in Figure 7.6 and Table 7.1. Apart from FedAvg and FedProx, all other methods achieve successful results. CFL works similarly to our H&M-FedAvg using the Euclidean distance, but as we are training for 10 rounds, CFL cannot detect more than four groups because it makes binary splits each of the four rounds. CFL gets an accuracy of 81.8%, and H&M-FedAvg with the Euclidean distances gets an 87.9%. H&M-FedAvg with the Manhattan distance gets similar results, achieving 87.3% accuracy. Like in the previous scenarios, the accuracy obtained is statistically similar, the usage of one or the other depends on the problem itself.



(a) Clustering results using the Euclidean distance function.



(b) Clustering results using the Manhattan distance function.



(c) Accuracies reached by all of the learning algorithms when performing on the Scenario C.

Figure 7.6: Scenario C results. On top, we see the points of the error vectors used for clustering. At the bottom, the accuracies of the different clusters and learning algorithms.

General Conclusions

In this section we review and further analyze the results obtained through the experiments in the different chapters of the dissertation.

Regarding Chapter 5, the focus of the experiments performed is to show that the constraints detailed in the previous sections of the chapter are indeed necessary, and that the associated heterogeneity cannot be effectively managed without them. This experiments also show how different manifestations of non-IID data degrade model quality and hinder performance.

Experiments are performed in two different directions, presenting scenarios that involve spatial non-IID distributions (Section 5.4.1) and those involving temporal non-IID distributions (Section 5.4.2). In both cases, all experiments utilize the *Digit-five* dataset, which aggregates MNIST, MNIST-M, SVHN, USPS, and Synthetic datasets. This approach allows to simulate domain shift in $P(x)$ across clients and time without manually modifying the distributions.

It is important to highlight that performing experiments over only this dataset is not enough to extrapolate all of the conclusions observed to a general framework. Each problem in FL has its own particularities and must be studied in depth to correctly assess a proper algorithmic solution that gets the most of the data collected and achieve the best possible results. However, this settings are useful to verify the constraints presented and show how the performance gets worse for standard FL algorithms in the presence of non-IID data.

Across all of the experiments, the model used is a basic Convolutional Neural Network (CNN) composed of four convolutional layers followed by three fully connected layers. Employing neural networks is a common practice in FL algorithms that improves the privacy preservation, as the parameters of the network do not present enough information to retrieve the initial data samples employed to train it. Follow-

ing this approach, each participant of the training process shares its parameters with the central orchestrator, maintaining its local data private.

With the results obtained both in spatial and temporal settings, it is clear that FedAvg, an standard state-of-the-art algorithm for FL, is seriously affected by the presence of non-IID data across its participants and also over time, and the performance achieved can be improved with algorithms specifically designed for those kind of scenarios, leveraging the information provided by the constraints we set previously.

Concerning Chapter 6, in this case we are focusing on the temporal dimension of non-IID data, presenting situations were concept drift stands as the main difficulty to achieve a high accuracy in the global model obtained through FL strategies. This chapter also presents a novel algorithm for FL in Continual settings, CDA-FedAvg, and the objective of the experiments performed is to showcase how this algorithm outperforms its standard counterpart, FedAvg.

To evaluate CDA-FedAvg, we selected a challenging real-world scenario: Human Activity Recognition (HAR) using smartphone data. This dataset comprises sensor readings for seven activities: sitting, standing, walking, jogging, climbing stairs, descending stairs, and cycling. A distinctive feature of this dataset is that each activity was recorded with the phone placed in five different body positions, posing a clear source of heterogeneity. For this dataset, 10 male volunteers aged between 25 and 30 were monitored while performing each of the designated tasks for roughly 20 minutes (4 minutes per each placement of the mobile phone). All trials took place indoors within the same building, except for the activity biking, which occurred outdoors.

To employ this dataset in our time-evolving framework, samples were ordered in each of the participants according to the mobile phone position, allowing for concept drift appearances along the training process. It is important to mention that, although the data collection process of this dataset was configured to get a realistic set of data that could fit into a real-world problem, it is not a sufficiently general task to be able to extrapolate the results obtained for every other FL problem that involves a Continual evolution over time. As we already mention, each problem in real-life should be treated and studied specifically to get the most out of it.

However, this settings expose a clear realistic scenario where concept drift damages the performance of standard FedAvg, in contrast with the CDA-FedAvg algorithm, that maintains a high overall accuracy and poses itself as a great alternative to be taken into account for this kind of heterogeneous scenarios.

Finally, the experiments presented in Chapter 7 focus on spatial non-IID settings, where the data distribution varies in different proportions across the participants. A bunch of different scenarios are presented to illustrate both compatible distributions

with domain shift and also contradictory behaviors between the users. We evaluate the performance of our novel algorithm H&M-FedAvg and compare it to other state-of-the-art techniques for both standard FL (FedAvg) and heterogeneous settings in FL (FedProx and Clustered Federated Learning).

All experiments utilize the *Digit-five* dataset, a widely adopted benchmark for digit classification tasks that was already explored in Chapter 5. The input images of this dataset differ significantly across the five sources employed, showcasing a diverse set of visual features. Each dataset within Digit-five represents a separate domain, corresponding to a distinct input distribution $P(x)$. Additionally, heterogeneity in the label distribution $P(y|x)$ is introduced by manually altering the labels of some samples. Overall, we craft a variety of settings that exhibit varying degrees of non-IID behavior, affecting both $P(x)$ and $P(y|x)$. By doing so, we can measure the impact of heterogeneous data at different scales and show the benefits of employing dedicated algorithms.

The evaluation in this case includes one additional purpose, apart from the obvious accuracy check of the models obtained: assessing the clustering part of the algorithms. Each client's error vector is computed using the H&M-CEL metric and then applying the DBSCAN algorithm to those vectors.

We further compare our model's predictive performance within each cluster to that of the algorithms mentioned before. All competing approaches use the same Convolutional Neural Network (CNN) architecture for both the global model and each of the cluster models. Based on the results obtained, H&M-FedAvg gets impressive results, outdoing its competitors in both the speed of the clustering formation, and also the accuracy obtained. It is important to mention that this does not mean that H&M-FedAvg will provide better results for every problem in FL. As we already remarked, each FL problem may have its own particularities and it is important to understand them in order to refine these strategies and get the best possible results with the appropriate learning model.

In the past few years, since it was originated, Federated Learning has consolidated itself as a promising approach for distributed, multi-device learning. Its framework presents a scalable architecture with very important advantages for nowadays necessities in the Machine Learning context, such as the user privacy protection. Nonetheless, FL is still evolving, and the quest to find an optimal algorithm for real-world problems remains unsolved. Early designed approaches of FL lack adaptation to the particularities of the users, as well as a constant and continual reevaluation.

Devices collect information constantly, on a daily basis, forming unbounded sets of data. For cases like this, data statistics are prone to vary unpredictably over time,

defying standard FL learning strategies. At the same time, each device has its own environment, its own capabilities, and its own context, so it can be hard to generalize the knowledge they all acquire. The main focus of this PhD thesis has been to fully understand and explain data heterogeneity and propose new learning strategies based on that understanding.

In the Introduction and Chapter 4 we set the basic information gathered prior to the beginning of this research, and all the important terms and concepts that comply the starting point for this investigation to commence.

In Chapter 5, an extensive review of all technical prior work is done and organized, setting a classification of non-IID data based on statistical information, and an empirical confirmation of some useful data restrictions is done, providing a guideline of essential requirements needed to confront data heterogeneity in real-life tasks.

In Chapter 6, the problem of data heterogeneity over time is formalized and faced, providing a novel algorithm of Continual Federated Learning: CDA-FedAvg. This approach presents an extension of standard FedAvg algorithm to the continual paradigm. It maintains the main advantages of a FL algorithm, like data privacy of the participants in the training process, and at the same time is able of identifying virtual concept drifts and react to them, increasing the overall accuracy of the resultant learning model. This approach enables FL over an unlimited period of time without decreasing its quality and performance, understanding what should be learned, and when. The final section of this chapter is dedicated to a thorough evaluation of CDA-FedAvg, comparing it with its predecessor, the FedAvg algorithm, and showing the benefits of being aware of data concept drifts. Experiments are carried out on a Human Activity Recognition task.

In Chapter 7, the focus is shifted to the data heterogeneity across the participants of FL model training. In this context, it is important to understand the singularity of each client, and how they can be potentially helped or harmed by the data collected by each other. To provide a fair, efficient and rigorous metric of their discrepancy, a novel error function is defined and employed. This new metric, named Heterogeneous and Multidimensional Cross-Entropy Loss, gathers the errors obtained by clients in each of the possible classes of the classification task that is being handled. In addition, it increases the error on samples that are not labeled correctly, adding a penalization term in the error function that holds invariant the error on samples correctly classified. Finally, it considers the median error in each of the classes, which makes it robust to outlier data samples. With all these ingredients, the Heterogeneous and Multidimensional Federated Averaging is presented. It uses this error metric to split participants into clusters that are guaranteed to benefit from collaboration, and

at the same time prevents undesired results from clients with large differences in their datasets. Furthermore, this novel metric allows for multiple ways of designing the resultant algorithm, depending on which distance function is employed to measure the discrepancies. As in the previous chapter, the final section is dedicated to evaluating this algorithm, comparing it both with FedAvg and another clustering technique, Clustered Federated Learning. In all of the scenarios presented, the H&M-FedAvg obtains either similar or better results than the other 2, with the advantage of understanding the clusters formed, and the velocity for identifying them.

At this point, it can be solidly affirmed that both H&M-FedAvg and CDA-FedAvg are valuable learning strategies that outperform previous state-of-the-art algorithms in context of Spatial and Temporal heterogeneity, respectively. As always, the choice of which algorithm to be employed for a particular task depends on the task itself, and the environmental conditions it carries.

With this dissertation, the objectives defined in the Introduction have been fulfilled, and this PhD thesis entails a step forward in data heterogeneity understanding, and in Federated Learning research. Two alternative algorithms have been proposed, for tackling either spatial or temporal non-IID data, and both approaches can be combined into a single learning algorithm for specific tasks that require so.

Bibliography

[1] M. F. Criado, F. E. Casado, R. Iglesias, C. V. Regueiro, S. Barro, Non-iid data and continual learning processes in federated learning: A long road ahead, Information Fusion 88 (2022) 263–280.

[2] F. E. Casado, D. Lema, M. F. Criado, R. Iglesias, C. V. Regueiro, S. Barro, Concept drift detection and adaptation for federated and continual learning, Multimedia Tools and Applications (2022) 1–23.

[3] L. Atzori, A. Iera, G. Morabito, The internet of things: A survey, Computer networks 54 (15) (2010) 2787–2805.

[4] S. Li, L. Da Xu, S. Zhao, 5g internet of things: A survey, Journal of Industrial Information Integration 10 (2018) 1–9.

[5] P. P. Shinde, S. Shah, A review of machine learning and deep learning applications, in: 2018 Fourth international conference on computing communication control and automation (ICCUBEA), IEEE, 2018, pp. 1–6.

[6] S. Li, L. D. Xu, S. Zhao, The internet of things: a survey, Information systems frontiers 17 (2015) 243–259.

[7] B. Custers, A. M. Sears, F. Dechesne, I. Georgieva, T. Tani, S. Van der Hof, EU personal data protection in policy and practice, Vol. 29, Springer, 2019.

[8] B. M. Gaff, H. E. Sussman, J. Geetter, Privacy and big data, Computer 47 (6) (2014) 7–9.

[9] P. Gupta, A. Sharma, R. Jindal, Scalable machine-learning algorithms for big data analytics: a comprehensive review, Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery 6 (6) (2016) 194–214.

[10] R. T. Potla, Scalable machine learning algorithms for big data analytics: Challenges and opportunities, Journal of Artificial Intelligence Research 2 (2) (2022) 124–141.

[11] B. McMahan, E. Moore, D. Ramage, S. Hampson, B. A. y Arcas, Communication-efficient learning of deep networks from decentralized data, in: Artificial Intelligence and Statistics, PMLR, 2017, pp. 1273–1282.

[12] J. Konečnỳ, H. B. McMahan, D. Ramage, P. Richtárik, Federated optimization: Distributed machine learning for on-device intelligence, arXiv preprint arXiv:1610.02527 (2016).

[13] P. Kairouz, H. B. McMahan, B. Avent, A. Bellet, M. Bennis, A. N. Bhagoji, K. Bonawitz, Z. Charles, G. Cormode, R. Cummings, et al., Advances and open problems in federated learning, arXiv preprint arXiv:1912.04977 (2019).

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

[15] L. U. Khan, W. Saad, Z. Han, E. Hossain, C. S. Hong, Federated learning for internet of things: Recent advances, taxonomy, and open challenges, IEEE Communications Surveys & Tutorials 23 (3) (2021) 1759–1799.

[16] A. Rahman, M. S. Hossain, G. Muhammad, D. Kundu, T. Debnath, M. Rahman, M. S. I. Khan, P. Tiwari, S. S. Band, Federated learning-based ai approaches in smart healthcare: concepts, taxonomies, challenges and open issues, Cluster computing 26 (4) (2023) 2271–2311.

[17] H. B. McMahan, E. Moore, D. Ramage, B. Aguera-Arcas, Federated learning of deep networks using model averaging, arXiv preprint arXiv:1602.05629v1 (2016).

[18] Y. Deng, M. M. Kamani, M. Mahdavi, Adaptive personalized federated learning, arXiv preprint arXiv:2003.13461 (2020).

[19] A. L. Samuel, Some studies in machine learning using the game of checkers. ii—recent progress, IBM Journal of research and development 11 (6) (1967) 601–617.

[20] F. Rosenblatt, The perceptron: a probabilistic model for information storage and organization in the brain., Psychological review 65 (6) (1958) 386.

[21] P. Werbos, Beyond regression: New tools for prediction and analysis in the behavioral sciences, PhD thesis, Committee on Applied Mathematics, Harvard University, Cambridge, MA (1974).

[22] T. Cover, P. Hart, Nearest neighbor pattern classification, IEEE transactions on information theory 13 (1) (1967) 21–27.

[23] J. R. Quinlan, Induction of decision trees, Machine learning 1 (1986) 81–106.

[24] C. Cortes, V. Vapnik, Support-vector networks, Machine learning 20 (1995) 273–297.

[25] D. E. Rumelhart, G. E. Hinton, R. J. Williams, Learning representations by back-propagating errors, nature 323 (6088) (1986) 533–536.

[26] G. E. Hinton, Learning multiple layers of representation, Trends in cognitive sciences 11 (10) (2007) 428–434.

[27] S. Dong, P. Wang, K. Abbas, A survey on deep learning and its applications, Computer Science Review 40 (2021) 100379.

[28] A. Kamilaris, F. X. Prenafeta-Boldú, Deep learning in agriculture: A survey, Computers and electronics in agriculture 147 (2018) 70–90.

[29] J. H. Lee, J. Shin, M. J. Realff, Machine learning: Overview of the recent progresses and implications for the process systems engineering field, Computers & Chemical Engineering 114 (2018) 111–121.

[30] M. Cord, P. Cunningham, Machine learning techniques for multimedia: case studies on organization and retrieval, Springer Science & Business Media, 2008.

[31] Y. J. Bae, M. Shim, W. H. Lee, Schizophrenia detection using machine learning approach from social media content, Sensors 21 (17) (2021) 5924.

[32] L. P. Kaelbling, M. L. Littman, A. W. Moore, Reinforcement learning: A survey, Journal of artificial intelligence research 4 (1996) 237–285.

[33] X. Wang, S. Wang, X. Liang, D. Zhao, J. Huang, X. Xu, B. Dai, Q. Miao, Deep reinforcement learning: A survey, IEEE Transactions on Neural Networks and Learning Systems 35 (4) (2022) 5064–5078.

[34] A. K. Shakya, G. Pillai, S. Chakrabarty, Reinforcement learning algorithms: A brief survey, Expert Systems with Applications 231 (2023) 120495.

[35] D. Dewey, Reinforcement learning and the reward engineering principle., in: AAAI spring symposia, 2014, pp. 13–16.

[36] J. Eschmann, Reward function design in reinforcement learning, Reinforcement learning algorithms: Analysis and Applications (2021) 25–33.

[37] A. Singh, N. Thakur, A. Sharma, A review of supervised machine learning algorithms, in: 2016 3rd international conference on computing for sustainable global development (INDIACom), Ieee, 2016, pp. 1310–1315.

[38] I. Muhammad, Z. Yan, Supervised machine learning approaches: A survey., ICTACT Journal on Soft Computing 5 (3) (2015).

[39] R. Saravanan, P. Sujatha, A state of art techniques on machine learning algorithms: a perspective of supervised learning approaches in data classification, in: 2018 Second international conference on intelligent computing and control systems (ICICCS), IEEE, 2018, pp. 945–949.

[40] K. Sudhaman, M. Akuthota, S. K. Chaurasiya, A review on the different regression analysis in supervised learning, Bayesian Reasoning and Gaussian Processes for Machine Learning Applications (2022) 15–32.

[41] J. E. Van Engelen, H. H. Hoos, A survey on semi-supervised learning, Machine learning 109 (2) (2020) 373–440.

[42] Y. Ouali, C. Hudelot, M. Tami, An overview of deep semi-supervised learning, arXiv preprint arXiv:2006.05278 (2020).

[43] M. Usama, J. Qadir, A. Raza, H. Arif, K.-L. A. Yau, Y. Elkhatib, A. Hussain, A. Al-Fuqaha, Unsupervised machine learning for networking: Techniques, applications and research challenges, IEEE access 7 (2019) 65579–65615.

[44] S. Naeem, A. Ali, S. Anam, M. M. Ahmed, An unsupervised machine learning algorithms: Comprehensive review, International Journal of Computing and Digital Systems (2023).

[45] C. E. Shannon, A mathematical theory of communication, The Bell system technical journal 27 (3) (1948) 379–423.

[46] C. Adami, Information theory in molecular biology, Physics of Life Reviews 1 (1) (2004) 3–22.

[47] A. Ben-Naim, Applications of information theory to psychology, Information Theory: An Exploration Across Disciplines (2024) 59–82.

[48] R. F. Wagner, D. G. Brown, M. S. Pastel, Application of information theory to the assessment of computed tomography, Medical physics 6 (2) (1979) 83–94.

[49] R. M. Gray, Entropy and information theory, Springer Science & Business Media, 2011.

[50] T. Van Erven, P. Harremos, Rényi divergence and kullback-leibler divergence, IEEE Transactions on Information Theory 60 (7) (2014) 3797–3820.

[51] A. Mao, M. Mohri, Y. Zhong, Cross-entropy loss functions: Theoretical analysis and applications, in: International conference on Machine learning, PMLR, 2023, pp. 23803–23828.

[52] D. Peteiro-Barral, B. Guijarro-Berdiñas, A survey of methods for distributed machine learning, Progress in Artificial Intelligence 2 (2013) 1–11.

[53] J. Verbraeken, M. Wolting, J. Katzy, J. Kloppenburg, T. Verbelen, J. S. Rellermeyer, A survey on distributed machine learning, Acm computing surveys (csur) 53 (2) (2020) 1–33.

[54] T. White, Hadoop: The definitive guide, " O'Reilly Media, Inc.", 2012.

[55] X. Meng, J. Bradley, B. Yavuz, E. Sparks, S. Venkataraman, D. Liu, J. Freeman, D. Tsai, M. Amde, S. Owen, et al., Mllib: Machine learning in apache spark, Journal of Machine Learning Research 17 (34) (2016) 1–7.

[56] A. Berson, Client/server architecture, McGraw-Hill, Inc., 1992.

[57] S. Kumar, A review on client-server based applications and research opportunity, International Journal of Recent Scientific Research 10 (7) (2019) 33857–3386.

[58] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, et al., {TensorFlow}: a system for {Large-Scale} machine learning, in: 12th USENIX symposium on operating systems design and implementation (OSDI 16), 2016, pp. 265–283.

[59] B. Pang, E. Nijkamp, Y. N. Wu, Deep learning with tensorflow: A review, Journal of Educational and Behavioral Statistics 45 (2) (2020) 227–248.

[60] S. Androutsellis-Theotokis, D. Spinellis, A survey of peer-to-peer content distribution technologies, ACM computing surveys (CSUR) 36 (4) (2004) 335–371.

[61] R. Schollmeier, A definition of peer-to-peer networking for the classification of peer-to-peer architectures and applications, in: Proceedings first international conference on peer-to-peer computing, IEEE, 2001, pp. 101–102.

[62] H. Robbins, S. Monro, A stochastic approximation method, The annals of mathematical statistics (1951) 400–407.

[63] F. Seide, H. Fu, J. Droppo, G. Li, D. Yu, 1-bit stochastic gradient descent and its application to data-parallel distributed training of speech dnns., in: Interspeech, Vol. 2014, Singapore, 2014, pp. 1058–1062.

[64] A. Agarwal, M. J. Wainwright, J. C. Duchi, Distributed dual averaging in networks, Advances in Neural Information Processing Systems 23 (2010).

[65] M. Zinkevich, J. Langford, A. Smola, Slow learners are fast, Advances in neural information processing systems 22 (2009).

[66] M. Zinkevich, M. Weimer, L. Li, A. Smola, Parallelized stochastic gradient descent, Advances in neural information processing systems 23 (2010).

[67] A. Triastcyn, B. Faltings, Federated learning with bayesian differential privacy, in: 2019 IEEE International Conference on Big Data (Big Data), IEEE, 2019, pp. 2587–2596.

[68] J. Liu, J. Lou, L. Xiong, J. Liu, X. Meng, Projected federated averaging with heterogeneous differential privacy, Proceedings of the VLDB Endowment 15 (4) (2021) 828–840.

[69] C. Zhang, S. Li, J. Xia, W. Wang, F. Yan, Y. Liu, {BatchCrypt}: Efficient homomorphic encryption for {Cross-Silo} federated learning, in: 2020 USENIX annual technical conference (USENIX ATC 20), 2020, pp. 493–506.

[70] J. Ma, S.-A. Naas, S. Sigg, X. Lyu, Privacy-preserving federated learning based on multi-key homomorphic encryption, International Journal of Intelligent Systems 37 (9) (2022) 5880–5901.

[71] J. W. Bos, K. Lauter, M. Naehrig, Private predictive analysis on encrypted medical data, Journal of biomedical informatics 50 (2014) 234–243.

[72] J. Geiping, H. Bauermeister, H. Dröge, M. Moeller, Inverting gradients-how easy is it to break privacy in federated learning?, Advances in neural information processing systems 33 (2020) 16937–16947.

[73] D. I. Dimitrov, M. Balunovic, N. Konstantinov, M. Vechev, Data leakage in federated averaging, Transactions on Machine Learning Research (2022).

[74] T. Li, A. K. Sahu, M. Zaheer, M. Sanjabi, A. Talwalkar, V. Smith, Federated optimization in heterogeneous networks, Proceedings of Machine Learning and Systems 2 (2020) 429–450.

[75] W. Y. B. Lim, N. C. Luong, D. T. Hoang, Y. Jiao, Y.-C. Liang, Q. Yang, D. Niyato, C. Miao, Federated learning in mobile edge networks: A comprehensive survey, IEEE Communications Surveys & Tutorials 22 (3) (2020) 2031–2063.

[76] Y. Zhao, M. Li, L. Lai, N. Suda, D. Civin, V. Chandra, Federated learning with non-iid data, arXiv preprint arXiv:1806.00582 (2018).

[77] T. Lesort, V. Lomonaco, A. Stoian, D. Maltoni, D. Filliat, N. Díaz-Rodríguez, Continual learning for robotics: Definition, framework, learning strategies, opportunities and challenges, Information fusion 58 (2020) 52–68.

[78] M. De Lange, R. Aljundi, M. Masana, S. Parisot, X. Jia, A. Leonardis, G. Slabaugh, T. Tuytelaars, A continual learning survey: Defying forgetting in classification tasks, IEEE transactions on pattern analysis and machine intelligence 44 (7) (2021) 3366–3385.

[79] L. Wang, X. Zhang, H. Su, J. Zhu, A comprehensive survey of continual learning: Theory, method and application, IEEE Transactions on Pattern Analysis and Machine Intelligence (2024).

[80] M. McCloskey, N. J. Cohen, Catastrophic interference in connectionist networks: The sequential learning problem, in: Psychology of learning and motivation, Vol. 24, Elsevier, 1989, pp. 109–165.

[81] R. Ratcliff, Connectionist models of recognition memory: Constraints imposed by learning and forgetting functions, Psychological Review 97 (2) (1990) 285–308.

[82] G. M. Van de Ven, T. Tuytelaars, A. S. Tolias, Three types of incremental learning, Nature Machine Intelligence 4 (12) (2022) 1185–1197.

[83] D.-W. Zhou, Q.-W. Wang, Z.-H. Qi, H.-J. Ye, D.-C. Zhan, Z. Liu, Class-incremental learning: A survey, IEEE Transactions on Pattern Analysis and Machine Intelligence (2024).

[84] M. Laal, Lifelong learning: What does it mean?, Procedia-Social and Behavioral Sciences 28 (2011) 470–474.

[85] F. G. Febrinanto, F. Xia, K. Moore, C. Thapa, C. Aggarwal, Graph lifelong learning: A survey, IEEE Computational Intelligence Magazine 18 (1) (2023) 32–51.

[86] T. Mitchell, E. Fredkin, Never-ending language learning, in: 2014 IEEE International conference on big data (Big Data), IEEE, 2014, pp. 1–1.

[87] T. Mitchell, W. Cohen, E. Hruschka, P. Talukdar, B. Yang, J. Betteridge, A. Carlson, B. Dalvi, M. Gardner, B. Kisiel, et al., Never-ending learning, Communications of the ACM 61 (5) (2018) 103–115.

[88] R. M. French, Catastrophic forgetting in connectionist networks, Trends in cognitive sciences 3 (4) (1999) 128–135.

[89] J. Serra, D. Suris, M. Miron, A. Karatzoglou, Overcoming catastrophic forgetting with hard attention to the task, in: International conference on machine learning, PMLR, 2018, pp. 4548–4557.

[90] X. Li, Y. Zhou, T. Wu, R. Socher, C. Xiong, Learn to grow: A continual structure learning framework for overcoming catastrophic forgetting, in: International conference on machine learning, PMLR, 2019, pp. 3925–3934.

[91] J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska, et al., Overcoming catastrophic forgetting in neural networks, Proceedings of the National Academy of Sciences 114 (13) (2017) 3521–3526.

[92] H. Shin, J. K. Lee, J. Kim, J. Kim, Continual learning with deep generative replay, Advances in neural information processing systems 30 (2017).

[93] H. Wang, Y. Liu, X. Wang, H. Zhang, D. Lin, Wandering within a world: Online contextualized continual learning in histopathological image classification, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2021, pp. 5741–5750.

[94] C. S. Lee, A. Y. Lee, Clinical applications of continual learning machine learning, The Lancet Digital Health 2 (6) (2020) e279–e281.

[95] C. Devin, A. Gupta, T. Darrell, P. Abbeel, S. Levine, Learning modular neural network policies for multi-task and multi-robot transfer, in: 2017 IEEE International Conference on Robotics and Automation (ICRA), IEEE, 2017, pp. 2169–2176.

[96] R. Aljundi, K. Kelchtermans, T. Tuytelaars, Task-free continual learning, in: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, 2019, pp. 11254–11263.

[97] S.-A. Rebuffi, A. Kolesnikov, G. Sperl, C. H. Lampert, icarl: Incremental classifier and representation learning, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017, pp. 2001–2010.

[98] M. Büchel, M. Rottmann, B. Sick, Continual learning in sensor fusion for autonomous driving, Information Fusion 83 (2022) 56–68.

[99] S. Ramírez-Gallego, B. Krawczyk, S. García, M. Woźniak, F. Herrera, A survey on data preprocessing for data stream mining: Current status and future directions, Neurocomputing 239 (2017) 39–57.

[100] G. I. Webb, R. Hyde, H. Cao, H. L. Nguyen, F. Petitjean, Characterizing concept drift, Data Mining and Knowledge Discovery 30 (4) (2016) 964–994.

[101] J. Gama, I. Žliobaitė, A. Bifet, M. Pechenizkiy, A. Bouchachia, A survey on concept drift adaptation, ACM computing surveys (CSUR) 46 (4) (2014) 1–37.

[102] J. Lu, A. Liu, F. Dong, F. Gu, J. Gama, G. Zhang, Learning under concept drift: A review, IEEE transactions on knowledge and data engineering 31 (12) (2018) 2346–2363.

[103] G. I. Webb, L. K. Lee, F. Petitjean, B. Goethals, Understanding concept drift, arXiv preprint arXiv:1704.00362 (2017).

[104] J. C. Schlimmer, R. H. Granger, Incremental learning from noisy data, Machine learning 1 (1986) 317–354.

[105] I. Khamassi, M. Sayed-Mouchaweh, M. Hammami, K. Ghédira, Discussion and review on evolving data streams and concept drift adapting, Evolving systems 9 (2018) 1–23.

[106] A. Tsymbal, The problem of concept drift: definitions and related work, Computer Science Department, Trinity College Dublin 106 (2) (2004) 58.

[107] A. S. Iwashita, J. P. Papa, An overview on concept drift learning, IEEE access 7 (2018) 1532–1547.

[108] S. Li, Y. Cheng, W. Wang, Y. Liu, T. Chen, Learning to detect malicious clients for robust federated learning, arXiv preprint arXiv:2002.00211 (2020).

[109] Z. Zhang, X. Cao, J. Jia, N. Z. Gong, Fldetector: Defending federated learning against model poisoning attacks via detecting malicious clients, in: Proceedings of the 28th ACM SIGKDD conference on knowledge discovery and data mining, 2022, pp. 2545–2555.

[110] C. Xu, Y. Qu, Y. Xiang, L. Gao, Asynchronous federated learning on heterogeneous devices: A survey, Computer Science Review 50 (2023) 100595.

[111] Y. Chen, Y. Ning, M. Slawski, H. Rangwala, Asynchronous online federated learning for edge devices with non-iid data, in: 2020 IEEE International Conference on Big Data (Big Data), IEEE, 2020, pp. 15–24.

[112] X. Li, Z. Qu, B. Tang, Z. Lu, Stragglers are not disaster: A hybrid federated learning algorithm with delayed gradients, arXiv preprint arXiv:2102.06329 (2021).

[113] L. Qu, Y. Zhou, P. P. Liang, Y. Xia, F. Wang, E. Adeli, L. Fei-Fei, D. Rubin, Rethinking architecture design for tackling data heterogeneity in federated

learning, in: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, 2022, pp. 10061–10071.

[114] A. M. Abdelmoniem, C.-Y. Ho, P. Papageorgiou, M. Canini, A comprehensive empirical study of heterogeneity in federated learning, IEEE Internet of Things Journal 10 (16) (2023) 14071–14083.

[115] Q. Li, Y. Diao, Q. Chen, B. He, Federated learning on non-iid data silos: An experimental study, arXiv preprint arXiv:2102.02079 (2021).

[116] F. Sattler, S. Wiedemann, K.-R. Müller, W. Samek, Robust and communication-efficient federated learning from non-iid data, IEEE transactions on neural networks and learning systems 31 (9) (2019) 3400–3413.

[117] D. Li, J. Wang, Fedmd: Heterogenous federated learning via model distillation, arXiv preprint arXiv:1910.03581 (2019).

[118] K. Wang, R. Mathews, C. Kiddon, H. Eichner, F. Beaufays, D. Ramage, Federated evaluation of on-device personalization, arXiv preprint arXiv:1910.10252 (2019).

[119] P. P. Liang, T. Liu, L. Ziyin, N. B. Allen, R. P. Auerbach, D. Brent, R. Salakhutdinov, L.-P. Morency, Think locally, act globally: Federated learning with local and global representations, arXiv preprint arXiv:2001.01523 (2020).

[120] C. T Dinh, N. Tran, J. Nguyen, Personalized federated learning with moreau envelopes, Advances in Neural Information Processing Systems 33 (2020) 21394–21405.

[121] Z. Ma, Y. Lu, W. Li, J. Yi, S. Cui, Pfedatt: Attention-based personalized federated learning on heterogeneous clients, in: Asian Conference on Machine Learning, PMLR, 2021, pp. 1253–1268.

[122] C. Finn, P. Abbeel, S. Levine, Model-agnostic meta-learning for fast adaptation of deep networks, in: International Conference on Machine Learning, PMLR, 2017, pp. 1126–1135.

[123] A. Fallah, A. Mokhtari, A. Ozdaglar, Personalized federated learning: A meta-learning approach, arXiv preprint arXiv:2002.07948 (2020).

[124] T. Yu, T. Li, Y. Sun, S. Nanda, V. Smith, V. Sekar, S. Seshan, Learning context-aware policies from multiple smart homes via federated multi-task learning, in: 2020 IEEE/ACM Fifth International Conference on Internet-of-Things Design and Implementation (IoTDI), IEEE, 2020, pp. 104–115.

[125] V. Smith, C.-K. Chiang, M. Sanjabi, A. Talwalkar, Federated multi-task learning, arXiv preprint arXiv:1705.10467 (2017).

[126] M. G. Arivazhagan, V. Aggarwal, A. K. Singh, S. Choudhary, Federated learning with personalization layers, arXiv preprint arXiv:1912.00818 (2019).

[127] H. Nam, B. Han, Learning multi-domain convolutional neural networks for visual tracking, in: Proceedings of the IEEE conference on computer vision and pattern recognition, 2016, pp. 4293–4302.

[128] F. Hanzely, P. Richtárik, Federated learning of a mixture of global and local models, arXiv preprint arXiv:2002.05516 (2020).

[129] T. Li, A. K. Sahu, M. Zaheer, M. Sanjabi, A. Talwalkar, V. Smith, Federated optimization in heterogeneous networks, arXiv preprint arXiv:1812.06127 (2018).

[130] N. Tajbakhsh, J. Y. Shin, S. R. Gurudu, R. T. Hurst, C. B. Kendall, M. B. Gotway, J. Liang, Convolutional neural networks for medical image analysis: Full training or fine tuning?, IEEE transactions on medical imaging 35 (5) (2016) 1299–1312.

[131] S. A. Dudani, The distance-weighted k-nearest-neighbor rule, IEEE Transactions on Systems, Man, and Cybernetics (1976) 325–327.

[132] J. Ren, X. Shen, Z. Lin, R. Mech, D. J. Foran, Personalized image aesthetics, in: Proceedings of the IEEE international conference on computer vision, 2017, pp. 638–647.

[133] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, Gradient-based learning applied to document recognition, Proceedings of the IEEE 86 (11) (1998) 2278–2324.

[134] G. Cohen, S. Afshar, J. Tapson, A. Van Schaik, Emnist: Extending mnist to handwritten letters, in: 2017 International Joint Conference on Neural Networks (IJCNN), IEEE, 2017, pp. 2921–2926.

[135] A. Krizhevsky, Learning multiple layers of features from tiny images, Master's thesis, Master's thesis, Department of Computer Science, University of Toronto (2009).

[136] F. Sattler, K.-R. Müller, W. Samek, Clustered federated learning: Model-agnostic distributed multitask optimization under privacy constraints, IEEE transactions on neural networks and learning systems 32 (8) (2020) 3710–3722.

[137] N. Shlezinger, S. Rini, Y. C. Eldar, The communication-aware clustered federated learning problem, in: 2020 IEEE International Symposium on Information Theory (ISIT), IEEE, 2020, pp. 2610–2615.

[138] C. Briggs, Z. Fan, P. Andras, Federated learning with hierarchical clustering of local updates to improve training on non-iid data, in: 2020 International Joint Conference on Neural Networks (IJCNN), IEEE, 2020, pp. 1–9.

[139] J. Hoffman, M. Mohri, N. Zhang, Algorithms and theory for multiple-source adaptation, arXiv preprint arXiv:1805.08727 (2018).

[140] M. Mohri, G. Sivek, A. T. Suresh, Agnostic federated learning, in: International Conference on Machine Learning, PMLR, 2019, pp. 4615–4625.

[141] Y. Mansour, M. Mohri, J. Ro, A. T. Suresh, Three approaches for personalization with applications to federated learning, arXiv preprint arXiv:2002.10619 (2020).

[142] H. Xiao, K. Rasul, R. Vollgraf, Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, arXiv preprint arXiv:1708.07747 (2017).

[143] B. Lake, R. Salakhutdinov, J. Gross, J. Tenenbaum, One shot learning of simple visual concepts, in: Proceedings of the annual meeting of the cognitive science society, Vol. 33, 2011, pp. 2568–2573.

[144] Y. Wu, J. Lim, M.-H. Yang, Online object tracking: A benchmark, in: Proceedings of the IEEE conference on computer vision and pattern recognition, 2013, pp. 2411–2418.

[145] S. Hadfield, K. Lebeda, R. Bowden, The visual object tracking vot2014 challenge results, in: European Conference on Computer Vision (ECCV) Visual Object Tracking Challenge Workshop, University of Surrey, 2014, pp. 15–23.

[146] H. Zhao, R. T. Des Combes, K. Zhang, G. Gordon, On learning invariant representations for domain adaptation, in: International Conference on Machine Learning, PMLR, 2019, pp. 7523–7532.

[147] A.-A. Liu, N. Xu, W.-Z. Nie, Y.-T. Su, Y.-D. Zhang, Multi-domain and multi-task learning for human action recognition, IEEE Transactions on Image Processing 28 (2) (2018) 853–867.

[148] J. Hoffman, B. Kulis, T. Darrell, K. Saenko, Discovering latent domains for multisource domain adaptation, in: European Conference on Computer Vision, Springer, 2012, pp. 702–715.

[149] F. Siyahjani, R. Almohsen, S. Sabri, G. Doretto, A supervised low-rank method for learning invariant subspaces, in: Proceedings of the IEEE International Conference on Computer Vision, 2015, pp. 4220–4228.

[150] C. Zhang, H. Zhang, L. E. Parker, Feature space decomposition for effective robot adaptation, in: 2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), IEEE, 2015, pp. 441–448.

[151] M. Wang, B. Liu, J. Tang, X.-S. Hua, Metric learning with feature decomposition for image categorization, Neurocomputing 73 (10-12) (2010) 1562–1569.

[152] K. Q. Weinberger, L. K. Saul, Distance metric learning for large margin nearest neighbor classification., Journal of machine learning research 10 (2) (2009).

[153] H. Daumé III, Frustratingly easy domain adaptation, arXiv preprint arXiv:0907.1815 (2009).

[154] Y. Ganin, V. Lempitsky, Unsupervised domain adaptation by backpropagation, in: International conference on machine learning, PMLR, 2015, pp. 1180–1189.

[155] K. You, M. Long, Z. Cao, J. Wang, M. I. Jordan, Universal domain adaptation, in: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, 2019, pp. 2720–2729.

[156] M. Long, H. Zhu, J. Wang, M. I. Jordan, Deep transfer learning with joint adaptation networks, in: International conference on machine learning, PMLR, 2017, pp. 2208–2217.

[157] X. Peng, Q. Bai, X. Xia, Z. Huang, K. Saenko, B. Wang, Moment matching for multi-source domain adaptation, in: Proceedings of the IEEE/CVF International Conference on Computer Vision, 2019, pp. 1406–1415.

[158] P. O. Pinheiro, Unsupervised domain adaptation with similarity learning, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2018, pp. 8004–8013.

[159] M. Baktashmotlagh, M. T. Harandi, B. C. Lovell, M. Salzmann, Unsupervised domain adaptation by domain invariant projection, in: Proceedings of the IEEE International Conference on Computer Vision, 2013, pp. 769–776.

[160] M. Dredze, K. Crammer, Online methods for multi-domain learning and adaptation, in: Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing, 2008, pp. 689–697.

[161] M. Dredze, A. Kulesza, K. Crammer, Multi-domain learning by confidence-weighted parameter combination, Machine Learning 79 (1-2) (2010) 123–149.

[162] E. Tzeng, J. Hoffman, K. Saenko, T. Darrell, Adversarial discriminative domain adaptation, in: Proceedings of the IEEE conference on computer vision and pattern recognition, 2017, pp. 7167–7176.

[163] Z. Pei, Z. Cao, M. Long, J. Wang, Multi-adversarial domain adaptation, in: Thirty-second AAAI conference on artificial intelligence, 2018.

[164] Z. Cao, L. Ma, M. Long, J. Wang, Partial adversarial domain adaptation, in: Proceedings of the European Conference on Computer Vision (ECCV), 2018, pp. 135–150.

[165] S. Motiian, Q. Jones, S. M. Iranmanesh, G. Doretto, Few-shot adversarial domain adaptation, arXiv preprint arXiv:1711.02536 (2017).

[166] X. Peng, Z. Huang, Y. Zhu, K. Saenko, Federated adversarial domain adaptation, arXiv preprint arXiv:1911.02054 (2019).

[167] M. Köppen, The curse of dimensionality, in: 5th online world conference on soft computing in industrial applications (WSC5), Vol. 1, 2000, pp. 4–8.

[168] L. Yang, R. Jin, Distance metric learning: A comprehensive survey, Michigan State Universiy 2 (2) (2006) 4.

[169] E. Xing, M. Jordan, S. J. Russell, A. Ng, Distance metric learning with application to clustering with side-information, Advances in neural information processing systems 15 (2002) 521–528.

[170] Y. Chen, X. Qin, J. Wang, C. Yu, W. Gao, Fedhealth: A federated transfer learning framework for wearable healthcare, IEEE Intelligent Systems 35 (4) (2020) 83–93.

[171] M. Long, Y. Cao, J. Wang, M. Jordan, Learning transferable features with deep adaptation networks, in: International conference on machine learning, PMLR, 2015, pp. 97–105.

[172] E. Tzeng, J. Hoffman, N. Zhang, K. Saenko, T. Darrell, Deep domain confusion: Maximizing for domain invariance, arXiv preprint arXiv:1412.3474 (2014).

[173] M. Long, H. Zhu, J. Wang, M. I. Jordan, Unsupervised domain adaptation with residual transfer networks, arXiv preprint arXiv:1602.04433 (2016).

[174] K. Saenko, B. Kulis, M. Fritz, T. Darrell, Adapting visual category models to new domains, in: European conference on computer vision, Springer, 2010, pp. 213–226.

[175] H. Venkateswara, J. Eusebio, S. Chakraborty, S. Panchanathan, Deep hashing network for unsupervised domain adaptation, in: Proceedings of the IEEE conference on computer vision and pattern recognition, 2017, pp. 5018–5027.

[176] X. Peng, B. Usman, N. Kaushik, J. Hoffman, D. Wang, K. Saenko, Visda: The visual domain adaptation challenge, arXiv preprint arXiv:1710.06924 (2017).

[177] T. Hastie, R. Tibshirani, J. H. Friedman, J. H. Friedman, The elements of statistical learning: data mining, inference, and prediction, Vol. 2, Springer, 2009.

[178] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, A. Y. Ng, Reading digits in natural images with unsupervised feature learning (2011).

[179] A. Bergamo, L. Torresani, Exploiting weakly-labeled web images to improve object classification: a domain adaptation approach, Advances in neural information processing systems 23 (2010) 181–189.

[180] Y. Chen, J. Bi, J. Z. Wang, Miles: Multiple-instance learning via embedded instance selection, IEEE Transactions on Pattern Analysis and Machine Intelligence 28 (12) (2006) 1931–1947.

[181] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, et al., Imagenet large scale visual recognition challenge, International journal of computer vision 115 (3) (2015) 211–252.

[182] N. Silberman, D. Hoiem, P. Kohli, R. Fergus, Indoor segmentation and support inference from rgbd images, in: European conference on computer vision, Springer, 2012, pp. 746–760.

[183] R. Gopalan, R. Li, R. Chellappa, Domain adaptation for object recognition: An unsupervised approach, in: 2011 international conference on computer vision, IEEE, 2011, pp. 999–1006.

[184] X.-D. Zhang, Machine learning, in: A Matrix Algebra Approach to Artificial Intelligence, Springer, 2020, pp. 223–440.

[185] T. Hiessl, S. Rezapour Lakani, J. Kemnitz, D. Schall, S. Schulte, Cohort-based federated learning service for industrial collaboration on the edge, Journal of Parallel and Distributed Computing 167 (2021) 64–76.

[186] Y. Yang, T. M. Hospedales, A unified perspective on multi-domain and multi-task learning, arXiv preprint arXiv:1412.7489 (2014).

[187] L. Corinzia, A. Beuret, J. M. Buhmann, Variational federated multi-task learning, arXiv preprint arXiv:1906.06268 (2019).

[188] S. Thrun, T. M. Mitchell, Lifelong robot learning, Robotics and autonomous systems 15 (1-2) (1995) 25–46.

[189] C. Tessler, S. Givony, T. Zahavy, D. Mankowitz, S. Mannor, A deep hierarchical approach to lifelong learning in minecraft, in: Proceedings of the AAAI Conference on Artificial Intelligence (AAAI-17), Vol. 31, 2017, pp. 1553–1561.

[190] A. Carlson, J. Betteridge, B. Kisiel, B. Settles, E. R. Hruschka, T. M. Mitchell, Toward an architecture for never-ending language learning, in: Twenty-Fourth AAAI conference on artificial intelligence, 2010.

[191] T. Xiao, J. Zhang, K. Yang, Y. Peng, Z. Zhang, Error-driven incremental learning in deep convolutional neural network for large-scale image classification, in: Proceedings of the 22nd ACM international conference on Multimedia, 2014, pp. 177–186.

[192] A. Gepperth, B. Hammer, Incremental learning algorithms and applications, in: European symposium on artificial neural networks (ESANN), 2016, pp. 357–368.

[193] A. Usmanova, F. Portet, P. Lalanda, G. Vega, A distillation-based approach integrating continual learning and federated learning for pervasive services, arXiv preprint arXiv:2109.04197 (2021).

[194] T. J. Park, K. Kumatani, D. Dimitriadis, Tackling dynamics in federated incremental learning with variational embedding rehearsal, arXiv preprint arXiv:2110.09695 (2021).

[195] J. Yoon, W. Jeong, G. Lee, E. Yang, S. J. Hwang, Federated continual learning with weighted inter-client transfer, in: International Conference on Machine Learning, PMLR, 2021, pp. 12073–12086.

[196] R. Kemker, M. McClure, A. Abitino, T. Hayes, C. Kanan, Measuring catastrophic forgetting in neural networks, in: Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 32, 2018, pp. 3390–3398.

[197] I. J. Goodfellow, M. Mirza, D. Xiao, A. Courville, Y. Bengio, An empirical investigation of catastrophic forgetting in gradient-based neural networks, arXiv preprint arXiv:1312.6211 (2013).

[198] M. S. Hammoodi, F. Stahl, A. Badii, Real-time feature selection technique with concept drift detection using adaptive micro-clusters for data stream mining, Knowledge-Based Systems 161 (2018) 205–239.

[199] A. Dries, U. Rückert, Adaptive concept drift detection, Statistical Analysis and Data Mining: The ASA Data Science Journal 2 (5-6) (2009) 311–327.

[200] J. Shao, Z. Ahmadi, S. Kramer, Prototype-based learning on concept-drifting data streams, in: Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining, 2014, pp. 412–421.

[201] J. Gama, P. Medas, G. Castillo, P. Rodrigues, Learning with drift detection, in: Brazilian symposium on artificial intelligence, Springer, 2004, pp. 286–295.

[202] M. Baena-Garcıa, J. del Campo-Ávila, R. Fidalgo, A. Bifet, R. Gavalda, R. Morales-Bueno, Early drift detection method, in: Fourth international workshop on knowledge discovery from data streams, Vol. 6, 2006, pp. 77–86.

[203] D. M. Manias, I. Shaer, L. Yang, A. Shami, Concept drift detection in federated networked systems, arXiv preprint arXiv:2109.06088 (2021).

[204] A. Bifet, R. Gavalda, Adaptive learning from evolving data streams, in: International Symposium on Intelligent Data Analysis, Springer, 2009, pp. 249–260.

[205] A. Bifet, G. Holmes, B. Pfahringer, R. Gavalda, Improving adaptive bagging methods for evolving data streams, in: Asian conference on machine learning, Springer, 2009, pp. 23–37.

[206] I. Frias-Blanco, J. del Campo-Ávila, G. Ramos-Jimenez, R. Morales-Bueno, A. Ortiz-Diaz, Y. Caballero-Mota, Online and non-parametric drift detection methods based on hoeffding's bounds, IEEE Transactions on Knowledge and Data Engineering 27 (3) (2014) 810–823.

[207] K. Nar, O. Ocal, S. S. Sastry, K. Ramchandran, Cross-entropy loss and low-rank features have responsibility for adversarial examples, arXiv preprint arXiv:1901.08360 (2019).

[208] L. Feng, S. Shu, Z. Lin, F. Lv, L. Li, B. An, Can cross entropy loss be robust to label noise?, in: IJCAI, 2020, pp. 2206–2212.

[209] Y. Ho, S. Wookey, The real-world-weight cross-entropy loss function: Modeling the costs of mislabeling, IEEE Access 8 (2019) 4806–4813.

[210] Z. Zhang, M. R. Sabuncu, Generalized cross entropy loss for training deep neural networks with noisy labels, in: 32nd Conference on Neural Information Processing Systems (NeurIPS), 2018.

[211] X. Li, L. Yu, D. Chang, Z. Ma, J. Cao, Dual cross-entropy loss for small-sample fine-grained vehicle classification, IEEE Transactions on Vehicular Technology 68 (5) (2019) 4204–4212.

[212] D. Rolnick, A. Ahuja, J. Schwarz, T. P. Lillicrap, G. Wayne, Experience replay for continual learning, arXiv preprint arXiv:1811.11682 (2018).

[213] A. Chaudhry, M. Rohrbach, M. Elhoseiny, T. Ajanthan, P. K. Dokania, P. H. Torr, M. Ranzato, Continual learning with tiny episodic memories (2019).

[214] G. M. Van de Ven, A. S. Tolias, Generative replay with feedback connections as a general strategy for continual learning, arXiv preprint arXiv:1809.10635 (2018).

[215] J. Schwarz, W. Czarnecki, J. Luketina, A. Grabska-Barwinska, Y. W. Teh, R. Pascanu, R. Hadsell, Progress & compress: A scalable framework for continual learning, in: International Conference on Machine Learning, PMLR, 2018, pp. 4528–4537.

[216] H. Ritter, A. Botev, D. Barber, Online structured laplace approximations for overcoming catastrophic forgetting, arXiv preprint arXiv:1805.07810 (2018).

[217] J. Yoon, E. Yang, J. Lee, S. J. Hwang, Lifelong learning with dynamically expandable networks, arXiv preprint arXiv:1708.01547 (2017).

[218] X. He, J. Sygnowski, A. Galashov, A. A. Rusu, Y. W. Teh, R. Pascanu, Task agnostic continual learning via meta learning, arXiv preprint arXiv:1906.05201 (2019).

[219] A. Mallya, D. Davis, S. Lazebnik, Piggyback: Adapting a single network to multiple tasks by learning to mask weights, in: Proceedings of the European Conference on Computer Vision (ECCV), 2018, pp. 67–82.

[220] A. Mallya, S. Lazebnik, Packnet: Adding multiple tasks to a single network by iterative pruning, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2018, pp. 7765–7773.

[221] M. Masana, T. Tuytelaars, J. van de Weijer, Ternary feature masks: zero-forgetting for task-incremental learning, arXiv preprint arXiv:2001.08714 (2020).

[222] A. A. Rusu, N. C. Rabinowitz, G. Desjardins, H. Soyer, J. Kirkpatrick, K. Kavukcuoglu, R. Pascanu, R. Hadsell, Progressive neural networks, arXiv preprint arXiv:1606.04671 (2016).

[223] M. G. Bellemare, Y. Naddaf, J. Veness, M. Bowling, The arcade learning environment: An evaluation platform for general agents, Journal of Artificial Intelligence Research 47 (2013) 253–279.

[224] Z. Li, D. Hoiem, Learning without forgetting, IEEE transactions on pattern analysis and machine intelligence 40 (12) (2017) 2935–2947.

[225] C. Wah, S. Branson, P. Welinder, P. Perona, S. Belongie, The caltech-ucsd birds-200-2011 dataset (2011).

[226] M.-E. Nilsback, A. Zisserman, Automated flower classification over a large number of classes, in: 2008 Sixth Indian Conference on Computer Vision, Graphics & Image Processing, IEEE, 2008, pp. 722–729.

[227] X. Yang, H. Yu, X. Gao, H. Wang, J. Zhang, T. Li, Federated continual learning via knowledge fusion: A survey, IEEE Transactions on Knowledge and Data Engineering 36 (8) (2024) 3832–3850.

[228] Z. Wang, F. Wu, F. Yu, Y. Zhou, J. Hu, G. Min, Federated continual learning for edge-ai: A comprehensive survey, arXiv preprint arXiv:2411.13740 (2024).

[229] J. Chen, J. He, J. Tang, W. Li, Z. Yin, Knowledge efficient federated continual learning for industrial edge systems, IEEE Transactions on Network Science and Engineering (2025).

[230] J. Konečnỳ, B. McMahan, D. Ramage, Federated optimization: Distributed optimization beyond the datacenter, arXiv preprint arXiv:1511.03575 (2015).

[231] L. Qinbin, W. Zeyi, H. Bingsheng, Federated learning systems: Vision, hype and reality for data privacy and protection, CoRR, vol. abs/1907.09693 (2019).

[232] G. I. Parisi, R. Kemker, J. L. Part, C. Kanan, S. Wermter, Continual lifelong learning with neural networks: A review, Neural Networks 113 (2019) 54–71.

[233] S. Grossberg, Nonlinear neural networks: Principles, mechanisms, and architectures, Neural networks 1 (1) (1988) 17–61.

[234] S. Caldas, S. M. K. Duddu, P. Wu, T. Li, J. Konečnỳ, H. B. McMahan, V. Smith, A. Talwalkar, Leaf: A benchmark for federated settings, arXiv preprint arXiv:1812.01097 (2018).

[235] F. E. Casado, D. Lema, R. Iglesias, C. V. Regueiro, S. Barro, Federated and continual learning for classification tasks in a society of devices, arXiv preprint arXiv:2006.07129 (2020).

[236] M. Baron, Convergence rates of change-point estimators and tail probabilities of the first-passage-time process, Canadian Journal of Statistics 27 (1) (1999) 183–197.

[237] A. Haque, L. Khan, M. Baron, Sand: Semi-supervised adaptive novel class detection and classification over data stream, in: Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 30, 2016, pp. 1652–1658.

[238] K. O. Bowman, L. Shenton, Estimation: Method of moments, Encyclopedia of statistical sciences 3 (2004).

[239] G. M. Van de Ven, A. S. Tolias, Three scenarios for continual learning, arXiv preprint arXiv:1904.07734 (2019).

[240] A. Hard, K. Rao, R. Mathews, S. Ramaswamy, F. Beaufays, S. Augenstein, H. Eichner, C. Kiddon, D. Ramage, Federated learning for mobile keyboard prediction, arXiv preprint arXiv:1811.03604 (2018).

[241] M. Shoaib, S. Bosch, O. D. Incel, H. Scholten, P. J. Havinga, Fusion of smartphone motion sensors for physical activity recognition, Sensors 14 (6) (2014) 10146–10176.

[242] P. Tian, W. Liao, W. Yu, E. Blasch, Wscc: A weight similarity based client clustering approach for non-iid federated learning, IEEE Internet of Things Journal (2022).

[243] M. Xie, G. Long, T. Shen, T. Zhou, X. Wang, J. Jiang, C. Zhang, Multi-center federated learning, arXiv preprint arXiv:2005.01026 (2020).

[244] A. Ghosh, J. Chung, D. Yin, K. Ramchandran, An efficient framework for clustered federated learning, Advances in Neural Information Processing Systems 33 (2020) 19586–19597.

[245] D. Caldarola, M. Mancini, F. Galasso, M. Ciccone, E. Rodolà, B. Caputo, Cluster-driven graph federated learning over multiple domains, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2021, pp. 2749–2758.

[246] Y. Kim, E. Al Hakim, J. Haraldson, H. Eriksson, J. M. B. da Silva, C. Fischione, Dynamic clustering in federated learning, in: ICC 2021-IEEE International Conference on Communications, IEEE, 2021, pp. 1–6.

[247] C. Chen, Z. Chen, Y. Zhou, B. Kailkhura, Fedcluster: Boosting the convergence of federated learning via cluster-cycling, in: 2020 IEEE International Conference on Big Data (Big Data), IEEE, 2020, pp. 5017–5026.

[248] M. S. H. Abad, E. Ozfatura, D. Gunduz, O. Ercetin, Hierarchical federated learning across heterogeneous cellular networks, in: ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), IEEE, 2020, pp. 8866–8870.

[249] G. An, The effects of adding noise during backpropagation training on a generalization performance, Neural computation 8 (3) (1996) 643–674.

[250] A. Neelakantan, L. Vilnis, Q. V. Le, I. Sutskever, L. Kaiser, K. Kurach, J. Martens, Adding gradient noise improves learning for very deep networks, arXiv preprint arXiv:1511.06807 (2015).

[251] M. Ester, H.-P. Kriegel, J. Sander, X. Xu, et al., A density-based algorithm for discovering clusters in large spatial databases with noise., in: kdd, Vol. 96, 1996, pp. 226–231.

[252] Y. Ganin, E. Ustinova, H. Ajakan, P. Germain, H. Larochelle, F. Laviolette, M. Marchand, V. Lempitsky, Domain-adversarial training of neural networks, The journal of machine learning research 17 (1) (2016) 2096–2030.

The objective of this PhD thesis is to face the issue of data heterogeneity when training a Federated Learning model across several devices or participants.

There exist a lot of previous approaches that aim to solve this kind of situation. In contrast with them, this dissertation starts with a bird's eye view of the whole data heterogeneity casuistry. Performing this analysis, data heterogeneity can be rigorously classified, making specific problems more approachable.

After classifying the data heterogeneity in such a way, two different algorithms are proposed. On the one hand, we present an algorithm designed for handling time evolving sets of data. On the other hand, we display an algorithm that focuses on identifying conflicting data distributions.