INTERNATIONAL DOCTORAL
SCHOOL OF THE USC

Fernando
Estévez Casado

PhD Thesis

Continual federated machine learning under concept drift

Santiago de Compostela, 2022

Doctoral Programme in Information Technology Research

PhD Thesis

CONTINUAL FEDERATED MACHINE LEARNING UNDER CONCEPT DRIFT

Fernando Estévez Casado

ESCOLA DE DOUTORAMENTO INTERNACIONAL DA UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

PROGRAMA DE DOUTORAMENTO EN INVESTIGACIÓN EN TECNOLOXÍAS DA INFORMACIÓN

SANTIAGO DE COMPOSTELA
2022

Declaración do autor da tese

Don Fernando Estévez Casado

Título da tese: **Continual federated machine learning under concept drift**

Presento a miña tese, seguindo o procedemento adecuado ao Regulamento, e declaro que:

1. *A tese abarca os resultados da elaboración do meu traballo.*
2. *De ser o caso, na tese faise referencia ás colaboracións que tivo este traballo.*
3. *Confirmo que a tese non incorre en ningún tipo de plaxio doutros autores nin de traballos presentados por min para a obtención doutros títulos.*
4. *A tese é a versión definitiva presentada para a súa defensa e coincide a versión impresa coa presentada en formato electrónico.*

E comprométome a presentar o Compromiso Documental de Supervisión no caso de que o orixinal non estea na Escola.

En Santiago de Compostela, 15 de agosto de 2022

Asdo. Fernando Estévez Casado

Autorización dos directores/titores da tese

Continual federated machine learning under concept drift

Don Roberto Iglesias Rodríguez, Profesor Titular da Área de Ciencia da Computación e Intelixencia Artificial da Universidade de Santiago de Compostela

Don Senén Barro Ameneiro, Catedrático da Área de Ciencia da Computación e Intelixencia Artificial da Universidade de Santiago de Compostela

INFORMAN:

Que a presente tese correspóndese co traballo realizado por ***Don Fernando Estévez Casado****, baixo a nosa dirección/titorización, e autorizamos a súa presentación, considerando que reúne os requisitos esixidos no Regulamento de Estudos de Doutoramento da USC, e que como directores/titores desta non incorre nas causas de abstención establecidas na Lei 40/2015.*

De acordo co indicado no Regulamento de Estudos de Doutoramento, declaramos tamén que a presente tese de doutoramento é idónea para ser defendida en base á modalidade de ***Monográfica con reproducción total ou parcial de publicacións****, nas que a participación do doutorando foi decisiva para a súa elaboración e as publicacións se axustan ao Plan de Investigación.*

En Santiago de Compostela, 15 de agosto de 2022

Asdo. Roberto Iglesias Rodríguez
Director de tese

Asdo. Senén Barro Ameneiro
Director de tese

None of us is as smart as all of us.

Ken Blanchard

The measure of intelligence is the ability to change.

Albert Einstein

Agradecementos

Esta tese supón o peche da miña etapa predoutoral, logo de catro anos de esforzo. Tamén simboliza a fin da miña vida como estudante, que non como estudoso, pois non conto con deixar de aprender nunca. Ao longo de todo este tempo, non fixen máis que rodearme de xente boa, á cal estou moi agradecido. Se ben será difícil que poida chegar a devolver toda a axuda recibida, sirvan as seguintes liñas, canto menos, para deixar constancia desta miña gratitude.

En primeiro lugar, quero expresar o meu máis franco agradecemento aos meus directores, Roberto e Senén, pola confianza que depositaron e seguen a depositar cada día en min, así como por tódalas súas achegas á miña formación e pola súa calidade humana. Grazas tamén a Carlos, polo seu interese e boa disposición desde o comezo. Se hoxe me considero un investigador maduro é con certeza froito do bo facer deles tres durante estes anos.

Tamén me gustaría darlle as grazas ao Centro Singular de Investigación en Tecnoloxías Intelixentes (CiTIUS) da USC, e en particular ao Grupo de Sistemas Intelixentes (GSI) e a todo o persoal de apoio do centro, por facilitarme as ferramentas e os espazos necesarios para a realización do meu traballo e por botarme unha man sempre que o precisei. Foi cunha bolsa de verán ofertada polo CiTIUS como comecei a miña andaina na investigación. Esa experiencia foi tan positiva que fixo que xermolara en min o interese por seguir especializándome na Intelixencia Artificial, primeiro cursando o máster, e logo facendo a tese de doutoramento. Estou moi agradecido por esa primeira oportunidade.

Tampouco podo esquecerme dos meus compañeiros e compañeiras, amigos e aimgas, do CiTIUS, por construír entre todos tan bo ambiente de traballo: Julián, Rubén, Álvaro O., Miguel, Pedro, Eric, Celia, Lorenzo, Nico, Álvaro G., Fran, Silvia, Ledicia, Ferre, César, Sabela... Grazas por eses cafés, comidas, ceas e partidos de bádminton. Inclúo aquí tamén a Dylan, que me axudou moito durante os primeiros anos de tese, así como a Denisa, por ser tan boa amiga e contaxiarme sempre a súa alegría. E, como non, mención especial a Marcos,

por todas as matemáticas que me ensinou, todas as gargalladas que me fixo soltar e, en xeral, por todo o apoio que me brindou dentro e fóra do traballo.

I switch to English for a moment, since I would like to thank Prof. Yiannis Demiris for his valuable advise and kind support during my research visit to the Personal Robotics Laboratory (PRL), at Imperial College London. I would also like to extend my gratitude to all the friends I made during those months abroad, who contributed to making my stay in London an unforgettable professional and personal experience: Cédric, Dimitra, Rodrigo, Mohamed, Haining, Ashley, Patrick, David, Ester, Ángela, Alba...

O meu agradecemento vai tamén para as entidades que fixeron de sustento económico. Esta tese doutoral foi financiada polo Ministerio de Universidades de España a través dunha axuda á contratación do programa de Formación de Profesorado Universitario (FPU17/04154). Tamén foi apoiada a través de diferentes proxectos financiados polo Ministerio de Ciencia e Innovación (TIN2017-90135-R e PID2020-119367RB-I00) e pola Consellería de Cultura, Educación e Universidade da Xunta de Galicia (ED431G/01, ED431G/08, ED431C-2018/29, e ED431F2018/02), co cofinanciamento do Fondo Europeo de Desenvolvemento Rexional (FEDER).

Remato dándolles as grazas a Lau, a Esther, a papá e a mamá. Todo ten moito máis sentido ao seu carón.

15 de agosto de 2022

Resumo

Dende hai xa un par de décadas a nosa sociedade está a experimentar unha auténtica explosión tecnolóxica. Isto trae da man unha serie de cambios ao noso redor, entre os que destaca a progresiva integración de 'dispositivos intelixentes' en multitude de tarefas do noso día a día. O exemplo máis claro é o dos nosos teléfonos móbiles, tamén coñecidos como *smartphones*, dos que xa non nos afastamos nin un intre. Sen embargo, non só estamos a falar dos móbiles, senón tamén doutros trebellos como os dispositivos de vestir (*wearables*), os robots de servizo persoal, as 'cousas' da Internet das Cousas (*Internet of Things*), etc. En definitiva, aparellos multi-función equipados con tecnoloxía punta que cada vez ofrecen mellores prestacións. Son precisamente os avances na sensorización, conectividade e capacidade de cómputo as que están a facer que o volume de datos xerado por estes dispositivos medre moi rapidamente. Tales cantidades de información, dispoñibles en tempo real e procedentes de entornos tan diversos, abren un amplo abano de oportunidades de aplicación nunha chea de eidos: educación, saúde, deporte, viaxes, banca, coidado do forgar, interacción social... Se facemos un bo uso destes datos, os dispositivos poden poñerse ao servizo da sociedade, traballando de xeito máis intelixente e autónomo.

A aprendizaxe automática é a rama da Intelixencia Artificial que procura o desenvolvemento de algoritmos que outorgan ás máquinas a capacidade de aprender por si mesmas a resolver problemas complexos baseándose en datos de experiencias previas. No contexto 'multidispositivo' no que vivimos, o emprego de técnicas de aprendizaxe automática facilitará non só que estes aparellos aprendan, senón tamén que evolucionen e axusten o seu comportamento co paso do tempo. A finalidade última de calquera proceso de aprendizaxe automática é crear un modelo de inferencia que sexa capaz de tomar boas decisións cando se enfronte a novas situacións nunca antes vistas. Cando se trata de múltiples dispositivos distribuídos, cada un deles recollerá os seus propios datos, tales como medicións de sensores, imaxes, vídeos,

ubicación, etc. A información local dun só dispositivo non abondará para obter un modelo robusto, canto menos nun prazo de tempo razoable. Pola contra, a colaboración entre os distintos dispositivos, aprendendo dunha maior cantidade e variedade de datos, é un bo xeito de acadar un bo rendemento, así como unha mellor capacidade de xeneralización.

O xeito máis inmediato de levar a cabo esta aprendizaxe multidispositivo sería enviar tódolos datos, desde cada un dos dispositivos, a un servidor central, na nube. Alí sería relativamente doado procesalos e obter un modelo funcional. Sen embargo, este enfoque en xeral non é viable, dado que presenta dúas limitacións importantes. Por un lado está a violación da privacidade dixital dos usuarios. Un dato privado é calquera peza de información que permita identificar a unha persoa na rede. Por exemplo, o seu NIF, número de teléfono, dirección, imaxes, vídeos, historial de búsqueda do navegador, etc. Nos últimos anos, comezáronse a implementar estritas lexislacións en todo o planeta que limitan a recollida e almacenaxe centralizado destes datos para así protexer aos consumidores. Por exemplo, no caso da Unión Europea, contamos co Regulamento Xeral de Protección de Datos (RXPD). O outro problema deste tipo de solucións centralistas está relacionado coa súa escalabilidade. Xuntar tódolos datos nun único punto da rede vai da man dun elevado número de comunicacións entre o servidor e os dispositivos. Ademais, o procesamento da información pode levar moito máis tempo que se o fixésemos en paralelo tomando porcións máis pequenas. Estes custes en almacenamento, comunicacións e computación poden medrar exponencialmente conforme medra o número de dispositivos involucrados. En definitiva, podemos afirmar que calquera proposta centralista vólvese, dunha forma ou doutra, inviable.

Unha mellor opción para aprender neste tipo de escenarios, onde os datos están espallados por natureza, é facelo de xeito distribuído. É dicir, conservando a infromación privada localmente, nos dispositivos, de modo que sexan eles mesmos os que aprendan o modelo. Neste contexto xorde a aprendizaxe federada (en inglés, *federated learning*). Trátase dun novo paradigma de aprendizaxe automática proposto orixinalmente por Google en 2016. A idea é aprender o modelo colaborativamente, levando a cabo una serie de 'roldas de adestramento' que alternan entre actualizacións locais e consensos globais. Habitualmente, o modelo que se constrúe é unha rede neuronal profunda, ou DNN (do inglés, *deep neural network*), composta por un conxunto—en xeral, moi grande—de parámetros. O proceso de adestramento consiste en axustar os valores destes parámetros entre todos os dispositivos, denominados 'clientes'. Ao principio, os parámetros son inicializados con valores aleatorios no servidor e compartidos con todos os clientes. Feito iso, comezan as roldas de adestramento. Cada cliente fai un

axuste local dos parámetros en base aos seus datos privados. Para isto empréganse técnicas de optimización como as baseadas no descenso estocástico do gradiente, ou SGD (do inglés, *Stochastic Gradient Descent*). Logo, os clientes envían as actualizacións de volta á nube mentres o servidor agarda por todas elas. Cando xa todos os clientes fixeron a súa aportación, o servidor leva a cabo a fase de consenso global, que normalmente consiste en facer unha media ponderada para cada un dos parámetros. Con isto remata a primeira rolda de aprendizaxe, pero o proceso repítese as veces que sexa oportuno: o servidor comparte os novos parámetros consensuados cos clientes, estes realizan un novo axuste, etc. Ao remate, obtense un modelo que é capaz de xeneralizar o coñecemento de tódolos participantes sen que eles teñan que revelar información privada. Ademais, a carga de traballo do lado do servidor é reducida, xa que a maior parte do cómputo recae nos clientes, polo que se mellora a escalabilidade do sistema.

Pese a que a aprendizaxe federada xa se ten aplicado con éxito en diversos casos de uso (texto preditivo, detección de software malicioso, diagnóstico médico, etc.), trátase dun paradigma moi recente e aínda queda moito traballo por diante para que se convirta nunha tecnoloxía madura. Sen dúbida, un dos retos que está atraendo máis a atención dos investigadores é desenvolver algoritmos federados que sexan robustos a escenarios onde os datos teñen propiedades estatísticas variables. Dita variabilidade pode atender a dous eixes. O primeiro é o eixe espacial, que abarca todas aquelas situacións nas que as distribucións locais dos datos dos clientes son moi dispares, o que se coñece como datos 'non-IID' (do inglés, *non independent and identically distributed*). O segundo é o temporal, pois ten que ver con fluxos de datos non estacionarios, é dicir, que van cambiando ao longo do tempo. Se ben existen xa bastantes traballos que procuran dar solución a problemas no eixe espacial, apenas hai propostas no eixe temporal.

A práctica totalidade de algoritmos—non só federados, senón de aprendizaxe automática en xeral—asumen que os datos de adestramento son tomados dunha distribución estable e estacionaria. Isto, sen embargo, é pouco habitual nos problemas reais. Menos o é aínda en contextos que involucran a múltiples dispositivos, onde a interacción destes cos seus usuarios e co entorno é frecuente. Aprender un modelo neste tipo de situacións é complexo. Por un lado, non podemos asumir que haxa datos no comezo, senón que irán estando dispoñibles co tempo. Polo tanto, o modelo terá que ser aprendido pouco a pouco, de maneira gradual. Por outra banda, o fluxo de datos é potencialmente infinito, pero non así o noso almacenamento. Daquela, non será viable tratar de gardar todos os datos na memoria, senón que teremos que

eliminar a información segundo a vaiamos procesando. Finalmente, hai que ter en conta que as propiedades estatísticas dos datos poden variar co tempo de maneira imprevisible. Isto quere dicir que, xeneralmente, non poderemos anticiparnos aos cambios. Este fenómenos é o que se coñece como 'deriva de concepto', ou *concept drift*, en inglés. Unha deriva de concepto soe traducirse en que o modelo aprendido en base a experiencias previas xa non se axusta ben aos novos datos, o que leva a unha caída no rendemento. A solución pasa por detectar cando se dan estes cambios para así actualizar o modelo en consecuencia. É importante destacar que, nos problemas multidispositivo, aprender de maneira continuada a partir de datos non estacionarios vólvese aínda máis complexo. Cada dispositivo terá que operar sobre un fluxo de datos local independente. Polo tanto, non é realista asumir que todos eles traballarán ao unísono de maneira coordinada para adestrar un modelo federado. Máis ben, cada cliente colaborará ao seu ritmo, procesando datos a velocidades distintas, podendo conectarse e desconectarse en calquera intre, etc.

O obxectivo desta tese de doutoramento é o de desenvolver novas estratexias de aprendizaxe federada que, mantendo tódalas vantaxes que esta tecnoloxía xa proporciona, permitan tamén lidar con escenarios continuos, en situacións non estacionarias suxeitas a derivas de concepto. Deste xeito, ao longo do presento documento propoñemos novos algoritmos de aprendizaxe federada continua e aplicámolos e avaliámolos en distintos casos de uso relacionados coa telefonía móbil e a robótica de servizos. A tese estrutúrase nun total de seis capítulos. No Capítulo 1 presentamos o problema da aprendizaxe federada continua, a nosa motivación e os obxectivos que abordamos. No Capítulo 2 facemos unha revisión en profundidade do estado da arte, centrándonos en particular na aprendizaxe federada e a deriva de concepto. Os Capítulos 3, 4, e 5 recollen as nosas contribucións principais. Ao longo deles propoñemos diferentes estratexias de aprendizaxe federado e continuo: CDA-FedAvg, ECFL e FLfD. Ademais, avaliamos as nosas propostas en distintos casos de uso, incluíndo o recoñecemento da actividade humana en *smartphones* e a asistencia a usuarios en cadeiras de rodas robóticas. Finalmente, no Capítulo 6 achegamos as nosas conclusións e tamén facemos unha reflexión sobre as futuras liñas de traballo. A continuación imos describir cun pouco máis de detalle cada unha das contribucións principais.

No capítulo 3, formalizamos o problema da aprendizaxe baixo deriva de concepto en sistemas multidispositivo. Así mesmo, propoñemos unha primeira aproximación á aprendizaxe federada continua: CDA-FedAvg (do inglés, *Concept-Drift-Aware Federated Averaging*). O noso método é unha extensión de FedAvg, o algoritmo máis popular a día de hoxe en apren-

dizaxe federada. CDA-FedAvg mantén todas as vantaxes de FedAvg, incluíndo a protección da privacidade dos clientes, pero sendo ademáis capaz de detectar derivas de concepto virtuais e adaptarse a elas. Unha deriva 'virtual' é unha variación na probabilidade marxinal do espazo de entrada (é dicir, non afecta ás saídas que, como respostas discriminadas polo modelo, seguerán sendo as mesmas). En CDA-FedAvg, cada cliente dispón de dúas memorias: unha a curto prazo e outra a longo prazo. A memoria a curto prazo permite analizar de maneira continua, en tempo real, o fluxo de datos local. Para detectar as derivas conceptuais, emprégase un algoritmo que, dada unha medida de confianza obtida para cada mostra da memoria a curto prazo, é capaz de cuantificar a disimilitude entre as distribucións dos datos máis antigos e dos novos. Se se produce unha deriva (alta disimilitude), o modelo federado é actualizado empregando unha técnica coñecida como *rehearsal*, que basicamente consiste en efectuar novas roldas de adestramento empregando datos tanto novos coma vellos. Para iso, almacénase na memoria a longo prazo un conxunto de datos representativo de cada concepto. Deste xeito, CDA-FedAvg permite levar a cabo un adestramento durante longos periodos de tempo sen que se produzan caídas de rendemento. O método é capaz en todo momento de determinar cando debe aprender e que información debe empregar para elo.

Xa comentamos antes que a gran maioría de propostas de aprendizaxe federada consisten no adestramento distribuído e paralelo dunha DNN. De feito, os métodos habituais para a agregación global están especificamente deseñados para este tipo de algoritmos. Se ben as DNNs proporcionan bos resultados nunha chea de aplicacións, tamén presentan certas limitacións, como a complexidade computacional, a opacidade nos modelos ou a tendencia ao 'sobreaxuste'. Porén, existe unha necesidade por explorar novas solucións federadas que permitan traballar con outras estratexias de aprendizaxe distintas ás DNNs. Con isto en mente, no capítulo 4 introducimos ECFL (*Ensemble and Continual Federated Learning*). Trátase dunha arquitectura baseada en comités (*ensembles*) para a aprendizaxe federada continua. Un 'comité' consiste en combinar as predicións feitas por múltiples modelos para dar unha única resposta consensuada. Nós propomos que o modelo federado sexa un comité, de xeito que estea composto de múltiples modelos locais independentes, un por cliente. Isto permite levar a cabo o adestramento federado sen restricións sobre o tipo de algoritmo de aprendizaxe a empregar, o que lle outorga á nosa proposta certa vantaxe en canto a simplicidade, flexibilidade e robustez. ECFL tamén está deseñada para aprender ao longo do tempo. Por unha banda, de xeito similar a en CDA-FedAvg, integra mecanismos de detección e adaptación aos cambios. Neste caso, manter o comité global actualizado é tan sinxelo como trocar modelos locais

obsoletos por outros máis recentes. Ademais, grazas a un sistema de etiquetado baseado na confianza do modelo global, permite traballar con fluxos de datos onde a información está parcialmente etiquetada ou a dispoñibilidade de etiquetas non é inmediata. Finalmente, cabe tamén destacar que a nosa proposta incorpora un sistema de selección de modelos locais, denominado DEV (*Distributed Effective Voting*), que lle permite filtrar do comité a aqueles clientes que aportan pouco ao conxunto ou que incluso resultan prexudiciais.

Ámbalas dúas propostas que acabamos de describir, CDA-FedAvg e ECFL, foron avaliadas en profundidade. Levamos a cabo experimentos e simulacións empregando distintos conxuntos de datos e escenarios. Principalmente, as probas xiraron arredor do recoñecemento da actividade humana con *smartphones*. O obxectivo é identificar que está a facer o usuario (camiñar, correr, subir e baixar escaleiras, ir en bicicleta, etc.) a partires dos datos procedentes dos sensores inerciais do dispositivo (acelerómetro, xiróscopo e magnetómetro). A elección deste problema ven motivada por dúas razóns. Por unha banda está o encaixe natural que ten aquí a aprendizaxe federada continua: falamos dun problema real que involucra a múltiples usuarios e teléfonos intelixentes, cunha adquisición de datos de natureza sensible que se prolonga no tempo. Por outra parte, destacamos a súa utilidade real en varios contextos: práctica deportiva, monitorización da saúde, localización en interiores, etc. A maiores, coa idea de probar as nosas aportacións sobre conxuntos de datos de ampla difusión no ámbito, realizamos outros experimentos no problema do recoñecemento de díxitos manuscritos, empregando os conxuntos de datos máis habituais (MNIST, SVHN, USPS, etc.).

Os resultados obtidos en todas as probas demostran un moi bo rendemento, tanto de CDA-FedAvg como de ECFL. A diferenza do que sucede con outros métodos que son estado da arte, como FedAvg ou FedProx, as nosas propostas son capaces de aprender de maneira continuada adaptándose ás derivas de concepto. Ademais, no caso particular de ECFL, o sistema DEV para a selección dos membros do comité outórgalle robustez ao modelo global aínda cando existen modelos locais atípicos. Polo tanto, podemos afirmar que CDA-FedAvg e ECFL son dúas alternativas viables para abordar problemas multidispositivo. A selección dunha ou doutra virá condicionada polas necesidades específicas do problema. Nalgúns casos, será conveniente empregar estratexias baseadas en DNNs, polo que CDA-FedAvg será a mellor opción. Por exemplo, en tarefas de visión por computador ou de procesamento de linguaxe natural. Noutros casos, será mellor empregar algoritmos de aprendizaxe máis tradicionais (árboles de decisión, máquinas de vectores de soporte, etc.), polo que empregaremos ECFL. Situacións deste

tipo serían aquelas onde a dispoñibilidade de datos etiquetados sexa reducida ou involucre dispositivos con baixa capacidade de cómputo.

CDA-FedAvg e ECFL foron avaliados mediante simulación, empregando datos que maioritariamente foron tomados por teléfonos móbiles intelixentes. Se ben é certo que o recoñecemento da actividade humana é unha tarefa complexa e se procurou que a experimentación recrease casos de uso realistas, sempre é desexable probar os algoritmos en tempo real e na maior variedade de problemas e de hardware posible. Nós cremos que, máis aló dos *smartphones*, hai outras plataformas que se poden beneficiar enormemente da aprendizaxe federada nos próximos anos. É, por exemplo, o caso dos robots de servizo, é dicir, robots semi ou totalmente autónomos que proporcionan servizos para o benestar da xente en distintos dominios como a axuda no fogar, a restauración, a atención sanitaria, ou os transportes. Motivados por explorar potenciais aplicacións das nosas tecnoloxías na robótica de servizos, puxémonos en contacto cos investigadores do *Personal Robotics Laboratory* (PRL) do Imperial College London, no Reino Unido. O PRL trátase dun dos laboratorios punteiros a nivel mundial en investigación en robótica de servizos. A última das contribucións recollidas nesta tese de doutoramento é froito dunha estadía de catro meses en Londres, nas instalacións do PRL. Dito traballo, presentado no Capítulo 5, é un sistema colaborativo de asistencia a usuarios de cadeiras de rodas robóticas baseado na aprendizaxe federada.

Os sistemas de navegación habituais en cadeiras de rodas robotizadas empregan mapas construídos de antenmán dos edificios. É dicir, a autonomía do robot está restrinxida a aqueles sitios para os que se dispón dun mapa. Se a isto lle engadimos o feito de que os entornos polos que nos desprazamos no noso día a día cambian constantemente, esta aproximación faise inviable. Unha alternativa interesante é empregar técnicas de aprendizaxe automática que permitan ás cadeiras tomar decisións durante a navegación, sexa cal sexa o lugar. Así, a nosa contribución no Capítulo 5 baséase na aprendizaxe federada a partires de demostracións, ou *Federated Learning from Demonstration* (FLfD), en inglés. Propoñemos aprender unha DNN capaz de estimar os controis da cadeira (a través dun *joystick*) tomando como entradas as medicións de proximidade e as imaxes proporcionadas polos sensores láser e a cámara cos que conta o robot. Nos nosos experimentos, avaliamos o rendemento da proposta en diversos escenarios reais, o que amosou unha boa capacidade de xeneralización en localizacións non vistas con anterioridade. Aínda que neste traballo os datos de adestramento foron obtidos de antemán dun único usuario, é interesante levar a cabo a aprendizaxe de maneira federada e

continua. Isto permitirá involucrar a múltiples persoas, robots e entornos ao longo do tempo, o que se traducirá en datos máis variados e modelos máis robustos e adaptativos.

No seu conxunto, esta tese de doutoramento supón un avance no campo da aprendizaxe multidispositivo. O noso traballo contribuíu á maduración e enriquecemento da emerxente aprendizaxe federada. Partindo dos seus cimentos, a nosa investigación procurou avanzar cara a aplicabilidade deste paradigma a problemas reais. Isto supuxo lidar con escenarios nos que os datos non están dispoñibles desde o comezo, senón que son adquiridos de maneira gradual polos dispositivos e poden evolucionar de xeito impredicible ao longo do tempo. Deste xeito, abrimos unha nova área de traballo en aprendizaxe federada continua. Esta tese céntrase principalmente en abordar o problema da deriva de concepto con datos que non son estacionarios. Sen embargo, tamén se atenden outras cuestións, como a aprendizaxe a partir de datos parcialmente etiquetados ou o aforro en custos de almacenaxe, comunicacións e computación. As contribucións a nivel teórico tamén foron aplicadas a casos de uso reais. Por un lado, traballamos con teléfonos intelixentes, abordando o recoñecemento da actividade humana. Por outra banda, levamos a aprendizaxe federada ao eido da robótica de servizos, desenvolvendo con éxito unha solución para a asistencia a usuarios de cadeiras robóticas durante a navegación.

A pesar de tódalas contribucións que acabamos de mencionar, cremos que esta tese supón tan só o punto de partida. No futuro, a nosa intención é avanzar en paralelo nos dous eixes xa antes mencionados, espacial e temporal, co obxectivo de mellorar cada vez máis a aplicabilidade da aprendizaxe federada continua a problemas reais. Para iso, pretendemos unificar a investigación en ambos eixes. Isto implica lidar non só con derivas de concepto ao longo do tempo, senón tamén con datos 'non-IID' (heteroxéneos) entre os distintos dispositivos. Estamos especialmene intersados en explorar estratexias de personalización que permitan adaptar o modelo federado ás particularidades locais sen perder a capacidade de xeneralización. Pensamos que dita adaptación pode facerse non só a nivel individual, senón tamén grupal. Por outra banda, cremos que é importante seguir a promover a transferencia destas tecnoloxías a novos casos de uso. Existen moitas aplicacións potenciais no contexto dos *smartphones*, *wearables* e a IdC. Algúns exemplos serían a práctica deportiva, a monitorización e coidado da saúde, a automatización do fogar ou a identificación biométrica. A máis longo prazo, irán aparecendo tamén novas oportunidades no eido da robótica, tanto na industria coma no sector servizos. En definitiva, prevemos un futuro cheo de retos e oportunidades para a aprendizaxe federada continua.

Contents

List of Figures

List of Tables

Acronyms

AI artificial intelligence. 7

CC BY Creative Commons Attribution. xxv, 23, 89, 119, 142, 144, 145

CDA-FedAvg Concept-Drift-Aware Federated Averaging. xxiii–xxv, xxvii, xxviii, 6, 23, 24, 26, 28–30, 33–41, 43–47, 50, 55, 62, 64–66, 68–70, 73, 90, 91, 93, 119–122, 124–126, 132, 134, 135, 137–139

CFL continual federated learning. 23, 24, 27, 37, 39, 47, 51, 62, 69, 73, 90–93, 95–97, 124, 127, 133, *see* CL & FL

CL continual learning. 17, 18, 21, 23–25

CNN convolutional neural network. xxviii, 40, 124, 138, 139

DEV Distributed Effective Voting. xxiv, 59, 60, 66, 67, 69, 90, 91, *see* EV

DL deep learning. 9, 16, 47, 48, 50, 78, 91

DML distributed machine learning. xxiii, 9–12, 50

DNN deep neural network. 2, 5, 13, 16, 33, 48–51, 74, 76, 78, 82, 91, 92

ECFL Ensemble and Continual Federated Learning. xxiii–xxv, xxvii, xxviii, 6, 47–49, 51, 52, 54–57, 60–70, 73, 90, 91, 93, 94, 96, 119, 120, 125, 130–139

EV Effective Voting. 58, 59

FedAvg Federated Averaging. xxiii–xxv, xxvii, xxviii, 13–15, 23, 25, 28, 33, 35–44, 50, 63–66, 68, 69, 71, 79, 90, 91, 120–122, 124–126, 133–135, 137–139

FedWeIT Federated Continual Learning with Weighted Inter-client Transfer. 24

FIL-VER Federated Incremental Learning with Variational Embeddings Rehearshal. 25

FL federated learning. xxiii, 2, 3, 5–7, 12–16, 23–26, 38, 44, 47–51, 61–63, 69, 73, 76, 78, 89–92, 94

FLfD Federated Learning from Demonstration. xxiv, 6, 74, 76, 78, 79, *see* LfD

FLwF-2T Federated Learning without Forgetting - 2 Teachers. 25

FNN feed-forward neural network. 63–65, 130, 134, 135

FTL federated transfer learning. 16, *see* FL

GDPR General Data Protection Regulation. 2, 3

GLM generalized linear model. 63–65, 69, 130, 137, 138

HAR human activity recognition. xxviii, 5, 38, 39, 46, 62, 70, 73, 90, 92, 97, 120, 124, 132, 135, 137–139

HFL horizontal federated learning. 16, *see* FL

ID3 Iterative Dichotomiser 3. 16, 50

IID independent and identically distributed. xxiii, xxiv, xxvii, xxviii, 11, 15, 40–42, 69, 121, 124, 125, 133, 134

IoT Internet of Things. 1, 9, 48

JCR Journal Citation Reports. 6, 141, 143, 145

LfD learning from demonstration. 76–78, 92

LSTM long short-term memory. 40, 83

ML machine learning. 1, 3, 5, 7–11, 15, 17, 48–50, 76, 78, 91, 92, 127

NB Naïve Bayes. 53, 63–65, 69, 91, 130, 134, 135, 137, 138

NELL Never-Ending Language Learner. 17

NLP natural language processing. 9, 91, 97

non-IID non independent and identically distributed. 15, 77, 93, 94, *see* IID

P2P peer-to-peer. 10, 16

PDF probability density function. 32, 33

PRL Personal Robotics Lab. 73, 80

RF Random Forests. 63–65, 69, 130, 133–135

RMSE Root Mean Squared Error. xxvii, 85, 86

SGB Stochastic Gradient Boosting. 63–65, 69, 130, 134, 135

SGD Stochastic Gradient Descent. 11, 13, 48, 53, 79

SJR Scientific Journal Rankings. 141, 143, 145

SMC Secure Multi-Party Computation. 15, 61

SVM support vector machine. xxiv, 8, 11, 16, 47, 50, 51, 53, 63–69, 130, 131, 133–135, 138

SVRG Stochastic Variance Reduced Gradient. 13

VFL vertical federated learning. 16, *see* FL

Chapter 1

Introduction

Over the last few decades, our society has experienced a technological explosion which, among other things, has gradually surrounded us with *smart devices* that are part of our daily lives. We are talking about mobile phones, of course, but also wearables, service robots, "things" from the Internet of Things (IoT), etc. In short, multifunctional devices with cutting-edge technology, including progressive sensorization and connectivity, that allows the volume of data generated to grow rapidly. Having such amount of data collected in real working conditions from distributed environments, together with a good intercommunication between devices, opens up a new world of application opportunities in all human domains [1]: education, health, sport, travel, banking, social interaction, etc. In particular, the use of *multi-device machine learning* will allow to evolve, adapt, and fine-tune the behavior of the devices in order to perform better and better, thus improving their autonomy and, in consequence, benefiting the consumers.

The ultimate goal of any machine learning (ML) process is, based on previously collected data, to create an inference model that accounts for prior experience and predicts optimal decisions when facing novel situations. When it comes to distributed devices, each of them collects their own data, such as sensor measurements, photos, videos, location, etc. In this context, local information from a single device may not be enough to obtain a robust model—at least within a reasonable amount of time. Instead, collaboration is a good way to learn from larger and more varied datasets, that allows for greater generalization capability and better performance.

The most immediate way to perform multi-device ML would be a centralized client-server approach. This involves uploading data from all the devices, the *clients*, to a central node

in the cloud, which is the *server*. In the server, the information can be jointly processed to provide insights and produce an effective inference model. Afterwards, the model can either stay on the server, so that clients can use it remotely by making queries, or be shared over the network, so that everyone can run it locally. Nevertheless, this approach is currently unfeasible in a large number of applications because it poses some major issues.

The main concern is that it violates users' *digital privacy*. Devices, by interacting with their users, often collect sensitive information, i.e., any data that would allow to identify the person on the Internet. For example, the ID card, telephone number, or address, but also pictures, videos, browsing history, or geolocation. In recent years, governments have been implementing data privacy legislation that controls and restricts centralized data collection in order to protect the consumers. Examples of this are the European Commission's General Data Protection Regulation (GDPR) [2], or the U.S. Consumer Privacy Bill of Rights [3]. In the particular case of Europe, the consent (GDPR, Art. 6) and data minimization principles (GDPR, Art. 5) limit data collection and storage only to what is consented by the consumer and absolutely necessary for processing.

Centralized solutions also have problems of *scalability*, both in storage and communication costs, as well as in computing speeds. Transferring huge amounts of data from thousands of devices to a central server over the network can be time-consuming and expensive. Note that communications may be a continuous overhead, as information from real environments is continuously been updated. This can be especially challenging in tasks involving unstructured data, e.g., in video analytics [4]. On top of this, connectivity and latency issues may also arise. Similarly, central computing can take much more time than parallel processing of smaller parts of data. Besides, it may imply long propagation delays and incur unacceptable latency for applications in which real-time decisions are critical, e.g., in autonomous driving systems [5].

A better option for learning in this kind of scenarios, where data are naturally distributed, seems to be a decentralized approach, keeping private information locally and pushing computing to the edge. In this sense, the main paradigm at present is *federated learning* (FL) [6, 7]. It consists of collaboratively learning a shared model, usually a deep neural network (DNN), by alternating local update stages in the client devices with global consensus in the cloud. The learnable parameters of this model are initialized on the server. In each federated round, each client receives the current parameter set from the server, performs a local adjustment using its private dataset, and sends the updated parameters back. The local contributions are then aggregated on the server, normally applying a weighted average. As a result, each local

learner benefits from the experience of the other clients through the shared global model, without explicitly accessing their sensitive data. In this way, the users' digital privacy is protected. Furthermore, the workload on the server is lightened since all devices participate in the training, thus enhancing scalability.

Federated learning has already been successfully applied in many real-world tasks, such as mobile keyboard prediction [8], malware detection [9], energy demand prediction [10], medical diagnosis [11], or autonomous navigation [12]. Nevertheless, despite its current popularity and visible advantages over traditional centralized architectures, we are talking about a very recent paradigm and there are still many open issues to face [6, 7, 13]. In fact, at the beginning of this research project, back in 2018, there were only a couple of publications in the field, and the term "federated learning" was almost unknown. It has been in the last 3 years when an exponential growth has been experienced.

Undoubtedly, one of the challenges that is currently attracting the most attention from researchers is developing new federated solutions robust to scenarios where data have varying statistical properties [14–17]. In particular, there is a need for new algorithms capable of dealing with problems where the data are *non-stationary*, i.e., susceptible to change over time. Most ML approaches, including federated ones, assume that training data are sampled from a stationary, stable, distribution. Nonetheless, in the real world, this scenario is rather uncommon. In contrast, it is more likely that devices collect data on a continual basis, dealing with changing environments and unbounded sequences of data. Therefore, it is important to develop learning strategies that are adaptive over time. A recent example that highlighted this need was the COVID-19 pandemic. During these last few years, our daily lives changed dramatically, which also affected our sentiment and consumption preferences. These transformations made that many recommendation algorithms and models became useless and had to be replaced [18].

Continual ML and non-stationary data entail a number of challenges and constraints:

- Data are not given beforehand, but become available over time. Therefore, the model has to be learned gradually, retaining prior relevant knowledge while acquiring new one, avoiding the well-known *catastrophic forgetting* [19].
- The data stream may be infinite. Thus, it can be infeasible to store all data in memory and each instance may be accessed a limited number of times. Even if storage is not a constraint, data can disappear for legal and privacy reasons (e.g., the "right to be forgotten", Art. 17 of the EU GDPR [2]).

- Statistical properties of data may vary over time in unpredictably ways. This phenomenon is known as *concept drift* [20], and can make the inducted knowledge of past data no longer relevant, leading to performance drops. Hence, it is important to detect these changes and adapt the model accordingly (see Section 2.3.1).

In multi-device applications, the problem of non-stationarity becomes even more complex, since each device deals with a local and independent data stream. Thus, learning a federated model in a continual manner involves additional restrictions. To give a few examples:

- Each client can drop in and drop out at any time, whether it is conditioned by resources such as battery or connectivity, or by data availability.

- Each client may collect and process data at different speeds depending on its capabilities and environment.

- Data distribution may vary not only over time, but also between clients.

- Concept drift can be experienced or detected at different times by each client.

It is in this unexplored context of continual and federated learning for multi-device applications that the present work is framed. Figure 1.1 shows a Venn diagram that helps to situate our research.

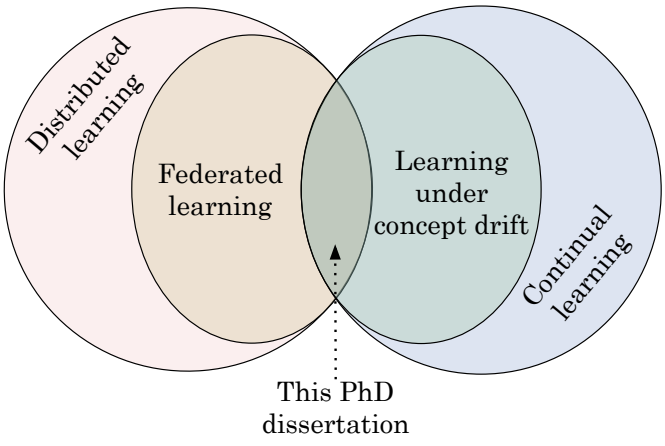


Figure 1.1: High-level contextualization of this PhD thesis.

1.1 PhD objectives and contributions

When this research project was proposed in early 2018, federated learning was still largely unknown. At that time, we wondered what would be the best way to accomplish machine learning for a society of devices. What was clear to us was that, whatever the answer would be, it had to be suitable for the real world, allowing for adaptive learning over time under the intrinsic constraints of multi-device environments. We initially thought about developing a continual ML strategy for scalable, privacy-friendly, and adaptive multi-device learning. The name we gave to it was *'glocal' learning*, reflecting the combination of local training, in the devices, with subsequent global refinement, in the cloud. We soon abandoned that name to adopt the term 'federated', which had recently been proposed by Google [21, 22]. Nevertheless, although the name changed, the goal remained the same.

Federated learning has shown great potential in multi-device contexts, enabling learning in a distributed manner from the data of many users while protecting their privacy. However, little progress has been made in formulating solutions for continual and adaptive FL. This is the challenge we address in this dissertation. Thus, our research was guided by the objective of **developing federated learning approaches that maintain all the existing advantages of this ML paradigm while allowing to handle continual scenarios with non-stationary data and concept drift.** We propose new algorithms and apply them to different use cases related to smartphones and service robotics.

The following are the main contributions of this PhD thesis:

1. **Formulation of two continual federated learning solutions robust to concept drift.** On the one hand, a new algorithm is proposed that adapts the most common FL method to continual and non-stationary settings. On the other hand, we introduce a new continual federated architecture that differs from the state of the art in that it is based on ensembles of local learners, thus enabling the training of models other than DNNs. Our proposals allow the temporal evolution of the models, detecting and adapting to concept drift. We also achieve a reduction in storage, computation and communications.

2. **Evaluation and application of our solutions to real use cases.** We want our algorithms to be useful in real-world problems, so it is important to test them in real use cases. Therefore, a large part of our experimental evaluation relies on the task of human activity recognition (HAR) using smartphones. The idea is to provide these devices with the ability to identify what the user is doing (walking, running,

sitting, etc.). This is very useful in a variety of contexts, such as sports, health care, or indoor localization. In addition, we also make a first approach to FL in service robotics. We propose a federated strategy that enables smart robotic wheelchairs to provide assistance to their users. The robot learns to solve day-to-day tasks, such as going through doorways or avoiding obstacles. This last application was developed in collaboration with the Personal Robotics Laboratory at Imperial College London (United Kingdom).

Our main contributions and results have been published in high impact journals, as well as presented in conferences of international relevance. Considering both journal articles and conference proceedings, this research has resulted in a total of 5 different papers (2 of them published in JCR-Q1 journals and another one presented in a CORE A international conference). This dissertation brings together all the contributions made during these years of work. It is structured in six chapters. Apart from the present introduction, it is organized as follows:

- **Chapter 2** provides a background in multi-device and continual learning, introducing and reviewing the fundamental aspects related to the research carried out in this thesis.
- **Chapter 3** describes Concept-Drift-Aware Federated Averaging (CDA-FedAvg), the first method to address continual federated learning.
- In **Chapter 4**, we propose Ensemble and Continual Federated Learning (ECFL), a continual federated architecture that relies on ensembles for local adaptation to change and global knowledge aggregation.
- **Chapter 5** presents our solution for active assistance to smart wheelchair users, a use case in service robotics that relies on Federated Learning from Demonstration (FLfD).
- Finally, **Chapter 6** presents our conclusions, as well as some thoughts on future work, thus marking the end of this PhD dissertation.

CHAPTER 2

BACKGROUND

Machine learning is a very broad research topic, and possibly the most prevalent one nowadays in the field of artificial intelligence (AI) and computer science. In this chapter, we review the most relevant concepts within the scope of this PhD dissertation, with a particular focus on the federated learning paradigm and the concept drift phenomenon. The aim is to provide a solid foundation on multi-device and continual ML.

2.1 Machine learning

Machine learning is the subfield of AI that focuses on the development of algorithms and systems that give machines the ability to solve complex problems by learning and improving automatically through experience, without explicitly being programmed to produce a particular outcome. *Experience* refers to the past information available to the learner, which typically takes the form of data samples collected via interaction with the environment. ML algorithms are able to optimally configure themselves, so that they become better and better at achieving the desired task. This process of adaptation is called *training*. The output of a ML algorithm after being trained on data is the *model*. A good model will account for prior experience and predict accurate decisions, not only when presented with the training inputs, but also when facing novel situations. Formally:

Definition 2.1 *Let the tuple* $(\mathbf{x}_i, \mathbf{y}_i)$ *be a data sample, where* $\mathbf{x}_i \in \mathcal{X}$ *is an input vector, also known as* features*, and* $\mathbf{y}_i \in \mathcal{Y}$ *is the desired outcome, possibly unknown. Let* $\mathcal{D} = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^{M}$ *be a training dataset, where* M *is the number of samples. A ML model is*

a mapping function defined over $\mathcal{X}$,

$$\begin{aligned} f : \mathcal{X} &\longrightarrow \mathcal{Y} \\ \mathbf{x} &\mapsto f(\mathbf{x}) = \hat{\mathbf{y}}, \end{aligned} \tag{2.1}$$

that establish an output for every possible input.

ML approaches are traditionally divided into four main categories, attending to the type of feedback that is given to the algorithm: (i) supervised learning, (ii) unsupervised learning, (iii) semi-supervised learning, and (iv) reinforcement learning. In *supervised learning*, the algorithm is presented with example inputs and their desired outputs, known as *labels*, and the goal is to learn how to map inputs to outputs. Within this category we usually distinguish between classification and regression methods. In *unsupervised learning*, instead, no labels are given, so the algorithm tries to find relevant information just based on the inputs. Examples of unsupervised techniques are clustering and dimensionality reduction. *Semi-supervised learning* uses a—generally small—amount of labeled data, supplemented by a comparatively large unlabeled set. As in supervised learning, the ultimate goal is to learn to predict the desired outputs given the inputs, but with the hope that the unlabeled dataset can help achieve a better performance. Finally, *reinforcement learning* allows the machine to interact with the environment where it must reach a certain goal (e.g., driving a vehicle or playing a video game). During this interaction, the machine receives feedback in the form of rewards, and tries to maximize this reward over a course of actions.

The term "machine learning" was coined in 1952 by Arthur Samuel, from IBM, who demonstrated that computers could be programmed to learn to play checkers [23]. This was followed by the implementation of the first neural network, the perceptron, by Frank Rosenblatt in 1957 [24], based on the conceptualization proposed by Warren McCulloch and Walter Pitts [25]. The field of ML continued to grow throughout the 1960s and 1970s, with the development of more sophisticated neural networks, such as the multilayer perceptron (MLP) [26], and other supervised approaches, like the nearest neighbor method [27]. It was also in this period that the term "reinforcement learning" was first used [28], and some of the most popular unsupervised methods were proposed, e.g. k-means [29]. During the 1980s and 1990s, the increase in computing power and availability of data led to the development of more complex and capable methods, including decision trees [30], support vector machines (SVMs) [31], and also the first ensemble techniques, such as Adaboost [32]. It was also around this time, in 1986, when the backpropagation algorithm was formally introduced by Rumelhart, Hinton and

Williamss [33] as a procedure to train neural networks. The 2000s saw the rise of Big Data and the need for more powerful ML algorithms to deal with it. As a result, deep learning (DL) architectures [34] quickly became popular and began to be used in commercial applications. Thanks to these techniques and the availability of inexpensive hardware and data, which made them practical, the pace of research and development has accelerated dramatically since 2005, which in turn has led to a substantial growth in the number of ML-based solutions on the market. We can find examples of applications in such diverse fields as computer vision [35, 36], natural language processing (NLP) [37, 38], finance [39], entertainment [40, 41], spacecraft engineering [42], or medicine and biomedicine [43, 44].

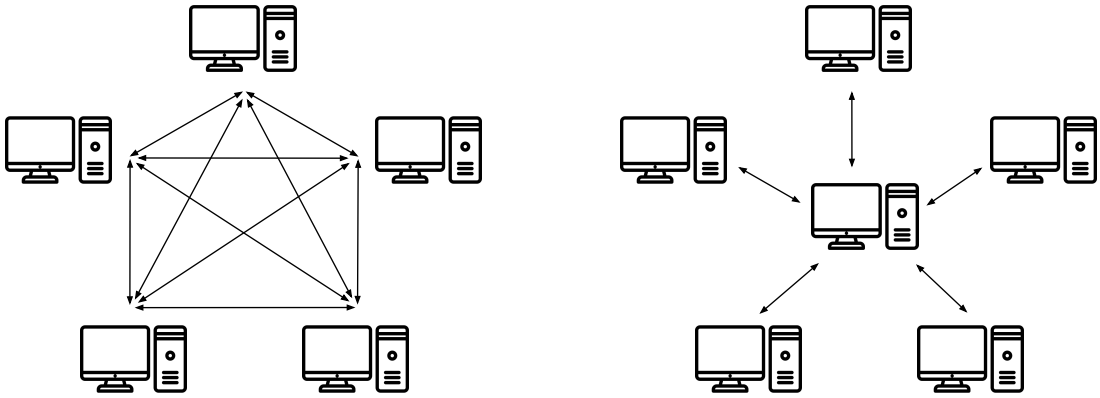
Over the last decade, the improvement and cheapening of electronic components, as well as the advances in communications (evolving from 4G to 5G, with 6G fast approaching), have continued. This has led to a rapid increase in the number of smartphones, tablets, wearables, home robots, and many other IoT devices, resulting in exponential growth of data being generated at the network edge. We have thus entered a new era, that of *multi-device learning*, where we have more data than ever before. If we make good use of this data, we will be able to train more capable, accurate and personalized ML models.

The vast majority of ML algorithms in the literature assume data is stationary and follow a centralized approach. That means that they were designed to be executed on a single machine using static—although sometimes huge—datasets. To apply any of these algorithms in a multi-device setting, making the most of all the available data, we should send local information from all devices to a central server or data center, where we would store them all together and perform the training process. Nevertheless, as we already mentioned in Chapter 1, this raises issues related to scalability (in terms of storage, communications, and computing speed) and privacy. Furthermore, there is a need for algorithms that allow devices to evolve and adapt to changing environments and non-stationary data. Therefore, distributed and continual ML approaches have a much more natural fit in the context of multi-device learning.

2.2 Distributed and federated machine learning

Distributed machine learning (DML) [45, 46] is a ML paradigm that poses the learning process in a decentralized manner, along a network of interconnected machines. Each of them performs a local training using a subset of data. Then, a final model is obtained by aggregating the partial contributions. The local learning can be performed with or without explicit knowledge

of the other participant machines. The first option is the most common in *peer-to-peer* (P2P) architectures (Figure 2.1a), in which there are no hierarchies between the different nodes of the network and each of them can communicate directly with its neighbors. The second one is usual in *client-server* architectures (Figure 2.1b), where the server is a central node in charge of orchestrating the learning process, often managing the computational resources and data available on the other machines, the clients.



(a) Peer-to-peer architecture. **(b)** Client-server architecture.

Figure 2.1: Main communication schemes in distributed machine learning.

We can trace the origins of DML back to the early 1990s. Until then, some advances had been made in research on intelligent distributed systems, addressing problems such as organization, coordination and cooperation. Nevertheless, the problem of distributed learning had been largely ignored. It was Brazdil *et al.* [47] who first discussed the possibility of conducting ML in multi-agent systems in order to enhance robustness and learn faster. Despite being a preliminary work, they already argued that multi-device learning would address some of the fundamental questions of intelligence and learning that can only be understood in this context. They also highlighted communication and cooperation as two unexplored tools in ML that are nevertheless essential for human learning.

Between the 1990s and the early 2010s, several distributed learning methods were proposed. We can refer to them as *traditional* approaches. The main motivation at that time was the need to scale up algorithms to deal with increasingly large datasets and speed up the learning process. Allocating the training among several machines, each of them with a relatively small storage capacity and computing power, proved to be a more affordable alternative than using just one large server with specialized—hence more expensive—hardware [46].

Therefore, algorithms from this period have a strong focus on parallelism. In fact, many of them suggest distributing the data artificially among the different nodes, even when the nature of the problem is not distributed. Data exchange between machines during training is also common. We could roughly classified traditional DML algorithms into two different groups: (1) parallel optimization methods, and (2) distributed ensemble methods.

Parallel optimization methods consist of dividing the training of a single shared model in smaller local sub-problems, each one being solved by a different machine. The vast majority of solutions that fall into this category consist of adaptations of ML algorithms that rely on easily parallelizable optimization methods, like those based on Stochastic Gradient Descent (SGD) [48]. The latter is, indeed, the most widely used. It is frequently employed with neural networks, as well as with other state-of-the-art algorithms such as SVMs. SGD-based methods minimize a loss function defined on the outputs of the model by iteratively adapting its parameters in the direction of the negative gradient. Examples of adaptations of SGD to distributed settings are Parallelized SGD [49], Delayed SGD [50], Dual Averaging Subgradient Method [51], 1-bit Data-parallel SGD [52], or Data-parallel Distributed SGD [53]. Some proposals, instead, rely on other optimization methods, like those based on the augmented Lagrangian, e.g. Distributed Alternating Direction Method of Multipliers (DADMM) [54], or Newton-like techniques, e.g. Distributed Approximate Newton (DANE) [55] or Distributed Self-Concordant Optimization (DiSCO) [56]. Regardless of which optimization method is used, this type of solutions generally requires that the number of machines is much smaller than the number of examples per machine, that the data is distributed across them in IID fashion, and that each node has an identical number of data points. Besides, they are usually synchronous, blocking on the result of each node's iteration before continuing to the next.

Distributed ensemble methods, rather than fitting a single shared model, rely on training a different one on each machine, which are then aggregated using some ensemble learning technique. *Ensemble learning* [57] consists of training multiple *base learners* and combining their outputs to obtain better predictive performance. Each of the learners can use separate parameters or even belong to a different algorithmic family. In general, this kind of methods are easily extensible to distributed environments. Some examples are Distributed Meta-learning [58], Effective Stacking [59], Knowledge Probing [60], Distributed Pasting Votes [61], or Distributed Boosting [62].

In the last decade, motivated by the increasing number of smart devices and the growing awareness of the importance of digital privacy, the perspective has changed. This has led

to a progressive transition from traditional DML to federated learning. The focus is now on multi-device problems, where data are by nature distributed and heterogeneous, but moving them between machines is not an option.

2.2.1 Federated learning

Federated machine learning—federated learning (FL) for short—[6, 7, 21, 22, 63] is a DML framework that allows training a model across multiple client devices, where each of them keeps their own data samples locally, without sharing them with the network. FL pays special attention to data privacy and security, and further decentralizes operations traditionally performed by the server. Thus, the server becomes more like an assistant that coordinates clients to work together, instead of micromanaging the workforce as in traditional DML. Clients, in turn, may have different computing resources and the amount of training data, as well as their distribution, can also vary. Figure 2.2 illustrates the typical FL approach. It consists in alternating between *local updates*, in the devices, and *global aggregations*, in the cloud. Clients and server share a common representation of the model, so the only information exchanged are updates of the model parameters, and never raw data.

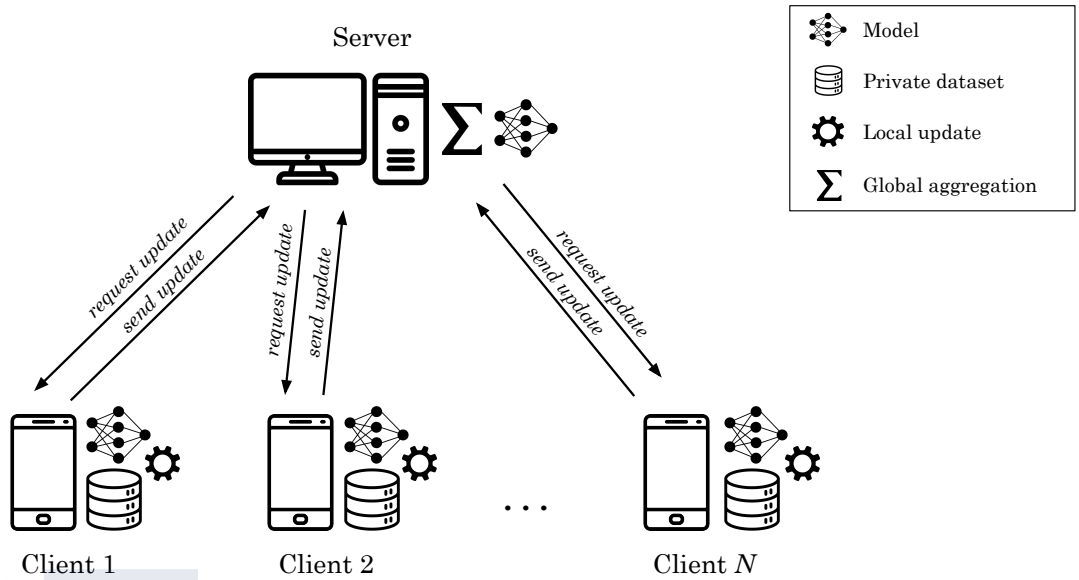


Figure 2.2: Most common federated learning setup.

The concept of federated learning first appeared in 2015, from Jakub Konečný, H. Brendan McMahan, and Daniel Ramage, researchers at Google. They formulated the federated

optimization problem, which has several key properties that differentiate it from traditional distributed optimization [21]:

1. **Statistically heterogeneous**. Local data are usually based on the interaction between the device and its user and the environment, so they will not be representative of the population distribution.

2. **Unbalanced**. Some users will use the service more than others, resulting in different amounts of local training data.

3. **Massively distributed**. The number of clients can be much larger than the average number of data samples per client.

4. **Limited communication**. Devices such as smartphones are frequently offline or on slow or expensive connections.

In that early paper [21], Konečný *et al.* also introduced the first FL algorithm, a federated version of Stochastic Variance Reduced Gradient (SVRG), originally proposed in [64]. Subsequently, in 2016, McMahan *et al.* further developed the method, presenting it under the name of Federated Averaging (FedAvg) [22, 63]. To date, this is still undoubtedly the most relevant method in the FL literature. FedAvg enables federated training of a DNN in a supervised manner. Suppose a multi-device system with N clients, such that $\mathcal{N} = \{1, \ldots, N\}$. We define a model $\mathbf{w} \in \mathbb{R}^P$, where P is the total number of parameters, including weights and biases. The FedAvg algorithm involves R learning rounds, alternating local updates and global aggregations. The process starts on the server side, where the model parameters are randomly initialized and shared with all the clients. Thus, all participants start the process at a common point, which is crucial for convergence purposes. In each round r, a random subset of N' clients, $\mathcal{N}^r \subseteq \mathcal{N}$, is selected. Each device $j \in \mathcal{N}^r$ receives the current parameter vector, $\mathbf{w}_G^{r-1}$, performs SGD on their local dataset, and sends back the updated parameters, $\mathbf{w}_j^r$. Next, the local results are aggregated in the server applying a weighted mean:

$$\mathbf{w}_G^r = \sum_{j=1}^{N} \frac{M_j}{M} \mathbf{w}_j^r, \tag{2.2}$$

where M is the total number of data samples and M_j is the number of samples from client j. Note that the more data a participant uses for local training, the more influence it has on the

Algorithm 2.1: Federated Averaging, server side.

Input : Set of participant clients $\mathcal{N} = \{1, \ldots, N\}$, number of learning rounds R, number of participants per round S, local minibatch size B, number of local epochs E, and learning rate η.
Output : Global model $\mathbf{w}_G$.

Initialize $\mathbf{w}_G^0$
for $r \leftarrow 1$ **to** R **do**
  $\mathcal{N}^r \leftarrow$ `randomSelection`$(\mathcal{N}, N')$ // Randomly choose N' participants from $\mathcal{N}$
  for each participant $j \in \mathcal{N}^r$ parallely **do**
    $\mathbf{w}_j^r \leftarrow$ `localTraining`$_j(\mathbf{w}_G^{r-1}, B, E, \eta)$ // Local update (Algorithm 2.2)
  end
  $\mathbf{w}_G^r \leftarrow \sum_{j=1}^{m} \frac{M^j}{M} \mathbf{w}_j^r$ // Averaging aggregation
end
return $\mathbf{w}_G$

Algorithm 2.2: Federated Averaging, client side (`localTraining`).

Input : Model $\mathbf{w}$, local minibatch size B, number of local epochs E, and learning rate η.
Output : Updated $\mathbf{w}$.

Split the local dataset into a set of batches $\mathcal{B}$ of size B
for epoch $e \leftarrow 1$ **to** E **do**
  for batch $b \in \mathcal{B}$ **do**
    $\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla l(\mathbf{w}; b)$ // ∇l is the gradient of the loss function l on b
  end
end
return $\mathbf{w}$

global model. After the aggregation, a new training round starts. Algorithms 2.1 and 2.2 show the pseudocode of FedAvg for server and clients, respectively.

Since the publication of FedAvg, research on FL has been attracting more and more attention and, in the last few years, the number of publications on this topic has increased exponentially. The main lines of work focus on security enhancement to ensure data protection, and algorithm optimization to improve efficiency and accuracy. Although FL is inherently more privacy-preserving than sharing raw data, the transmission of gradients and model parameters may lead to indirect privacy leakage [65, 66]. Therefore, several protection mechanisms are being explored to provide additional layers of security. At the client side, differential privacy [67, 68] is often introduced, typically by adding random noise to the data, allowing each participant to have a personalized privacy budget. Regarding global aggregation, there are

strategies based on Secure Multi-Party Computation (SMC) [69, 70], where communications are secured and protected with cryptographic methods. One example of this is homomorphic encryption [71], which allows operations to be performed directly on encrypted data and models without the need to decrypt them. Many researchers are also interested in other security issues, such as preventing poisoning attacks [72, 73] or free-riding [74].

In terms of algorithm optimization, improving communication cost and coping with statistical heterogeneity are the main issues currently faced by researchers. By far, the key bottleneck of FL has been the difficulty of decreasing communication overhead during training [75]. To tackle with large amounts of data and make FL scalable, a number of options have been explored that seek to optimize network bandwidth [76, 77], reduce communication rounds [78, 79], and improve model update speed [80, 81]. Other works focus on developing methods that are robust to transmission delays and outage constraints [14, 82].

Statistical heterogeneity of the data is a major issue in multi-device problems. Traditional ML approaches, implicitly or explicitly, generally assume data is independent and identically distributed (IID). That means that all data samples are independent events taken from the same probability distribution. This assumption can be suitable when the entire training dataset is in a single place and a centralized method is applied. However, in federated settings, data are collected and processed in various devices in a distributed manner. Each device will interact with its user and with the environment in a particular way, thus gathering biased data. In such situations, we speak of *non independent and identically distributed (non-IID)* or *heterogeneous* data. Statistical discrepancies between clients can compromise the performance and even the convergence of the model. Some authors have evaluated FedAvg on certain non-IID scenarios [17, 22], showing that it still converges to high accuracies, though taking more rounds than in IID settings. Li *et al.* [83] propose FedProx, a generalization of FedAvg that adds a proximal term during optimization, thus improving stability during training and providing further convergence guarantees. Some other authors explore different personalization techniques [84–86], trying to balance the general knowledge common to all clients with the specific one of each of them. Beyond the differences between clients, statistical heterogeneity can also be caused by changes over time. User behavior and the environment may evolve in different ways. Hence, ideally devices should constantly extract, process and learn from new data. This line of research is still largely unexplored and there is a need for strategies that allow for continual adaptation (see Section 2.3).

Literature on federated learning is closely related to deep learning. The vast majority of FL methods that have been proposed in recent years are designed to collaboratively train a DNN in a supervised manner. Nevertheless, some papers explore alternative approaches. For instance, Bakopoulou *et al.* [87] propose a federated SVM with linear kernels, and Ludwig *et al.* [88] implement a federated Iterative Dichotomiser 3 (ID3) decision tree that grows at the server side while the clients only perform light computations. There are also some recent works that already consider unsupervised FL. For example, Soliman *et al.* [89] introduce a federated k-means to address decentralized clustering by integrating HyperLogLog counters with a P2P architecture. Servetnyk *et al.* [90], instead, perform clustering employing sophisticated techniques such as self-organizing maps to determine the appropriate weights for the weighted aggregation. Finally, some authors are starting to delve into reinforcement FL, such as Fan *et al.* [91], who present a federated reinforcement learning framework and provides theoretical guarantee in the potential presence of faulty agents.

Based on how training data are distributed, Yang *et al.* [75] propose to distinguish between three categories of federated methods: (i) horizontal, (ii) vertical, and (iii) transfer-based. In horizontal federated learning (HFL) settings, all clients share the same feature space but have different sample spaces, i.e., each participant has different local instances in their dataset, but they all have a common representation of the problem. This is the most common scenario in the literature. For example, in 2018 Google proposed a HFL solution to improve the text prediction model of Android devices' keyboards by learning from real user conversations [8]. In vertical federated learning (VFL), instead, clients share the sample space, but with different features. This means that participants have an overlap on data samples, but each one has partial information about them. Finally, federated transfer learning (FTL) is applicable for the case of no or minimal overlap in data samples and features. It consist of applying transfer learning [92] techniques to find a common representation between the different feature spaces under a federation.

Despite being a fairly recent learning paradigm, federated learning has already being used in several applications. As we mentioned above, the first use case came from Google in 2018, which applied FL to improve predictive text in Android's keyboard [8]. This has been followed by further improvements in text and also emoji prediction [93, 94]. Also in the context of smartphones, proposals have been developed for user mobility estimation [95], and activity recognition [96–98]. As a disruptive approach to preserving data privacy, FL has great prospects in healthcare as well. In this way, it has started to be applied in tasks

such as hospitalization and mortality prediction [11, 99], biomedical imaging analysis [100, 101], patient similarity learning [102], or computational phenotyping [103]. Finally, it is worth mentioning some other examples of applications in the industry, such as energy demand prediction [10], autonomous navigation [12, 104, 105], or malware detection [9].

2.3 Continual learning and concept drift

Continual machine learning—or just *continual learning* (CL)—[106–109] is a ML paradigm focused on building adaptive models over time. This approach seeks to smoothly update the predictor to take into account different data distributions and tasks but still being able to re-use and retain useful knowledge over time. It is highly inspired by the human learning process, as people acquire skills in numerous tasks over their lifespan, making use of past knowledge to learn about new concepts without forgetting the previous ones. CL has also been referred to in the literature as *lifelong learning* [110–112], *never-ending learning* [113, 114], and *incremental learning* [109, 115]. In this PhD thesis, we adopt the term "continual learning", since it is the most widely used nowadays.

The CL paradigm began to be explored in the field of robotics in the mid-1990s [110, 116]. This early work focused on reinforcement learning strategies to enable robots to capture invariant knowledge about the different tasks and environments. Since then, attention has been devoted to the topic within the supervised, unsupervised, semi-supervised, and reinforcement learning domains. Supervised CL was first studied by Thrun [117], who explored the problem of learning on a sequence of classification tasks. This was followed by other techniques, mainly based on neural networks [115, 118–120]. Regarding unsupervised learning, there are some proposals for continual topic modeling [111, 121] and information extraction [122, 123]. As for semi-supervised methods, probably the greatest exponent is the Never-Ending Language Learner (NELL) system [113, 114], which has been continuously reading the Web since 2010 for information extraction. Finally, it is in the area of reinforcement CL that the greatest number of contributions have been made, with special attention to robotics [112, 124–127].

Continual learning has been successfully implemented in a variety of contexts and applications. Autonomous robotics and human-machine interaction are inherently continuous, since they are open-ended. Thus, there have been several proposals in autonomous control [120], service robotics [128], computer vision [129], or autonomous driving [130]. CL is also

present in data analytics and big data processing, including domains such as data visualization [123], image and video processing [131, 132], automated annotation [133], and outlier detection [134]. It is important to note that, although many works are oriented to robotics and thus to naturally distributed settings, CL has traditionally been addressed in a centralized manner.

In CL settings, we do not work with static datasets, but with data streams. A *data stream* $\mathcal{D}$ is a potentially unbounded sequence of data arriving over time [135]. The stream follows a sequence of unknown distributions $\mathcal{D} \sim (D^1, \ldots, D^L)$. In addition, it can respond to one or more tasks $(t_1, \ldots, t_T)$, such that $\mathcal{D} = (\mathcal{D}^{t_1}, \ldots, \mathcal{D}^{t_T})$, where $\mathcal{D}^{t_i}$ is the subset of samples associated with task t_i. A *task* is a learning experience characterized by a unique target function. For instance, in a supervised context, one task might be to distinguish dogs from cats and another could be cars versus motorcycles. It is important to note that the tasks are just an abstract representation that helps to split the full learning experience into smaller learning pieces. However, there is not a necessarily bijective correspondence between data distributions and tasks.

CL algorithms have to learn gradually on the data stream, which involves dealing with two conflicting objectives: retaining previously learned knowledge that is still useful while replacing the one no longer relevant with current information. This is usually known as the *stability-plasticity dilemma* [136]. In practice, it translates into two main challenges: avoiding catastrophic forgetting and tackling concept drift. *Catastrophic forgetting* [137] is a phenomenon that consists in the inability to retain old information in the presence of new one, resulting in the degradation of the model performance. It is common when training neural networks on a sequence of tasks. *Concept drift* [20], on the contrary, is a problem that arises when a single task is learned, but the data stream has unpredictable distributional shifts. As a result, the model performance tends to drop dramatically. Concept drift management can also involve preventing catastrophic forgetting. When working with several tasks, all of them are usually known in advance. However, when working on a single, non-stationary task, it is usually not possible to anticipate changes in distribution, so explicit detection mechanisms for these distributional shifts are needed. Thus, concept drift can be very challenging.

2.3.1 Concept drift

Concept drift has been extensively studied in the supervised learning literature [20, 138–141]. It was first proposed by Schlimmer and Granger [142], who aimed to point out that noisy data

may become non-noisy information at a different time. These changes might be caused by shifts in hidden variables which cannot be measured directly. Formally, we can define concept drift as follows:

Definition 2.2 *Let* $[0,t]$ *be a time period, and* $\mathcal{D}^{0,t} = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=0}^{M}$ *a set of samples of size M, where* $\mathbf{x}_i$ *is the feature vector and* $\mathbf{y}_i$ *is its correspondent output.* $\mathcal{D}^{0,t}$ *follows a certain probability distribution* $D^t(\mathbf{x},\mathbf{y})$*, estimated by the joint probability density function* $P^t(\mathbf{x},\mathbf{y})$*. We say a* concept drift *occurs at timestamp* $t+1$ *if there is a significant difference between* $D^t(\mathbf{x},\mathbf{y})$ *and* $D^{t+1}(\mathbf{x},\mathbf{y})$*, denoted as:*

$$\exists\, t: \quad D^t(\mathbf{x},\mathbf{y}) \nsim D^{t+1}(\mathbf{x},\mathbf{y}), \quad P^t(\mathbf{x},\mathbf{y}) \neq P^{t+1}(\mathbf{x},\mathbf{y}). \tag{2.3}$$

The joint probability can be factorized in two parts: $P(\mathbf{x},\mathbf{y}) = P(\mathbf{x}) \cdot P(\mathbf{y}|\mathbf{x})$. Taking this into account, concept drift is usually categorized into three different types, according to which factor of the equation is altered [20, 139]: (1) virtual, (2) real, and (3) total. *Virtual concept drift* makes reference to variations in the input marginal probability density function, so that $P^t(\mathbf{x}) \neq P^{t+1}(\mathbf{x})$ and $P^t(\mathbf{y}|\mathbf{x}) = P^{t+1}(\mathbf{y}|\mathbf{x})$. On the other hand, *real concept drift* is caused by a change in the conditional probability of the labels with respect to the input features, $P^t(\mathbf{x}) = P^{t+1}(\mathbf{x})$ and $P^t(\mathbf{y}|\mathbf{x}) \neq P^{t+1}(\mathbf{y}|\mathbf{x})$. Finally, *Total concept drift* is the combination of the two previous cases, $P^t(\mathbf{x}) \neq P^{t+1}(\mathbf{x})$ and $P^t(\mathbf{y}|\mathbf{x}) \neq P^{t+1}(\mathbf{y}|\mathbf{x})$. Figure 2.3 represents the three concept drift scenarios in a binary classification problem with a two-dimensional input space.

Consider a smartphone application for elderly health monitoring. Our software integrates an activity recognition model that predicts when the user is walking based on the inertial signal recorded by the accelerometer of the device. The model is adjusted to detect average human walking patterns. An elderly woman, user of this system, always carries her smartphone in her back trouser pocket and the model is able to identify perfectly when she is walking. However, suppose that summer arrives and the woman begins to wear skirt instead of trousers. Since the skirt has no pockets, she starts to carry the smartphone inside a handbag. This causes the signal registered now by the accelerometer to be totally different from the one recorded inside the pocket. Note that this happens although the woman keeps walking exactly as before. This would be an example of virtual drift. Suppose now a different situation in which, unfortunately, one day the woman suffers a fall and sprains her ankle. She is now unable to walk properly,

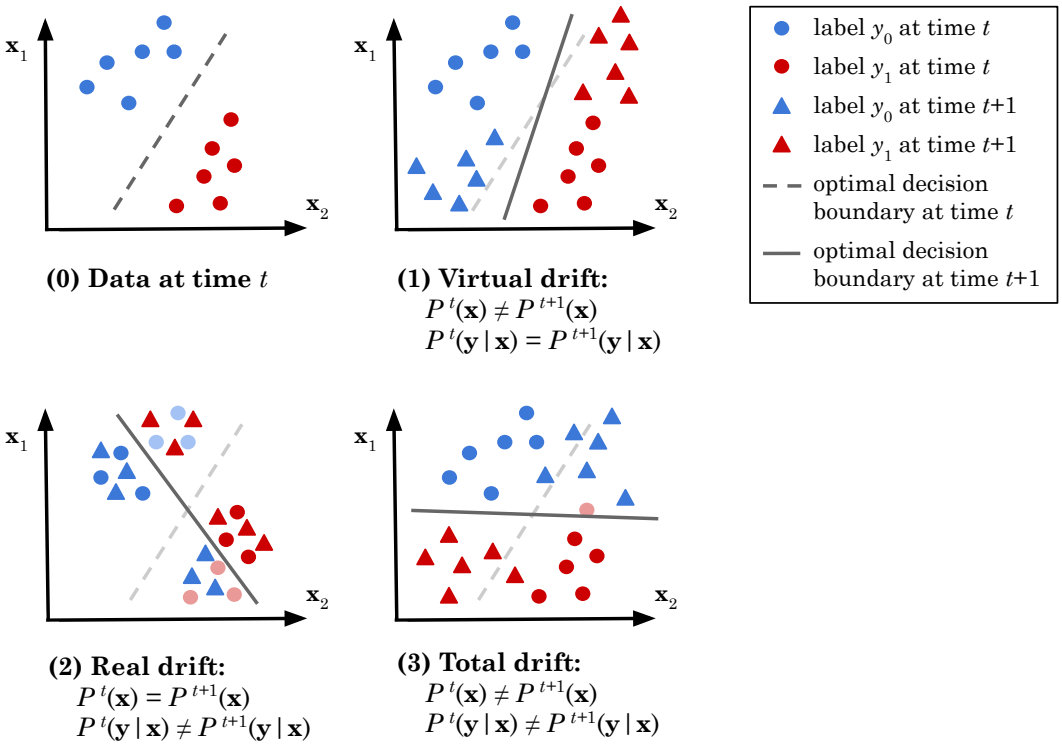


Figure 2.3: The three types of concept drift in a a two-dimensional classification problem.

so she moves around on crutches and at a much slower pace. Although she still carries the smartphone in the pocket, her behavior has changed. The model, which had just been trained to detect average walking, classifies her movements as "standing still". This would be a real drift. If we would combine both situations, so that both the walking manner and the signal perceived by the accelerometer were different, we would be talking about a total drift.

Concept drift can be also classified according to how the new joint distribution is different from the previous one. In this sense, four types are differentiated [139, 140]: (i) sudden, (ii) recurring, (iii) gradual, and (iv) incremental. We say a drift is *sudden* if there is a timestamp that separates the old concept from the new one. This process can happen repeatedly and even go back to the original concept, and in that case we say it is a *recurring* drift. On the contrary, if the data from the new concept arises intertwined with the old concept, we name it *gradual* concept drift. Lastly, *incremental* drift takes place when data shifts smoothly between the concepts, and therefore the drift cannot be detected in a single instant but within a window of consecutive timestamps.

Research on concept drift can be broadly divided into two main sub-problems: (1) drift detection, and (2) drift adaptation. *Drift detection* consists of determining whether or not

concept drift occurs, identifying change points in data distribution. There are many drift detection algorithms in the literature. Depending on which information they rely on, we can distinguish between data distribution-based and error rate-based techniques. *Data distribution-based* methods [143–146] use a distance function or metric to quantify the dissimilarity between the distribution of historical and new input data. When this dissimilarity is statistically significant, it is considered that a drift has occurred. Since this kind of algorithms only require the input feature vectors, they are well suited for virtual drift detection in contexts where access to true labels is limited. On the other hand, *error rate-based* methods [147–149] are based on tracking changes in the online error of the model. When the error model increases abruptly, a concept drift is detected. The main drawback of this kind of approaches with respect to data-distribution based ones is that they need to know the actual labels of the patterns in order to estimate the model error. In return, they allow to detect real concept drift.

Drift adaptation focuses on addressing the change, updating existing models accordingly so that performance is not compromised. The three main approaches for drift adaptation are simple retraining, ensemble learning, and model adjusting. *Simple retraining* [150, 151] is the most naive and straightforward strategy, since it consists of just replacing the obsolete model with a new one, trained from scratch on the latest data. *Ensemble learning* [152–154] allows to combine several learners in an ensemble. Thus, when a drift is detected, a new model is trained using the new data and it is added to the ensemble, that also preserves the old learners. Finally, *model adjusting* [106, 155, 156] consists of further training of the existing model, so that it adaptatively learns from the changing data by partially updating itself. This is arguably the most efficient approach when drift only occurs in local regions. However, online model adjusting is not straightforward and it will depend on the specific learning algorithm being used.

CL algorithms that deal with concept drift can also be categorized according to other criteria [139, 141]:

- Depending on whether or not they perform explicit drift detection, we distinguish between *active* and *passive* methods. Active drift detection involves observing the stream to search for changes and determine whether and when a drift occurs, so that the model is updated only when a drift is detected. Instead, passive drift detection considers that changes may occur constantly or occasionally, and therefore the learner is continually updated as data arrive.

- Given the number of learners employed during classification, we find *single classifier* and *ensemble-based* approaches. Single classifiers use just one learner to make predictions, whereas ensemble-based techniques combine the results of a set of learners.
- Lastly, determined by the amount of data considered during training, there are *online* and *batch* algorithms. Online approaches update the model instance by instance. In contrast, batch methods wait until a representative amount of data is collected for training.

Online proposals are usually passive and rely on a single classifier, while batch approaches tend to be active and either based on a single learner or ensembles.

Chapter 3

Concept-Drift-Aware Federated Averaging

Throughout Chapters 1 and 2, we presented and reviewed federated learning. We explained why it has become the most appealing way to implement multi-device learning, considering its advantages in terms of digital privacy and scalability. We also discussed the importance of developing methods that allow continual learning on data streams that may be non-stationary, and we introduced the phenomenon of concept drift (Section 2.3). In the present chapter, we make a first approach towards *continual federated learning* (CFL). We propose a new method, called Concept-Drift-Aware Federated Averaging (CDA-FedAvg), which extends the popular FedAvg algorithm, enhancing it for continual learning under concept drift. Our method includes virtual drift detection and adaptation mechanisms, allowing it to deal with supervised scenarios where all client devices share a common goal, but the underlying joint distribution of data evolves over time. We empirically show the weaknesses of regular FedAvg in this kind of continual setting and a remarkable improvement in the learning ability of CDA-FedAvg over FedAvg. We also prove a reduction in storage, communication, and computational costs in the devices.

> [98] F. E. Casado*, D. Lema*, M. F. Criado*, R. Iglesias*, C. V. Regueiro†, and S. Barro*. "Concept drift detection and adaptation for federated and continual learning," *Multimedia Tools and Applications*, Springer, vol. 81, no. 3, pp. 3397–3419, 2021. DOI: 10.1007/s11042-021-11219-x. ISSN: 1380-7501.

The rest of the chapter is organized as follows: Section 3.1 reviews the state of the art on CFL. Section 3.2 provides a formal definition of concept drift in CFL settings. In Section 3.3, CDA-FedAvg is introduced. Section 3.4 presents a theoretical analysis of the comptational complexity of our proposal. Section 3.5 provides some thoughts on the benefits of drift management in CFL. Finally, Section 3.6 presents the experimental results.

3.1 Towards continual federated learning: State of the art

As we could see in Section 2.3, research on CL and concept drift has received considerable attention in recent decades [20, 106, 107]. Despite this, it has traditionally been approached in a centralized, single-learner fashion, and there is very little work dealing with continual and federated settings at the same time. This is not surprising given how young the FL paradigm is. We have already mentioned that at the beginning of this PhD project, in 2018, there were only a few works in FL. The papers from which the contents of this chapter are drawn [97, 98] were early pioneers in CFL. More recently, since 2021, other authors have been publishing new algorithms in similar directions, which highlights the importance and growing interest in this type of learning strategies. The following is an updated review of these works.

Yoon *et al.* [157] propose a method for inter-client knowledge transfer, called Federated Continual Learning with Weighted Inter-client Transfer (FedWeIT). They pose a continual and federated scenario where each client learns on a sequence of tasks. FedWeIT decomposes the model weights into two separated sets: globally shared parameters and sparse task-specific parameters. Hence, they allow each participant to adaptively train, benefiting from the knowledge of other clients that share common tasks. They validate their approach on several settings and datasets. When comparing with existing baselines, they show significantly higher accuracy

*Centro Singular de Investigación en Tecnoloxías Intelixentes (CiTIUS), Universidade de Santiago de Compostela, 15782 Santiago de Compostela, Spain.
†CITIC, Computer Architecture Group, Universidade da Coruña, 15071 A Coruña, Spain.

and reduced communication costs. The major weakness of the proposal is that it approaches CL in a very naive way, not very applicable to real problems. The training is addressed in a classical manner, prefixing a number of communication rounds and assuming that the different tasks are known in advance. Concept drift problem is not considered.

Usmanova *et al.* [158] introduce a method call Federated Learning without Forgetting - 2 Teachers (FLwF-2T). The authors focus on federated incremental multi-task scenarios. They propose to use knowledge distillation to transfer general knowledge from the server and minimize forgetting when a client learns a new task. Knowledge distillation was originally proposed to transfer the knowledge from a large model (teacher) to a smaller one (student). In this case, the authors suggest to use two teachers: the past model of the client and the global model from the server. They evaluate FLwF-2T on the task of human activity recognition, showing promising results. As in the previous work, the tasks are known in advance and the number of communication rounds is prefixed. In addition, it is assumed that the clients work synchronously and that, at each moment, each one deals with a different task, so that the server always receives updates on all tasks and has a complete view of the problem. All this implies serious limitations for implementation in real applications. Concept drift detection and adaptation are also not addressed.

Park *et al.* [159] present Federated Incremental Learning with Variational Embeddings Rehearshal (FIL-VER), an algorithm designed to address incremental single-task FL problems. FIL-VER follows a passive approach, based on applying rehearsal in order to incrementally learn a task while preventing forgetting. For that, the authors use variational embedding encoders. They validate the method on a new dataset created from different random transformations of the original MNIST dataset. The algorithm shows parity with offline training in different scenarios subject to concept drift and where clients can drop in or out dynamically. Nevertheless, there is no explicit drift detection, but rehearsal is applied all the time regardless of whether a change happened or not. This could be improved to be more efficient, knowing exactly when training is necessary and which data to use. Besides, the method requires having a pre-trained encoder for rehearsal, which would be an issue in many real-world problems.

Canonaco *et al.* [160] propose Adaptive-FedAvg, an extension of the original FedAvg method, adapting it to be robust to concept drift in non-stationary single-task problems. Adaptive-FedAvg implements a server-side passive approach, in which the server increases the learning rate to be used by the clients based on the variability between consecutive model updates. To this end, they estimate the variance of the global parameters through an

exponential moving average. Thanks to the adaptive learning rate, Adaptive-FedAvg is meant to guarantee the decaying property required for convergence while providing a more reactive behavior in presence of concept drifts. Authors carried out several experiments using two image-classification benchmarks, showing good results in both stationary and non-stationary conditions. As in the previous work, there is no active management of concept drift.

Jothimurugesan *et al.* [161] also focus on continual single-task and federated scenarios subject to concept drift. As opposed to the previous works, they do integrate active drift detection and adaptation mechanisms. Their strategy aims to create one model for each new concept, so that all clients under the same concept can train that model collaboratively, similar to what is done in personalized FL. They follow an error rate-based approach for drift detection and pose drift adaptation as a time-varying clustering problem, introducing two new algorithms for model creation and client clustering. The first one, FedDrift-Eager, is a specialized algorithm that creates models based on drift detection. It is effective if new concepts are introduced one at a time. The second, FedDrift, is a general algorithm that leverages hierarchical clustering to adaptively determine the appropriate number of models. FedDrift isolates drifted clients and conservatively merges clients via hierarchical clustering, so that it can effectively handle general cases where an unknown number of new concepts emerge simultaneously. Authors tested both methods on 4 different datasets, showing significantly higher accuracy than existing baselines, and comparable to an oracle that knows the ground-truth clustering of clients to concepts at each time step.

The method we describe in the present chapter, CDA-FedAvg [98], was the first to provide active drift detection and adaptation in continual and federated settings. We believe that this is the best way to deal with distributed and non-stationary data since, as we will see later, it is more efficient than passive approaches in terms of communications and computational needs. In short, we can say that active drift management helps us to determine exactly *what* to learn and *when* to learn it. As we have just discussed, the work of Jothimurugesan *et al.* [161] has recently followed a very similar direction. To the best of our knowledge, there are no other proposals in the literature for continual and federated learning with explicit drift detection and adaptation.

3.2 Concept drift in continual federated settings

In this section, we will provide a formal definition of concept drift for CFL. In the previous chapter, we already described the concept drift problem in detail (Section 2.3.1, Definition 2.2). Nevertheless, in a non-stationary setting with multiple devices, each of them has its own data stream, which makes the problem more complex. If we want to train a global model in a federated manner, we have to take into account that each local data stream may change differently over time.

Formally, we can define *local concept drift* at client level as follows:

Definition 3.1 *Let $[0,t]$ be a time period, $\mathcal{N} = \{1,\dots,N\}$ a set of clients of size N, and $\mathcal{D}_j^{0,t} = \{(\mathbf{x}_i,\mathbf{y}_i)\}_{i=0}^{M_j^t}$ the local dataset of client $j \in \mathcal{N}$, of size M_j^t, where $\mathbf{x}_i$ is the feature vector and $\mathbf{y}_i$ is its correspondent output. Each local dataset $\mathcal{D}_j^{0,t}$ follows a certain probability distribution $D_j^t(\mathbf{x},\mathbf{y})$, estimated by the joint probability density function $P_j^t(\mathbf{x},\mathbf{y})$. We say a* local concept drift *occurs at timestamp $t+1$ for client j if there is a significant difference between $D_j^t(\mathbf{x},\mathbf{y})$ and $D_j^{t+1}(\mathbf{x},\mathbf{y})$, denoted as:*

$$\exists\, t, j: \quad D_j^t(\mathbf{x},\mathbf{y}) \nsim D_j^{t+1}(\mathbf{x},\mathbf{y}), \quad P_j^t(\mathbf{x},\mathbf{y}) \neq P_j^{t+1}(\mathbf{x},\mathbf{y}). \tag{3.1}$$

Nevertheless, a change in the local distribution of one client does not necessarily means a change in the global joint distribution. For instance, imagine that all clients except client j experience a new concept for the first time at timestamp t. Later, at time $t+k$, that change occurs for client j. At time t the global data distribution has been altered. However, at $t+k$ it remains the same, and only the local distribution of client j changes. In that situation, the federated model will not be affected by the local change. Therefore, we must differentiate local concept drift from *global concept drift*:

Definition 3.2 *Let $[0,t]$ be a time period, $\mathcal{N} = \{1,\dots,N\}$ a set of clients, and $\mathcal{D}_j^{0,t} = \{(\mathbf{x}_i,\mathbf{y}_i)\}_{i=0}^{M_j^t}$ the local dataset of client $j \in \mathcal{N}$, of size M_j^t, that follows a certain probability distribution $D_j^t(\mathbf{x},\mathbf{y})$. Let $M^t = \sum_{j=1}^{N} M_j^t$, so that $D_G^t(\mathbf{x},\mathbf{y}) = \sum_{j=1}^{N} \frac{M_j^t}{M^t} D_j^t(\mathbf{x},\mathbf{y})$ is the global data distribution, with associated probability density function $P_G^t(\mathbf{x},\mathbf{y})$ We say a* global concept drift *occurs at timestamp $t+1$ if:*

$$\exists\, t: \quad D_G^t(\mathbf{x},\mathbf{y}) \nsim D_G^{t+1}(\mathbf{x},\mathbf{y}), \quad P_G^t(\mathbf{x},\mathbf{y}) \neq P_G^{t+1}(\mathbf{x},\mathbf{y}). \tag{3.2}$$

The above definitions apply to both real and virtual drifts. Note that a global concept drift implies that a local concept drift has occurred in at least one client. Note also that a global drift will always have an impact on the performance of the federated model, while a local one may not. As we will show in Section 3.6, naive learning using state-of-the-art algorithms such as FedAvg in continual federated scenarios can lead to forgetting and considerable decrease in model performance when concept drift occurs. In order to avoid that, we propose to introduce drift detection and adaptation mechanisms during training, working at both local and global levels.

3.3 The CDA-FedAvg method

We present Concept-Drift-Aware Federated Averaging (CDA-FedAvg), a federated method able to learn in continual and non-stationary environments, being robust to concept drift. CDA-FedAvg can be considered an extension of the original FedAvg, enhanced with drift detection and adaptation mechanisms. As we will see below, our proposal specializes in dealing with virtual concept drift in supervised classification tasks. Drift detection is carried out locally, by each client, looking for significant variance in local distribution (see Section 3.3.1). To adapt to change, we follow a rehearsal approach, conducting new local updates which will then be aggregated on the server side (Section 3.3.2). Thanks to active drift management, each client has enough autonomy to decide when to train and what data to use for that purpose, which reduces the responsibilities and workload of the server. Algorithms 3.1 and 3.2 detail the pseudocode of CDA-FedAvg on the server and client sides, respectively.

As we can see in Algorithm 3.1, the server starts by initializing the global model and communicating it to all the participants (lines 1–2). Then, it will periodically check if there has been any local update on one or more clients (lines 4–5). If at least S local updates are received from N' different clients, then a global aggregation is performed (line 6). Each time the model is globally consensuated, the server will have to broadcast it again to all the participants so that they always have the latest version of the model (line 7).

Notice that, following this approach, it is possible for the clients to send updates at any time, giving room to different participation rates among them. Hence, the global model could be better fitted to the particularities of the most active participants. The choice of the parameter

Algorithm 3.1: CDA-FedAvg, server side.

Input : List of participant clients $\mathcal{N} = \{1, 2, \ldots, N\}$, minimum number of local updates to perform aggregation N'.
Output : Global model $\mathbf{w}_G$.

1 Initialize $\mathbf{w}_G^0$
2 `broadcast`$(\mathbf{w}_G^0, \mathcal{N})$ // Send the initial parameters to all clients
3 **while** *true** **do**
4 Listen for local updates $\mathbf{w}_j^t,\ \forall j \in \mathcal{N}$
5 **if** $\exists\{j_1, \ldots, j_{N'}\} \in \mathcal{N}$: *new updates* $\{\mathbf{w}_{j_1}^t, \ldots, \mathbf{w}_{j_{N'}}^t\}$ *are received* **then**
6 $\mathbf{w}_G^t \leftarrow \sum_{j=j_1}^{j_{N'}} \frac{M_j^t}{M^t} \mathbf{w}_j^t$ // Averaging aggregation
7 `broadcast`$(\mathbf{w}_G^t, \mathcal{N})$
8 **end**
9 **end**

* Given the continual nature, we leave the choice of the stop criteria as a matter of implementation.

N' (the minimum number of local updates required to perform a global aggregation) plays a fundamental role in preventing overfitting to these clients.

Clients are responsible for learning the task locally, managing, if necessary, concept drift. To that end, each client will continually acquire new data from its environment. This data will be processed to identify new concepts (drift detection) and learn from them (drift adaptation). We propose that each client handles two different data storages: a *short-term memory* and a *long-term memory*. The short-term memory, $\mathcal{Q}$, is used to store the data instances the device has acquired in the last time interval. This recent data is kept for a limited amount of time and is processed to check whether a drift occurs. The long-term memory, $\mathcal{L}$, will store a representative amount of data from each of the concepts seen so far. This information will be kept for a long time and will be used to train and retrain the model locally, so that new concepts are learned without forgetting the previous ones.

Each client operates as follows (Algorithm 3.2): At the beginning of the process, both short-term and long-term memories are empty (lines 1–2), and the model has never been trained locally. Thus, the first data acquired by each client will automatically belong to the initial concept. This data will be stored in the long-term memory and used to perform the first local update (line 3). After that, each client continues to acquire new data, saving it in the short-term memory and processing it to check whether a drift is detected (lines 5–13). Only when the drift detection algorithm confirms the shift (line 14), new data related to the new concept will be stored in the long-term memory and further training rounds will be

carried out (line 16). In the following two subsections, we will discuss in more detail the two fundamental parts of our method on the client side: drift detection (Section 3.3.1) and drift adaptation (Section 3.3.2).

Algorithm 3.2: CDA-FedAvg, client side.

Input : Reference to the latest global model $\mathbf{w}_G$, minimum amount of data to train L, local minibatch size B, number of rounds R per change, number of local epochs E per round, learning rate η, sensitivity to change λ, padding Δ, and maximum size Q_{max} for the sliding window.

Output : None.

1 $Q \leftarrow \{\varnothing\}$ // Initialize the sliding window (short-term memory)
2 $\mathcal{L} \leftarrow \{\varnothing\}$ // Initialize the long-term memory
3 $\mathcal{L} \leftarrow$ `localUpdate`$(\mathbf{w}_G, \mathcal{L}, L, B, R, E, \eta)$ // Learn first concept (Algorithm 3.4)
4 **while** *true** **do**
5 **if** *new input pattern,* $\mathbf{x}_i$*, is observed* **then**
6 $(\hat{\mathbf{y}}_i, q_i) \leftarrow$ `predict`$(\mathbf{w}_G, \mathbf{x}_i)$ // Classify the pattern
7 $Q \leftarrow Q \cup q_i$ // Add the confidence into Q
8 **if** $|Q| >= Q_{max}$ **then**
9 $Q \leftarrow Q \setminus \{q_1\}$ // Remove the oldest element in Q
10 **end**
11 $z \leftarrow$ `random`$(0, 1)$ // Generate random number in the interval [0,1]
12 **if** $e^{-2q_i} \geq z$ **then**
13 $d \leftarrow$ `driftDetection`$(Q, \lambda, \Delta, Q_{max})$ // Check for drift (Algorithm 3.3)
14 **if** *d is true* **then**
15 $Q \leftarrow \{\varnothing\}$
16 $\mathcal{L} \leftarrow$ `localUpdate`$(\mathbf{w}_G, \mathcal{L}, L, B, R, E, \eta)$ // Update (Alg. 3.4)
17 **end**
18 **end**
19 **end**
20 **end**

* Given the continual nature, we leave the choice of the stop criteria as a matter of implementation.

3.3.1 Drift detection

In Section 2.3.1 we explained that drift detection algorithms generally fall into two categories, those based on the data distribution and those based on the error rate. Algorithms in the first group use a distance function or metric to quantify the dissimilarity between the distribution of historical and new data. Those in the second group focus on tracking changes in the online error rate of the model. In this work we decided to use a data distribution-based algorithm.

Our choice is motivated by the capability of this type of technique to detect virtual concept drift without relying on knowing the true labels of the data samples.

We propose a CUSUM-type detection method that works with *beta* distributions [162, 163]. For each input pattern $\mathbf{x}_i$, we estimate a metric that helps us to quantify the dissimilarity between the old and the new data distributions. Any of the metrics that have been proposed in the literature can be used [139]. We chose the confidence of the classifier because our experimental evaluation is carried out on a classification task (Section 3.6). We define the *confidence* of the model on a prediction as the classifier's maximal conditional posterior probability, i.e., the probability $P(c_k|\mathbf{x}) \in (0,1]$ of a class c_k from one of the C possible classes $\mathscr{C} = \{c_1, c_2, \ldots, c_C\}$ to be the correct class for an instance $\mathbf{x}$. We use the most recent version of the global federated model to predict a probabilistic output $\hat{\mathbf{y}}_i$ and confidence q_i associated to each pattern $\mathbf{x}_i$ (line 6, Alg. 3.2). The confidence q_i is then stored in the short-term memory of the client, Q (line 7, Alg. 3.2). This short-term memory works as a sliding window of length $Q = |Q|$ and maximum size Q_{max}. Once this maximum size is reached, adding a new element to Q implies deleting the oldest one (lines 8–10, Alg. 3.2). Following this approach, we are able to detect sudden, recurring and gradual drifts, and even incremental ones if the sliding window is big enough to cover the change interval. Therefore, the value of Q_{max} will be strongly dependent on the task to be solved.

The core of our detection method is detailed in Algorithm 3.3, which is called by Algorithm 3.2 in line 13. Algorithm 3.3 aims to identify changes in the distribution of the confidences stored in Q. For that, the first thing we do is to split Q into two sub-windows (lines 5–6). Let Q_a and Q_b be the two sub-windows, where Q_a contains the most recent confidences. Each sub-window is required to contain at least Δ elements to maintain the statistical properties of a distribution. When a concept drift occurs, it is expected that confidence scores will decrease. Therefore, we only need to detect changes in the negative direction. Namely, if m_a and m_b are the mean values of the confidences in Q_a and Q_b respectively, a change point is searched only if $m_a \leq (1-\lambda) \times m_b$, where λ is the sensitivity to change (line 7). We propose to set $\lambda = 0.05$ and $\Delta = 100$, which are widely used values in the literature [163]. We also suggest that Q_{max} is given by Δ, so that $Q_{max} = 20\Delta$.

The confidence values in each sub-window (Q_a and Q_b) are expected to follow two different *beta* distributions. However, the actual parameters for each one are unknown. Algorithm 3.3 estimates these parameters at lines 9 and 10, given the mean and the variance of each sub-window, by using the method of moments [164]. Next, the sum of the log likelihood ratios s_k

Algorithm 3.3: Drift detection method (`driftDetection`).

Input : Sliding window Q, sensitivity to change λ, padding Δ, and maximum size Q_{max} for the sliding window.
Output : Boolean indicating whether a drift is detected or not.

1 $s_f \leftarrow 0$
2 $T_h \leftarrow -\log(\lambda)$
3 $Q \leftarrow |Q|$
4 **for** $k \leftarrow \Delta$ **to** $Q - \Delta$ **do**
5 $\quad m_b \leftarrow \texttt{mean}(q_1 : q_k \in Q)$
6 $\quad m_a \leftarrow \texttt{mean}(q_{k+1} : q_Q \in Q)$
7 $\quad$ **if** $m_a \leq (1-\lambda) \cdot m_b$ **then**
8 $\quad\quad s_k \leftarrow 0$
9 $\quad\quad (\hat{\alpha}_b, \hat{\beta}_b) \leftarrow \texttt{estimateParams}(q_1 : q_k)$ // Get parameters of beta distribution
10 $\quad\quad (\hat{\alpha}_a, \hat{\beta}_a) \leftarrow \texttt{estimateParams}(q_{k+1} : q_Q)$
11 $\quad\quad$ **for** $i \leftarrow k+1$ **to** N **do**
12 $\quad\quad\quad s_k \leftarrow s_k + \log\left(\frac{f\left(q_i \mid \hat{\alpha}_a, \hat{\beta}_a\right)}{f\left(q_i \mid \hat{\alpha}_b, \hat{\beta}_b\right)}\right)$
13 $\quad\quad$ **end**
14 $\quad\quad s_f \leftarrow \max(s_f, s_k)$
15 $\quad$ **end**
16 **end**
17 **if** $s_f > T_h$ **then**
18 $\quad$ **return** *true*
19 **else**
20 $\quad$ **return** *false*
21 **end**

is calculated in the inner loop between lines 11 and 13, where $f\left(q_i \mid \hat{\alpha}, \hat{\beta}\right)$ is the probability density function (PDF) of the *beta* distribution, having estimated parameters $\left(\hat{\alpha}, \hat{\beta}\right)$, applied on the confidence $q_i \in Q$. This PDF describes the relative likelihood for a random variable, in this case q_i, to take on a given value, and it is defined as:

$$f\left(q_i \mid \alpha, \beta\right) = \begin{cases} \frac{q_i^{\alpha-1}(1-q_i)^{\beta-1}}{B(\alpha,\beta)}, & \text{if } 0 < q_i < 1 \\ 0, & \text{otherwise,} \end{cases} \tag{3.3}$$

where

$$B\left(\alpha, \beta\right) = \frac{\left(\int_0^\infty q_i^{\alpha-1} \cdot e^{-q_i} dq_i\right) \cdot \left(\int_0^\infty q_i^{\beta-1} \cdot e^{-q_i} dq_i\right)}{\int_0^\infty q_i^{\alpha+\beta-1} \cdot e^{-q_i} dq_i}. \tag{3.4}$$

We can consider the variable s_k as a dissimilarity score. We estimate s_k multiple times, splitting Q taking different cutoff points k, for all k such as $\Delta \leq k \leq Q - \Delta$. The larger the difference between the PDFs in Q_a and Q_b, the higher the value of s_k (line 12). Let k_{max} be the cutoff point for which the algorithm obtains the maximum s_k value. Finally, a change is detected at point k_{max} if $s_{k_{max}} \equiv s_f$ is greater than a prefixed threshold T_h (line 17). As it is usually done in literature, we use $T_h = -\log(\lambda)$ [163].

Note that our drift detection method can be a bottleneck in the system if we have to run it after inserting each confidence value in Q. Consequently, we limit the number of executions, so that Algorithm 3.3 will be run with a probability of e^{-2q_i}, for any confidence value q_i (line 12 in Algorithm 3.2). Hence, the higher the confidence, the lower the likelihood of executing the drift detection, and *vice versa*. In case a drift is detected, the sliding window Q is reinitialized and drift adaptation is carried out, performing a new local update (lines 14–17 in Algorithm 3.2).

3.3.2 Drift adaptation

As we have already discussed in Section 2.3.1, there exist three main approaches to drift adaptation, which are simple retraining, ensemble learning, and model adjusting. CDA-FedAvg, like FedAvg, is based on neural networks. Depending on the application, these networks can be deep (DNNs), with thousands or millions of parameters, so training them is often expensive. Taking this into account, in this work we ruled out the use of adaptation techniques that involve training a new model from scratch (i.e., simple retraining and ensembles), since they would be rather inefficient. Instead, we opted for an adjusting strategy that allows us to continue adapting the existing model in order to learn new concepts without forgetting the previous ones.

When dealing with neural networks, model adjusting can be achieved in basically three different ways [106]: (1) regularization, (2) rehearsal, and (3) generative replay. On the one hand, *regularization* consists of protecting the important weights of previous concepts from modification. On the other hand, both *rehearsal* and *generative replay* involve feeding the network with examples of past concepts while training on new ones. The difference lies in how these data are managed. Rehearsal approaches save raw samples of each learned concept. Generative techniques, instead, train generative models on the data distribution, thus being able to sample patterns from past experiences at any time, without the need to store raw data.

Although generative replay techniques eliminate the need to store old data, rehearsal has been shown to provide the best results, as it ensures that the memories do not degrade over time [165]. When working in cloud-centric settings, rehearsal can be a major shortcoming, as it can involve a high demand for storage. Nonetheless, when following a federated architecture this is not a problem, since smaller pieces of old memories can be stored locally by each of the clients. Thus, we propose to use rehearsal for drift adaptation in CDA-FedAvg, storing a representative amount of data for each observed concept in the long-term memory. Formally, the long-term memory of a client can be seen as a dataset $\mathcal{L} = \mathcal{L}^0 \cup \mathcal{L}^m \cup \ldots \cup \mathcal{L}^l$, which gathers representative data of each concept detected so far, where $\mathcal{L}^0$ is the data of the first concept, and $\{m, \ldots, l\}$ are the timestamps where the different drifts occur, being $0 < m < \cdots < l$.

Algorithm 3.4 details our approach. It is called the first time the client performs local training, and thereafter whenever a drift is detected. The first thing we do is to wait until we have collected enough data $\mathcal{L}^{new}$ belonging to the new concept we want to learn (line 1). A common question in batch learning is how much data is necessary for training. Unfortunately, the answer is not simple, since it depends on many factors, such as the complexity of the problem or the learning algorithm [166]. Statistical heuristic methods have been frequently used to calculate a suitable sample size, typically based on the number of classes, the number of input features or the number of model parameters. In continual classification tasks, the data from each of the classes can be collected in any order. Thus, to guarantee a minimally balanced rehearsal dataset with the representation of all classes, we define a heuristic rule to control the patterns that are saved in the long-term memory. We set a minimum amount of data, L, so that there must be at least $\frac{L}{2C}$ examples from each class $c_k \in \mathscr{C}$ in the long-term memory representing each concept, where C is the number of possible classes $\mathscr{C} = \{c_1, c_2, \ldots, c_C\}$. Formally, the client keeps collecting data until the following condition is met:

$$\forall c_k \in \mathscr{C} : \left| \left\{ (\mathbf{x}, \mathbf{y}) \in \mathcal{L}^{new} : \mathbf{y} = \mathbf{v}_{c_k} \right\} \right| \geq \frac{L}{2C}, \tag{3.5}$$

where $\mathbf{v}_{c_k}$ is a vector with the binarized representation of c_k, so that all its elements are 0 except for the k component, which is 1.

Once enough data on the new concept is gathered, $\mathcal{L}^{new}$ is added to the long-term memory (line 2). Then, the model update takes place (lines 3–12). The client trains the global model during a certain number of rounds, R. After each round, the update is sent to the server, where it is aggregated with the contributions of other clients (line 11). The next round does not start until the client has received the updated version of the global model.

Algorithm 3.4: Local update (`localUpdate`).

Input : Reference to the latest global model $\mathbf{w}_G$, Long-term memory $\mathcal{L}$, minimum amount of data to train L, minibatch size B, number of rounds R per change, number of local epochs E per round, and learning rate η.
Output : Updated long-term memory $\mathcal{L}$.

$\mathcal{L}^{new} \leftarrow$ `collectData()` // Collect enough new data on the new concept, Eq. (3.5)
$\mathcal{L} \leftarrow \mathcal{L} \cup \mathcal{L}^{new}$ // Expand the long-term memory with the new data
/* Perform local training */
for $r \leftarrow 1$ **to** R **do**
  Split $\mathcal{L}$ into a set of batches $\mathcal{B}$ of size B
  $\mathbf{w}_j \leftarrow \mathbf{w}_G$
  for epoch $e \leftarrow 1$ **to** E **do**
    for batch $b \in \mathcal{B}$ **do**
      $\mathbf{w}_j \leftarrow \mathbf{w}_j - \eta \nabla l(\mathbf{w}_j; b)$ // ∇l is the gradient of the loss function l on b
    end
  end
  $\mathbf{w}_G \leftarrow$ `sendUpdate`$(\mathbf{w}_j)$ // Send new local update to the server and wait for the global aggregation
end
return $\mathcal{L}$

3.4 Analysis of computational complexity

In this section we provide some insights into the complexity of CDA-FedAvg, compared to the original FedAvg. We will analyze three different aspects: (1) time, (2) communications, and (3) memory requirements.

Before comparing the costs of both methods, we need to set some assumptions to get upper bounds on the number of operations required:

1. Firstly, we assume that our loss functions l_j's, which are the same for all clients, are H-smooth. This means that they are differentiable, so we can calculate their gradients, and at the same time their gradients are H-Lipschitz functions [167], which can be expressed as follows:

$$\exists H \in \mathbb{R}^+ : \; || \nabla l_j(\mathbf{u}; b) - \nabla l_j(\mathbf{v}; b) || \leqslant H \, || \mathbf{u} - \mathbf{v} || \quad \forall \, \mathbf{u}, \mathbf{v} \in \mathcal{W}, \;\; \mathbf{u} \neq \mathbf{v}, \tag{3.6}$$

where $\mathcal{W}$ denotes the set of all possible weights, and b is a batch of data. This equation has a geometrical interpretation:

$$|| \nabla l_j(\mathbf{u}; b) - \nabla l_j(\mathbf{v}; b) || \leqslant H \, || \mathbf{u} - \mathbf{v} || \Leftrightarrow \frac{|| \nabla l_j(\mathbf{u}; b) - \nabla l_j(\mathbf{v}; b) ||}{|| \mathbf{u} - \mathbf{v} ||} \leqslant H. \tag{3.7}$$

This last equation, when $\mathbf{u}$ tends to $\mathbf{v}$, is the definition of the derivative function, so this inequality says the derivative of ∇l_j is upperly bounded by a constant value H.

2. Secondly, we also assume that the expectation of the difference between the variations of the local loss functions for each client $l_j(\mathbf{u}; b)$ and the variation of the global loss function $\ell(\mathbf{u})$ is upperly bounded. We define this global loss function as the expected error in all clients when using the weights $\mathbf{u}$:

$$\ell(\mathbf{u}) = \underset{b_j \sim P_j^t}{E} \left[\sum_{j=1}^{N} l_j(\mathbf{u}; b_j) \right]. \tag{3.8}$$

Given this global loss function, our assumption of proximity could be expressed as follows:

$$\exists \sigma \in \mathbb{R}^+ : \mathbb{E} \left[\, || \, \nabla_{\mathbf{u}} l_j(\mathbf{u}; b) - \nabla \ell(\mathbf{u}) \, || \, \right] \leqslant \sigma^2 \quad \forall j \in \{1, \ldots, N\}. \tag{3.9}$$

3. Lastly, we assume the global loss function we try to minimize is convex:

$$\ell(\mu \mathbf{u} + (1-\mu)\mathbf{v}) \leq \mu \ell(\mathbf{u}) + (1-\mu)\ell(\mathbf{v}) \quad \forall \, \mathbf{u}, \mathbf{v} \in \mathcal{W}, \;\; \forall \mu \in [0, 1]. \tag{3.10}$$

This guarantees the global loss minimization problem has a unique solution, and there are no local minimums except for the global minimum.

Taking into account the above assumptions, we can give an upper bound for the time complexity (number of elementary operations) of both FedAvg and CDA-FedAvg. Kairouz *et al.* [168] states that the complexity of FedAvg in terms of the number of training rounds R, the number of local epochs E, and the number of clients selected to train on each round N', is:

$$O\left(\frac{H}{R^2} + \frac{\sigma}{\sqrt{REN'}} \right). \tag{3.11}$$

In our case, under the same assumptions, we carry out the same operations, but we also check whether or not concept drift occurs using Algorithm 3.3. All of those further calculations depend linearly on the maximum size of the sliding window, Q_{max}, which in turn is given by Δ, so our upper bound is shortly bigger:

$$O\left(\frac{H}{R^2} + \frac{\sigma}{\sqrt{REN'}} + \Delta \right). \tag{3.12}$$

However, note that in CDA-FedAvg the number of training rounds R is typically fewer than in FedAvg since, after learning a concept, we stop training until a drift is detected.

In the case of the second aspect we want to consider, communications, CDA-FedAvg is expected to make fewer exchanges of data than FedAvg. In the original method, the communications are bounded by

$$O(RN'). \tag{3.13}$$

Nevertheless, in our case, if a drift is detected, the next data communication takes place after adapting the local model to that drift. If we set ν as the number of times a drift is actually detected divided by the total number of times the detection process (Algorithm 3.3) is executed, then the number of communications our method performs is

$$O\left(RN'(1-\nu) + RN'\frac{\nu C}{L}\right) = O\left(RN'(1-\nu)\right), \tag{3.14}$$

where C is the number of possible classes, and L is a minimum amount of data needed in the long-term memory from each concept. Therefore, if there are no drifts ($\nu = 0$), and assuming the number or rounds R is the same as in FedAvg (although CDA-FedAvg typically uses a lower R), we get the same cost of Equation (3.13). However, in any other case, the communication cost of CDA-FedAvg gets reduced proportionally to the total number of communications expected. In order to compare both cost differences, we can express ν in terms of Δ. Each time a drift is detected, we set the short-term memory $Q = \varnothing$. To detect subsequent drifts, the sliding window needs to be at least of size 2Δ, which implies $\nu \leqslant \frac{1}{2\Delta}$. Thus, we can rewrite

$$O\left(RN'(1-\nu)\right) = O\left(RN' - RN'\nu\right) \leqslant O\left(RN' - \frac{RN'}{2\Delta}\right). \tag{3.15}$$

Finally, regarding the last aspect we want to compare, memory costs, we must be aware of the fact that in FedAvg each client needs to storage all the data obtained locally. Nevertheless, in our case, each client keeps data just until there are enough samples of the current concept. After that, no more storage is needed until detecting a new drift.

3.5 Benefits of active drift management

CDA-FedAvg not only allows continual federated learning, but also answers two fundamental questions that are often overlooked in the literature on federated learning: *what* to learn and

when to do it. This is made possible by shifting decision making from the server to the clients, in particular with active drift detection at the local level. We will briefly discuss why this is so important and what are the benefits.

A naive implementation of FedAvg raises several concerns when deployed in a real multi-device environment. First of all, there is no given criterion on what data to store and what to discard. By default, unless there are task-dependent or implementation restrictions, devices will save all the data they collect. This will often result in unnecessary memory usage. Secondly, there are no defined limits on the number of training rounds to be performed, which could be infinite. Thus, a high commitment of computational resources by the devices is assumed. It is true that only a random subset of clients is selected in each round, so they are not training all the time. In addition, it is possible to limit training even further based on conditions such as whether the device has access to WiFi or is connected to a power supply (this is specially intended for smartphones) [8, 169]. In any case, these criteria do not take into account whether there is a real need for further training. Finally, training constantly also implies an increase in the required bandwidth. In fact, in many works in the literature on FL, authors state that the main bottleneck is the communication cost [22, 157].

Following our approach, instead, it is possible to determine which data is relevant and which is not anymore thanks to the integration of the concept drift detector and a long-term memory. We do not quantify the difference of storage explicitly, but in the FedAvg approach, memory usage grows linearly with the number of training rounds R, whereas in our case it stops when we reach a sufficient number of samples. Besides, it can be assumed that it is only necessary to continue training when a new drift is detected, so that the model can be adapted to it. As shown in Equations (3.11) and (3.12), our approach needs to perform more calculations for the same number of training rounds, at most $O(\Delta)$. However, in the absence of drifts, we can pause the training process, reducing this cost. In addition, Equations (3.13) and (3.15) show that we save at least $O\left(\frac{RN'}{2\Delta}\right)$ communications. Thus, we prove it is possible to significantly reduce the number of communications between clients and server, as well as the computational burden on clients.

3.6 Experimental Results

In this section, we evaluate CDA-FedAvg and compare it with the original FedAvg in different situations. To this end, we conducted several experiments on a real-world task: human activity

recognition (HAR) on smartphones. We believe that this is a good benchmark to test our proposal given its complexity and its natural fit in a collaborative and continual framework. The present section focuses on this single problem in order to properly illustrate how our method works and highlight its main features. For more in-depth results and additional evaluation on other datasets, see Appendix A.

At present, there are several public datasets for HAR, but not so many that are appropriate for experimentation with CFL. This implies having data acquired on a continuous basis over time and from multiple participants. In addition, metadata must be provided indicating the timestamp, user and device to which each sample belongs. Based on these requirements, we selected Shoaib's HAR dataset [170]. It includes data from 10 different people performing seven physical activities: walking, going upstairs, going downstairs, sitting, standing, jogging, and biking. All participants were male, aged between 25 and 30 years. The activities were carried out indoors, in a single building, except for biking, which took place outside. Data were collected at 50 Hz and comprise readings from the inertial sensors of the smartphone: the tri-axis accelerometer, gyroscope, and magnetometer. Recordings were fully labeled for all the participants.

Perhaps the most interesting feature of this dataset is that it provides, for each activity, data recorded with the smartphone placed in 5 different locations (Figure 3.1): (1) on the belt, (2) in the left jeans pocket, (3) in the right jeans pocket, (4) on the right upper arm, and (5) on the right wrist. The smartphones were oriented vertically for the upper arm, wrist, and two pockets, and horizontally for the belt position. Each participant performed each of the 7 activities for about 20 minutes, 4 minutes for each phone position. We took advantage of this to analyze in greater detail the performance of FedAvg and CDA-FedAvg in response to changes in data distribution.

We posed the problem as a multiclass classification task where the goal was to correctly predict which of the 7 activities was carried out by the user. In order to address it in a continual and federated manner, we considered each of the user-smartphone pairs as a client. Besides, we simulated a continuous data acquisition, so that the local datasets were not completely available from the beginning, but the data were processed in a stream. For simplicity in presenting the results, we assumed all clients acquired the data at the same time and with the same frequency. Under this setup, we executed both FedAvg and CDA-FedAvg. We repeated each experiment 10 times, each time using 9 clients for training and the remaining one for testing. The test client was a different one in each execution. Since there were relatively

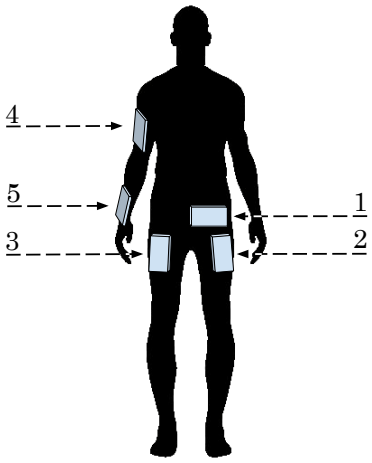


Figure 3.1: Positions in which the smartphone was placed during data recording in HAR.

few participants, we allowed all clients to participate in all training rounds. 10 epochs were performed by each client in each local update.

We split the original raw data into windows of 124 samples, which is equivalent to about 2.5 seconds. We decided to use just the accelerometer and gyroscope signals. Thus, each input pattern was a 6-dimensional time windows, consisting of 124 records for each of the axes of the accelerometer ($\mathbf{a}_x$, $\mathbf{a}_y$, $\mathbf{a}_z$) and the gyroscope ($\boldsymbol{\omega}_x$, $\boldsymbol{\omega}_y$, $\boldsymbol{\omega}_z$). Each client had a total of $5,000$ time windows or instances, $1,000$ for each phone location. In total, in each execution we used $45,000$ samples for training (distributed among the 9 clients) and $5,000$ for testing. Regarding the model, we opted for a convolutional neural network (CNN), given the performance this type of network has shown for inertial signal processing [171, 172]. The architecture consisted of two 1D convolutional layers, pooling, dropout, and two fully connected layers at the end. However, note that, just as with FedAvg, there is nothing preventing CDA-FedAvg from relying on a different network configuration, including any other feed-forward architecture, or even recurrent ones such as long short-term memory (LSTM). For more details on data preprocessing and model architecture, see Section A.5 in Appendix A.

Scenario A: Stationary baseline

To provide a baseline, we first evaluated FedAvg in an ideal continual scenario, where data were stationary (IID at the local level). For that, we randomly shuffled all the samples of each user, without taking into account the position of the smartphone. Each local data stream consisted of $5,000$ iterations, since this was the amount of data available to each participant.

Note that this method was originally designed to deal with static datasets. Therefore, we had to slightly adapt it to the continual scenario. We set a minimum amount of data required to perform the first training round (30 samples per class). We also allowed it to keep the entire local dataset in memory without having to forget old patterns. These are very favorable conditions. We ran FedAvg algorithm during a total of 25 rounds during the continual data acquisition. We repeated the experiment 10 times. On average, at the end of the process, we achieved over 85% accuracy. Table 3.1 details the final performance evaluating on the test data specific to each phone position. Figure 3.2 shows the average evolution over time. The thick black line is the overall accuracy, whilst each of the thin colored lines represents the performance by position.

Table 3.1: Final average results of FedAvg in an ideal (stationary, IID) setting after all clients have processed $5,000$ data samples.

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)

The results obtained with CDA-FedAvg in this scenario are practically identical to those in Table 3.1. This is due to the fact that our method, in stationary situations, works similarly to a regular FedAvg. We have omitted the comparison for the sake of readability. See Section A.2 for further details.

Scenario B: Stationary baseline

Although the results in Table 3.1 are quite good, in real life data are often non-stationary. Hence, in our second experiment, we forced this scenario by sorting the data of all users by phone position. Each change in the placement of the device implies a change in the underlying distribution of data, i.e., a concept drift. Data were sorted in the same way for all clients: 1^{st} belt, 2^{nd} left pocket, 3^{rd} right pocket, 4^{th} upper arm, and 5^{th} wrist. In this way, 4 drifts occur. This is not a totally realistic scenario since in real life each user would acquire data in a particular manner, but it is helpful to constrain the changes and check their impact during

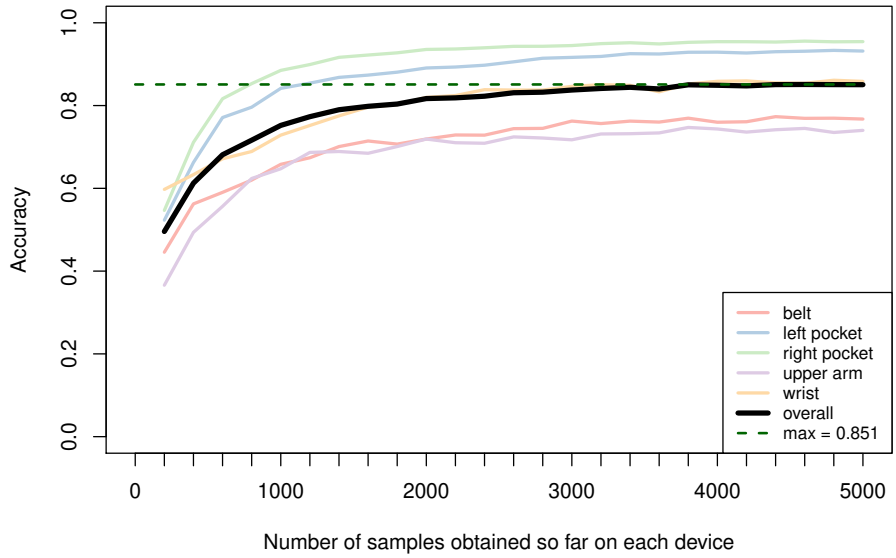


Figure 3.2: Performance of FedAvg in an ideal (stationary, IID) setting. The thick black line is the overall test accuracy.

training. We applied FedAvg again, with exactly the same configuration as before, training a total of 25 rounds. As in the previous case, we repeated the experiment 10 times. Table 3.2 provides the average results by phone location at the end of the training process. Figure 3.3 shows the average evolution over time of such training. The vertical dashed lines indicate where a change in distribution occurs, which is every $1,000$ patterns for all the users. It can be seen that, although the general tendency of the model is to improve, it forgets old concepts as it learns new ones. For example, from iteration $1,000$ it begins to learn about the left pocket position, which causes a drop in accuracy for data related to the previous concept, the belt. The final average accuracy is around 63%.

Table 3.2: Final average results of FedAvg in an non-stationary setting after all clients have processed 5000 data samples.

Phone position	**Average accuracy**
Belt	0.362 (± 0.100)
Left pocket	0.569 (± 0.112)
Right pocket	0.785 (± 0.101)
Upper arm	0.577 (± 0.091)
Wrist	0.873 (± 0.083)
Overall	0.633 (± 0.036)

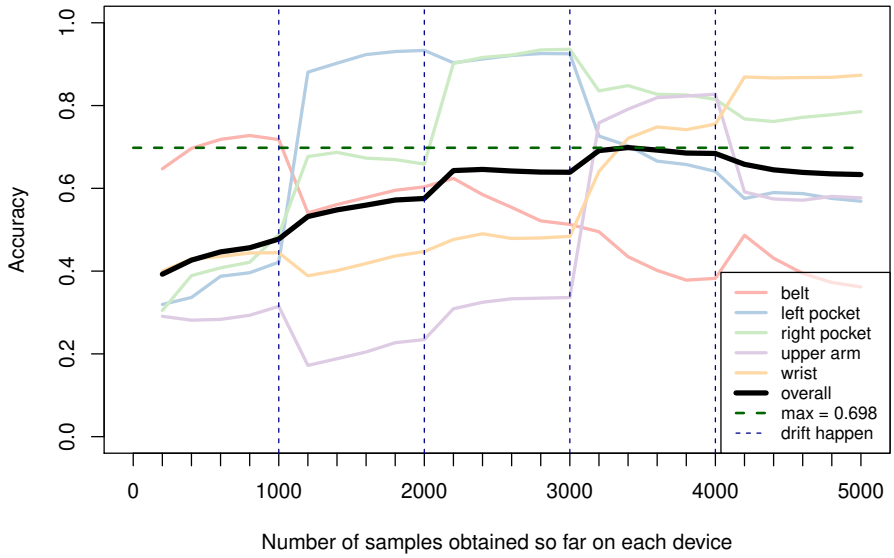


Figure 3.3: Performance of FedAvg in a non-stationary setting. The vertical dashed lines indicate when a distribution change occurs.

We also executed CDA-FedAvg in this scenario. Unlike FedAvg, our method is aware of the existence of concept drift. The first round is not performed until every client has a representative amount of data from the first concept (here, belt position) in its long-term memory. This is given by the parameter L, which in our case we set to 420, and implies that, applying the criterion from Equation (3.5), the training is performed when there are at least 30 examples of each class. We also set the number of training rounds per concept, R, to 5. Thus, in the presence of 5 different concepts, the model will be trained for a total of 25 rounds. This configuration is intentional and seeks to put CDA-FedAvg on equal footing with FedAvg. Even so, FedAvg has additional facilities, such as unlimited storage capacity.

Table 3.3 details the final accuracies per device position when using CDA-FedAvg. As we can see, the overall accuracy at the end of the learning process on the test set is around 82%. This is much closer to the result obtained in the baseline scenario (Table 3.1). If we look at the precision per phone position, we can see how the model completes the training providing good performance for both old and new concepts, never falling below 72% accuracy. This was not the case using FedAvg (Table 3.2), where the performance in the old concepts fell even below 50%. Therefore, CDA-FedAvg denotes greater robustness than FedAvg in this type of scenario.

Unlike in the previous experiments, we cannot represent the average evolution of the accuracy for the 10 executions of CDA-FedAvg in a single plot. This is because, in this case,

drifts are actually detected, depending on the execution, this happens at slightly different times. Therefore, in Figures 3.4 and 3.5 we show two particular executions as examples. Drifts occur at the same time points as in the previous case (iterations $1,000$, $2,000$, $3,000$, and $4,000$). Nevertheless, in this case the vertical dashed lines indicate where each one is detected by at least one of the clients. We can notice that all drifts are identified shortly after they actually occur. In some of the executions, such as the one shown in Figure 3.5, the second drift is not detected. This makes sense, since this change is between the concepts of left pocket and right pocket, which are very similar. In case it was necessary to be more sensitive to change, fine tuning of the λ parameter used for change detection (Algorithm 3.3) could be done.

Table 3.3: Final average results of CDA-FedAvg in an non-stationary setting after all clients have processed 5000 data samples.

Phone position	Average accuracy
Belt	0.758 (± 0.194)
Left pocket	0.907 (± 0.110)
Right pocket	0.847 (± 0.116)
Upper arm	0.725 (± 0.091)
Wrist	0.855 (± 0.075)
Overall	0.819 (± 0.059)

FL has been positioned in recent years as the best choice for multi-device learning, enabling collaborative training while protecting the private data of each client. Nevertheless, the experiments we have just presented provide evidence of the shortcomings of regular federated algorithms. In particular, throughout this section, we have shown that FedAvg is not capable of dealing with continual and non-stationary problems. We have observed that, in the face of virtual drift, the method forgets previous concepts in order to learn the new ones. We have also seen that the average accuracy can drop by more than 25% when moving from a stationary to a non-stationary stream. In the experiments we have also evaluated our proposal, CDA-FedAvg. Its main advantage compared to the original FedAvg is its ability to detect concept drift and adapt to it. This allows it to decide what to learn and when to learn it, thus saving storage, communication and computational resources. We have empirically shown that our method outperforms FedAvg, since CDA-FedAvg provides similar performance in both stationary and non-stationary streams. We have seen that the knowledge acquired on previous concepts does

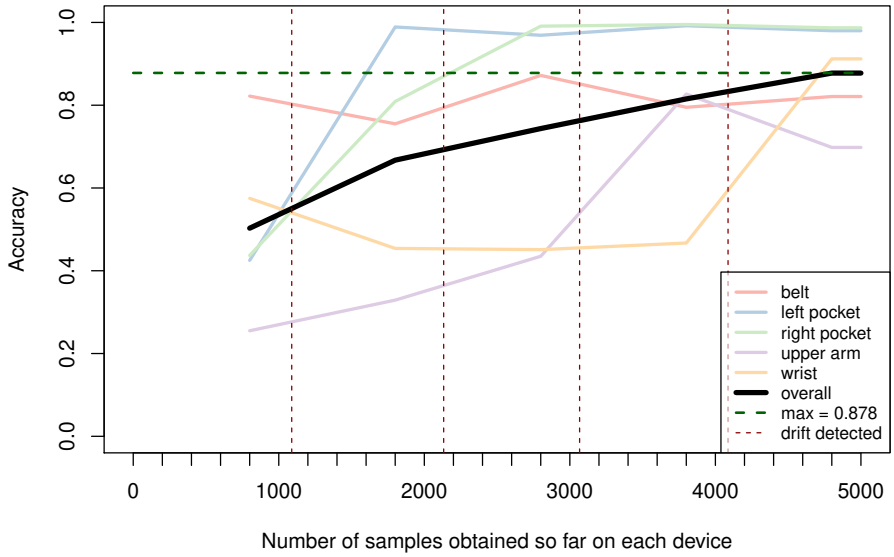


Figure 3.4: Results for CDA-FedAvg in a non-IID and non-stationary setting, training with all users except user 8, whose data is reserved for testing. The vertical dashed lines indicate when a drift is detected by our algorithm.

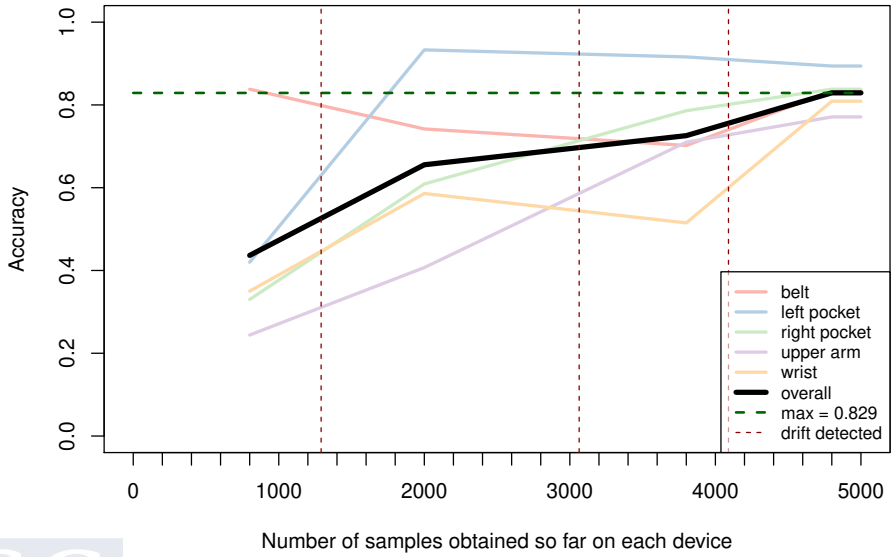


Figure 3.5: Results for CDA-FedAvg in a non-IID and non-stationary setting, training with all users except user 3, whose data is reserved for testing. The vertical dashed lines indicate when a drift is detected by our algorithm.

not degrade over time. Therefore, we can confirm that CDA-FedAvg is able to gradually adapt to changes.

In this section we have provided the most significant results of the experiments conducted on the HAR task. We refer the reader to Appendix A for further discussion. In particular, Sections A.1 and A.5 contain supplementary descriptions and extended experiments. Besides, Section A.2 report further results in the task of handwritten digit recognition involving several datasets, such as MNIST, SVHN, or USPS, among others.

Chapter 4

Ensemble and Continual Federated Learning

In the previous chapter we introduced CDA-FedAvg, which was the first approach to continual federated learning. Nevertheless, as could be sensed from the literature review in Section 3.1, CFL is a very incipient line of research and there is still a lot of work ahead. We already mentioned that FL usually goes hand in hand with DL. This is also the case for CDA-FedAvg. While this combination usually works well, there is no reason to abandon algorithmic diversity. It is well known that traditional learners (Bayesian methods, decision trees, SVMs, etc.) offer other advantages, such as simplicity or explainability, that can become essential in many multi-device problems. Therefore, non-deep alternatives must also be supported in FL and, by extension, in CFL. With this in mind, in this chapter we introduce Ensemble and Continual Federated Learning (ECFL), a continual federated architecture based on ensemble techniques. We propose the federated model to be an ensemble, consisting of several independent learners, which are locally trained. In this way, we enable a flexible aggregation of heterogeneous client models. Our approach tries to make the most of the data stream, being able to deal not only with supervised, but also with semi-supervised classification tasks. Besides, it also integrates active concept drift detection and adaptation mechanisms. Thus, ECFL becomes a simple, flexible and generalizable alternative to carry out CFL. In order to show the benefits of our proposal and illustrate how it works, we have evaluated it in different tasks related to human activity recognition using smartphones.

The contents of this chapter have been reproduced from the following publication (see Appendix B):

> [173] F. E. Casado*, D. Lema*, R. Iglesias*, C. V. Regueiro†, and S. Barro*. "Ensemble and continual federated learning for classification tasks," *Machine Learning*, Springer. ISSN: 0885-6125.
> This paper is currently under review (second round).

The rest of the chapter is structured as follows: Section 4.1 presents in more detail the context and motivation of the problem. Section 4.2 provides a review of the state of the art in ensemble-based methods for FL. In Section 4.3, the ECFL architecture is described. In Section 4.4, some thoughts about privacy are given. Finally, the experimental results are shown in Section 4.5.

4.1 Motivation

The vast majority of works in FL rely on the parallel training of DNNs. That is, all client devices share a common representation of the model and the feature set. Then, each of them locally updates this global model, normally applying SGD in their private data. However, although DL provides good results in a large number of applications, it also has shortcomings that sometimes limit its scope. Perhaps the most critical one is computational complexity [174]. Many multi-device applications deal with low performance hardware, such as smartphones or IoT devices. However, FL typically involves a high number of operations in each training step. This can have a negative impact on the user's daily usage experience: battery draining, overheating, etc. Other well-known problems are model opacity [175, 176] and tendency to overfitting [177]. On top of this, we must remember that DNNs are very prone to suffer catastrophic forgetting [106], which is critical when processing data continuously.

Considering these limitations, new solutions that do not rely on DNNs have started to be explored [87–89]. Typically, these are adaptations of classical ML algorithms to the federated setting. Nevertheless, given that not all ML methods are SGD-based, the common aggregation mechanisms of FL are not always possible. Therefore, specific modifications of the original

*Centro Singular de Investigación en Tecnoloxías Intelixentes (CiTIUS), Universidade de Santiago de Compostela, 15782 Santiago de Compostela, Spain.
†CITIC, Computer Architecture Group, Universidade da Coruña, 15071 A Coruña, Spain.

algorithms are proposed. This is not desirable, since it leads to a lack of homogeneity in solution designs, and makes it difficult to extrapolate them to new scenarios. On top of that, these kind of alternative solutions are scarce for now. In fact, no FL architectures have been proposed that would allow the training of any state-of-the-art algorithm.

In view of this situation, we present ECFL [173], a general and flexible architecture intended to solve multi-device classification tasks using any ML learner in a continual and federated fashion. Our approach strongly relies on ensemble techniques. As already mentioned in Section 2.2, an ensemble is a set of base learners whose outputs are combined to obtain a better predictive performance. The use of ensembles within FL is a natural fit: We propose each client device to train its own local model. Each of these local models will be a potential base learner to join a global and shared ensemble. Keeping the global knowledge up to date over time is as straightforward as replacing learners in the ensemble. ECFL is prepared to work in supervised and semi-supervised scenarios, by using the global model for labeling local data. This is very useful because, when dealing with data streams, true class labels may be scarce or arrive with delay [178].

Our approach has the following strengths:

- It has several advantages in terms of *simplicity*. ECFL opens the door to employ learners that have fewer parameters, require smaller amounts of data, and involve lighter computations than DNNs. Thus, it is more environmentally friendly, in line with the green ML initiative [174, 179].

- There are also benefits regarding *explainability*, since it is possible to use interpretable learners such as decision trees [175]. This is very useful in applications such as medical diagnosis or autonomous driving, where it is critical to know why each decision is made.

- We highlight its *flexibility* as well. The local learning stage consists of training an independent learner on each device, in an asynchronous way, instead of training a single shared model in parallel. This enables the aggregation of heterogeneous client models, which may differ in size, structure, or even algorithmic family. It also reduces the dependency of clients on the server during training, thus avoiding problems such as connection drops and bottlenecks.

- The architecture also provides continual *adaptability*. Local learning integrates concept drift detection and semi-supervised labeling. Each client can update its local model

individually and efficiently. Moreover, the fact that the global model is an ensemble makes it easy to add and remove local contributions.

- Finally, it enhances *generalizability*. It has been shown the ability of ensembles to reduce bias and variance [180, 181]. The more uncorrelated and independent the base learners are, the greater the generalization. In our proposal, each base model is trained on a different device, using local data, thus ensuring this independence. The improvement in generalizability translates into higher robustness of the model.

4.2 Related work

As we have already discussed, literature on FL is closely related to DL, with most of the methods relying on DNNs. Some examples are the aforementioned FedAvg [22], FedProx [83], or our CDA-FedAvg (Chapter 3). Nevertheless, there are very few proposals based on other, non-deep algorithms. In Section 2.2.1 we mentioned some of them. It is the case of the federated SVM introduced by Bakopoulou *et al.* [87], or the federated ID3 decision tree presented by Ludwig *et al.* [88]. These methods are characterized by being designed to solve a specific problem, so their use in new applications is limited. This is mainly because the ML algorithms on which they are based do not naturally lend themselves to federation.

Ensemble techniques [57] are a potential general approach for global model aggregation when parameter averaging is not an option. In fact, they have already been considered on some occasions to implement DML solutions (see Section 2.2). Recently, the idea of using ensemble-based algorithms for FL has also started to be studied. The following is a review of the few existing works in this line.

Motivated by the need to reduce the communication costs, Hamer *et al.* [182] propose FedBoost, an algorithm for learning a weighted ensemble of base predictors in a federated manner. The method assumes that the server is provided with a set of pre-trained based learners. Each of them will have a certain weight in the ensemble. The goal is to learn the set of weights, so that the performance of the ensemble is maximized. The authors experimentally validate FedBoost on synthetic data and also on a real language modeling problem using the Shakespeare corpus. They prove that the method is efficient, with a per-round communication cost independent of the size of the ensemble.

Lin *et al.* [183] develop an algorithm for ensemble distillation in FL settings. They suggest that clients train local learners that are then merged on the server by distillation to obtain the

global model. This approach involves training the global model through unlabeled data on the outputs of the base learners from the clients. The authors validate their approach using several datasets under different constraints. Their knowledge distillation technique mitigates privacy risk and also reduces the size of the ensemble. However, they assume the availability of unlabeled data samples on the server, which is necessary to carry out the distillation. This is a constraining factor, since the protection of clients' private data is indeed one of the main goals of FL. The authors argue that these data does not have to be real, but could also be generated using a generative adversarial network (GAN). Nevertheless, it would still require having a GAN pre-trained beforehand.

Guha *et al.* [77] present *one-shot* FL. Their aim is to learn a global model over a large network of devices in a single round of communication. For that, they propose an ensemble of locally trained SVMs. One of their main concerns is the size of the ensemble, which could grow out of control in federated settings with millions of clients. Therefore, they suggest several strategies to select a subset with the best local models to add to the global ensemble and thus limit its size. In this paper, the authors conduct several experiments using EMMINST, Sentiment140, and Gleam datasets. Although it is a preliminary work, the results show the potential of the approach.

To the best of our knowledge, there are no proposals in the literature for continual federated learning based on methods other than DNNs. The architecture we present in this chapter is in line with the aforementioned works, relying on an ensemble of local learners as a means of global aggregation. Nonetheless, we also take advantage of ensemble techniques to enable continual adaptation.

4.3 The ECFL architecture

We present Ensemble and Continual Federated Learning (ECFL), a new CFL architecture for solving classification tasks based on ensemble learning. Figure 4.1 shows a high-level diagram of our proposal. As we can see, it involves a cyclical process following a client-server communication flow. A key aspect is that there is not a single shared model that is trained in parallel. As in other FL approaches, all clients share a global model. However, in addition to this, in ECFL each of them creates and updates its own local model of the problem in a totally independent way. For that, each client is continuously acquiring new information from its own data stream. The raw data is locally preprocessed and then used to train or update a personal

model. Then, as in conventional federated methods, local models are sent to the cloud where a global aggregation stage is performed, thus obtaining a global model. The global model is then shared with all clients. After that, each device can take advantage of the global knowledge to make more accurate predictions and, at the same time, keep improving the local one. The enhancement of the local learners will also result in an improvement at the global level.

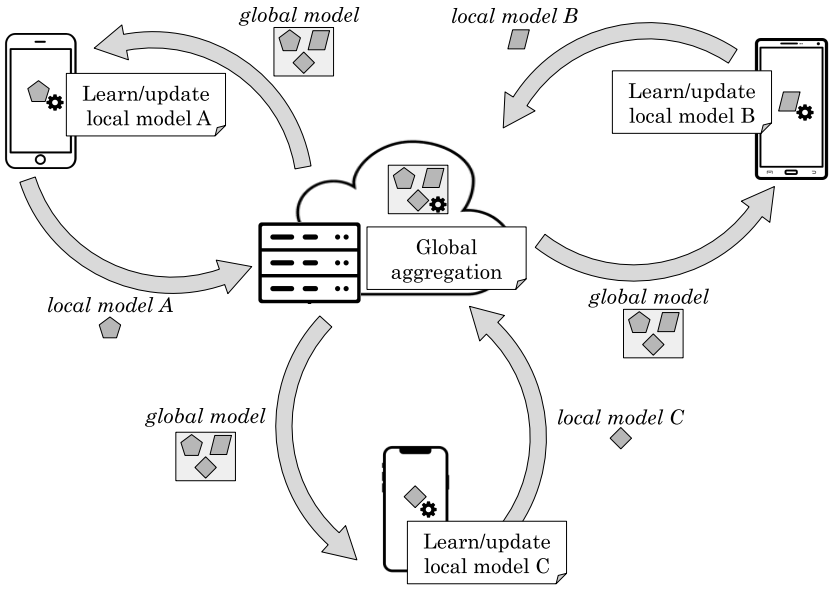


Figure 4.1: High-level diagram of ECFL.

Our architecture is designed to address real multi-device problems of a continual nature. The information available at the local level will increase progressively and may evolve in unpredictable ways. Therefore, ECFL allows adaptive learning from non-stationary data streams without demanding large storage resources. It is able to deal with virtual concept drift, as it integrates local mechanisms for change detection and adaptation. Another problem when operating on data streams is that access to true class labels can be limited or delayed. For instance, spam filtering algorithms can be improved when the user provides feedback, but it can take time since an email arrives until the user classifies it [178]. ECFL is also robust to the lack of labeled instances by incorporating a semi-supervised labeling system.

Ensemble techniques play an important role in our proposal. On the one hand, the global model is an ensemble composed of a selection of local models. This provides an alternative and general way to perform global aggregation different from the common weighted average. The server is in charge of choosing the most relevant local models to become part of the global ensemble through a distributed voting system. The main advantage of this ensemble-based

aggregation is that it works regardless of the learning algorithm used locally, which does not necessarily have to be based on SGD. On the other hand, we also use ensemble techniques locally to allow for continual adaptation to virtual concept drift. In fact, each local model will be an ensemble able to preserve old learners while adding new ones, trained on new data. Drift detection is implemented locally, so that each client is able to determine when the global model has become obsolete. If a drift is detected, the local model is updated and the changes are communicated to the server. Bellow we explain the details of our approach at both learning levels, local (Section 4.3.1), and global (Section 4.3.2).

4.3.1 Local learning

Figure 4.2 shows the local workflow for each client. Devices continually gather raw data from the environment. These data, conveniently preprocessed, are used to create and update the local models. The preprocessing of the data refers to feature extraction, normalization, instance selection, etc. If there is a global model available, each client uses it to annotate the unlabeled samples. In particular, we perform *semi-supervised transduction*. This technique consists of predicting which is the most likely class label for each pattern. The predictions are filtered based on their degree of confidence. As we defined in Section 3.3.1, the confidence of a prediction is the classifier's maximal conditional posterior probability, i.e., the probability $P(c_k|\mathbf{x}) \in (0, 1]$ of a class $c_k \in \mathscr{C}$ to be the correct class for pattern $\mathbf{x}$. We will explain how the confidence of the global model is obtained in Section 4.3.2. We accept a predicted label as the real label when its confidence is equal to or greater than a threshold γ, whose optimal value we have empirically set at $\gamma = 0.9$. Low thresholds ($\gamma < 0.8$) may introduce noise in the training set, while very high thresholds ($\gamma >= 0.95$) may allow to add very few examples to the labeled set. See the experimental results for more details (Section 4.5).

As we mentioned above, in order to train the local models, we have opted for the use of ensembles. In particular, every client builds its own local ensemble of base classifiers*. Any algorithm that provides posterior probabilities for its predictions can be used as base classifier. In our experiments (Section 4.5), we tried different methods: Naïve Bayes, C5.0 decision trees, SVMs, etc. In the same way, any state-of-the-art algorithm could be used to combine the predictions of the base classifiers in the local ensemble. We opted for a simple but effective

*From now on, to avoid confusion we will refer to each of the components of the ensemble as *base classifier* or *learner*, while the term *model* will be used exclusively to designate the ensemble itself.

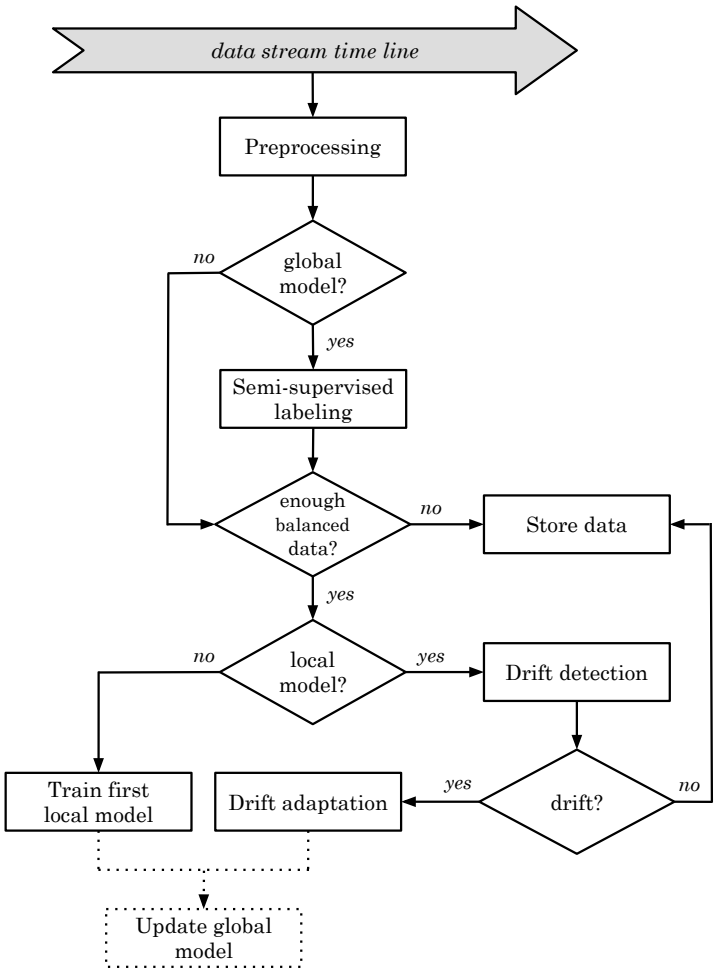


Figure 4.2: ECFL: workflow on the client-side. Note that the process is executed for each new data sample.

approach, employing decision rules, which combine the posterior class probabilities from all base classifiers.

Rule based ensembles have received very much attention because of their simplicity and because they do not require training [184, 185]. When the base classifiers operate in the same measure space, as it is this case, averaging the different posterior estimates of each base classifier reduces the estimation noise, thus improving the decision [186]. Therefore, we should use a rule that averages the posterior estimates of the base classifiers. We use the *median rule* because it is robust to outliers. In this way, we predict that an instance $\mathbf{x}$ belongs to class $c_k \in \mathscr{C} = \{c_1, \ldots, c_C\}$ if the following condition is fulfilled:

$$\text{median}\{\hat{y}_{1,k}(\mathbf{x}), \ldots, \hat{y}_{M,k}(\mathbf{x})\} = \max_{j=1}^{C} \text{median}\{\hat{y}_{1,j}(\mathbf{x}), \ldots, \hat{y}_{M,j}(\mathbf{x})\}, \tag{4.1}$$

where M is the number of base classifiers, and $\hat{\mathbf{y}}_i(\mathbf{x}) = \{\hat{y}_{i,1}(\mathbf{x}), \ldots, \hat{y}_{i,C}(\mathbf{x})\}$ is the output of the i-th classifier given $\mathbf{x}$, for $i = 1, \ldots, M$.

Algorithm 4.1 details the complete local learning process. As soon as a new data sample$(\mathbf{x}, \mathbf{y})$ is available, we classify it using the global model (if any), obtaining an estimated output $\hat{\mathbf{y}}$ with an associated confidence q (lines 8–19). We do this regardless of whether we know the actual label, $\mathbf{y}$, because we will use the confidence record later for drift detection. Both instance and confidence are temporarily stored in a sliding window $\mathcal{S}$ of maximum size S_{max} (line 20). Depending on if we know the true label or not, we will store $\mathbf{y}$ or its estimate $\hat{\mathbf{y}}$. Note that $\mathcal{S} = \{\mathbf{s}_1, \mathbf{s}_2, \ldots, \mathbf{s}_S\}$ is a history of 3-tuples of the form "(features, label, confidence)". The number of 3-tuples that can be stored in $\mathcal{S}$ is given by S_{max}. The memory $\mathcal{S}$ follows the FIFO (First In, First Out) rule. Once its maximum size is reached, adding a new element to it implies deleting the oldest one (lines 5–7).

We keep collecting data samples until we have enough instances to perform the training of the first base classifier for the local model. To ensure a minimally balanced training set with representation of all classes, we follow a heuristic approach similar to the one already used for CDA-FedAvg (Section 3.3.2). We define a minimum amount of labeled data, L, so that there must be at least $L/(2C)$ samples from each class in $\mathcal{S}$ to allow the training process, where C is the number of possible classes. Formally, the client collects data until the following condition is met:

$$\forall c_k \in \mathscr{C} : |\{(\mathbf{x}, \mathbf{y}) \in (\mathcal{X} \times \mathcal{Y}) \cap \mathcal{S} : \mathbf{y} = \mathbf{v}_{c_k}\}| \geq \frac{L}{2C}, \tag{4.2}$$

where $\mathbf{v}_{c_k}$ is a vector with the binarized representation of class c_k. In our experiments we have evaluated ECFL working with different values for L. As a rule of thumb, we can say that a reasonable amount of data is given by $L = 2\Delta$, being Δ the padding parameter used for drift detection that we have already seen in the previous chapter (Section 3.3.1).We also use a maximum size $S_{max} = 20\Delta$ for the sliding window $\mathcal{S}$. When the data in $\mathcal{S}$ meet the condition from Equation (4.2) (line 21, Alg. 4.1), we can proceed to train the first base learner if our local ensemble is empty (line 32). Instead, if the client already has a model (line 22), we must consider whether it is necessary to update it.

It makes no sense to change the model if it is performing well, but it will be important to do it in the presence of concept drift. ECFL reuses the drift detection approach of CDA-FedAvg, which makes it possible to identify virtual drifts. Our detection algorithm analyzes changes in the distribution of the confidences stored in $\mathcal{S}$ and reports them (line 25, Alg. 4.1). This process can be a bottleneck if we have to execute it after each new instance is collected.

Algorithm 4.1: ECFL, client side.

Input : Reference to the global model $\mathcal{E}_G$, minimum of data to train L, confidence threshold γ, maximum size S_{max} for the sliding window, maximum size M_l for the local ensemble, sensitivity to change λ, and padding Δ.

Output : None.

$\mathcal{S} \leftarrow \{\varnothing\}$ // Initialize the sliding window
$\mathcal{E}_l \leftarrow \{\varnothing\}$ // Initialize the local model (an ensemble, for now empty)
while *true** **do**
  if *new data instance,* $(\mathbf{x}, \mathbf{y})$*, is observed* **then**
    if $|\mathcal{S}| = S_{max}$ **then**
      $\mathcal{S} \leftarrow \mathcal{S} \setminus \{\mathbf{s}_1\}$ // Remove the oldest element in $\mathcal{S}$
    end
    if $\mathcal{E}_G \neq \{\varnothing\}$ **then**
      $(\hat{\mathbf{y}}, q) \leftarrow \texttt{classify}(\mathcal{E}_G, \mathbf{x})$ // Classify the pattern using the global model
      if $\mathbf{y} \neq \varnothing$ **then**
        $\mathbf{s}_{new} \leftarrow (\mathbf{x}, \mathbf{y}, q)$
      else if $q \geq \gamma$ **then**
        $\mathbf{s}_{new} \leftarrow (\mathbf{x}, \hat{\mathbf{y}}, q)$ // Apply semi-supervised labeling
      else
        $\mathbf{s}_{new} \leftarrow (\mathbf{x}, \varnothing, q)$
      end
    else if $\mathbf{y} \neq \varnothing$ **then**
      $\mathbf{s}_{new} \leftarrow (\mathbf{x}, \mathbf{y}, \varnothing)$
    end
    $\mathcal{S} \leftarrow \mathcal{S} \cup \mathbf{s}_{new}$
    if $\forall c_k \in \mathcal{C} : |\{(\mathbf{x}, \mathbf{y}) \in (\mathcal{X} \times \mathcal{Y}) \cap \mathcal{S} : \mathbf{y} = \mathbf{v}_{c_k}\}| \geq \frac{L}{2C}$ **then** // Condition from Eq. (4.2)
      if $\mathcal{E}_l \neq \{\varnothing\}$ **then**
        $z \leftarrow \texttt{random}(0, 1)$ // Generate random number between 0 and 1
        if $e^{-2q} \geq z$ **then**
          $d \leftarrow \texttt{driftDetection}(\mathcal{S}, \lambda, \Delta, S_{max})$ // Check for drift (Alg. 3.3)
          if *d is true* **then**
            $\mathcal{E}_l \leftarrow \texttt{localUpdate}(\mathcal{S}, \mathcal{E}_l, M_l)$ // Update local model (Alg. 4.2)
            $\mathcal{S} \leftarrow \{\varnothing\}$ // Reinitialize the sliding window
          end
        end
      else
        $\mathcal{E}_l \leftarrow \texttt{localUpdate}(\mathcal{S}, \mathcal{E}_l, M_l)$ // Learn first base classifier (Algorithm 4.2)
      end
    end
  end
end

* Given the continual nature, we leave the choice of the stop criteria as a matter of implementation.

Therefore, we restrict the number of executions, so that we check for drift with a probability of e^{-2q} (line 24). Hence, the higher the confidence, the lower the probability of executing the change analysis. The details of drift detection are described in Algorithm 3.3, Section 3.3.1. Note that this algorithm takes as input a one-dimensional vector with the confidence history, $\mathcal{Q} = (q_1, \ldots, q_Q)$. In this case, ECFL works with a sliding window $\mathcal{S} = (\mathrm{s}_1, \ldots, \mathrm{s}_S)$, being each $\mathrm{s}_i \in \mathcal{S}$ a 3-tuple whose third component is the confidence q_i. For simplicity, in Algorithm 4.1 we abuse the notation by invoking the drift detection method by passing $\mathcal{S}$ as an argument, although only the confidences are needed.

In case a drift is detected, the local model has to be updated accordingly (line 27, Alg. 4.1). To this end, ECFL integrates a custom ensemble-based adaptation mechanism. In particular, as we already mentioned before, we conceive each local model as an ensemble of base historical classifiers, using the median rule from Equation (4.1). Recall that, for simplicity, we refer to this ensemble as the local model of the client. Algorithm 4.2 details the method. The client is allowed to keep up to M_l base classifiers. The new base learner will be trained using only the labeled data stored in the sliding window $\mathcal{S}$ (lines 1 and 2, Alg. 4.2). Then, it will be added to the local ensemble. If there are already M_l learners in the ensemble, the new one replaces the oldest, thus ensuring that there are at most M_l classifiers at any time (lines 3–6, Alg. 4.2). Each time drift adaptation is applied, the sliding window $\mathcal{S}$ is reinitialized in the main process (line 28, Alg. 4.1). Following this approach, we face the infinite length problem, since the maximum storage memory size for keeping both training data and the model is bounded. In our experiments, we evaluated ECFL for different values of M_l (see Section 4.5).

Algorithm 4.2: Change adaptation algorithm (`localUpdate`).

Input : Sliding window $\mathcal{S}$, local ensemble $\mathcal{E}_l = \{l_1, \ldots, l_m\}$, maximum size of the local ensemble M_l.
Output : $\mathcal{E}_l$.

1 $\mathcal{D} \leftarrow$ `getLabeledPatterns`$(\mathcal{S})$ // Get those samples that have label (real or estimated)
2 $l_{new} \leftarrow$ `train`$(\mathcal{D})$ // Train a new base classifier
3 **if** $|\mathcal{E}_l| = M_l$ **then**
4 $\quad \mathcal{E}_l \leftarrow \mathcal{E}_l \setminus \{l_1\}$ // Remove the oldest base classifier in the ensemble
5 **end**
6 $\mathcal{E}_l \leftarrow \mathcal{E}_l \cup l_{new}$ // Add the new base classifier to the local ensemble
7 **return** $\mathcal{E}_l$

4.3.2 Global learning

The global model is an ensemble that integrates a selection of local models. In this case, not any state-of-the-art ensemble technique can be used to combine the local predictions. There are some interesting approaches, e.g. *stacking* [187], which however we have to discard since they would involve moving raw data to the server. A simple but effective alternative is to use a rule-based ensemble, as we already do at the local level (Section 4.3.1). In this case, the optimal combination rule is the *product rule*, because each local classifier operates in a different measure space (different environments and users) [185]. The product rule predicts that an instance $\mathbf{x}$ belongs to class $c_k \in \mathscr{C}$ if the following condition is met:

$$\prod_{i=1}^{N} \hat{y}_{i,k}(\mathbf{x}) = \max_{j=1}^{C} \prod_{i=1}^{N} \hat{y}_{i,j}(\mathbf{x}), \tag{4.3}$$

where N is the number of clients, $\hat{\mathbf{y}}_i(\mathbf{x}) = \{\hat{y}_{i,1}(\mathbf{x}), \ldots, \hat{y}_{i,C}(\mathbf{x})\}$ is the expected output of the i-th classifier given $\mathbf{x}$, for $i = 1, \ldots, N$.

Although the time complexity on the server side is linear, $O(N)$, including all local models in the global ensemble is not the best option for several reasons. First, depending on the problem, there could be hundreds or thousands of devices connected, so using an ensemble of those dimensions could be computationally very expensive. Second, since the global model is sent back to the clients, it would also have a negative impact on the bandwidth and computational requirements of the clients. Finally, even assuming that there is an optimal global representation of the knowledge, not all the local models will bring the same wealth to the ensemble. On the contrary, there will be clients that, accidentally or intentionally, will have local models with poor performance. It is desirable to detect these cases and prevent them from participating in the ensemble. For all these reasons, we propose to keep a selection of the M_G best local models to participate in the global ensemble. In this way, we can know *a priori* the computational and storage resources we will need.

When the server receives a new local updated, if there are already M_G local models in the global one, the new candidate must compete against those in the ensemble. For that, the server will keep a score representing the relevance of each of the M_G models and will also compute that score for the new update. The computation of the scores is based on the Effective Voting (EV) technique [188]. The original EV performs a cross validation to evaluate each model and then applies a paired t-test for each pair of models to measure the statistical significance of their relative performance. Finally, the most significant ones are selected. In a multi-device context, a local cross-validation is not a fair way to measure the performance of the clients'

models due to the skewness and bias inherent in this type of scenario. With this in mind, we propose an alternative statistical method, inspired by EV. The idea is that each local model is evaluated by other clients on their private data. We call our approach *Distributed Effective Voting* (DEV).

Figure 4.3 summarizes the DEV selection process. When a new local model arrives, the server chooses p different local devices, randomly selected, and asks them to evaluate that classifier on their respective local training sets. Once this evaluation is done, each device sends back to the cloud its accuracy. Assuming that not all the p selected devices are necessarily available to handle the request, the server waits until it has received $\tilde{p}$ performance measures for that model. Both p and $\tilde{p}$ parameters depend on the maximum ensemble size, M_G, and the total number of devices available online, N, so that $M_G \leq \tilde{p} \leq p \leq N$. This process could be considered a distributed cross-validation.

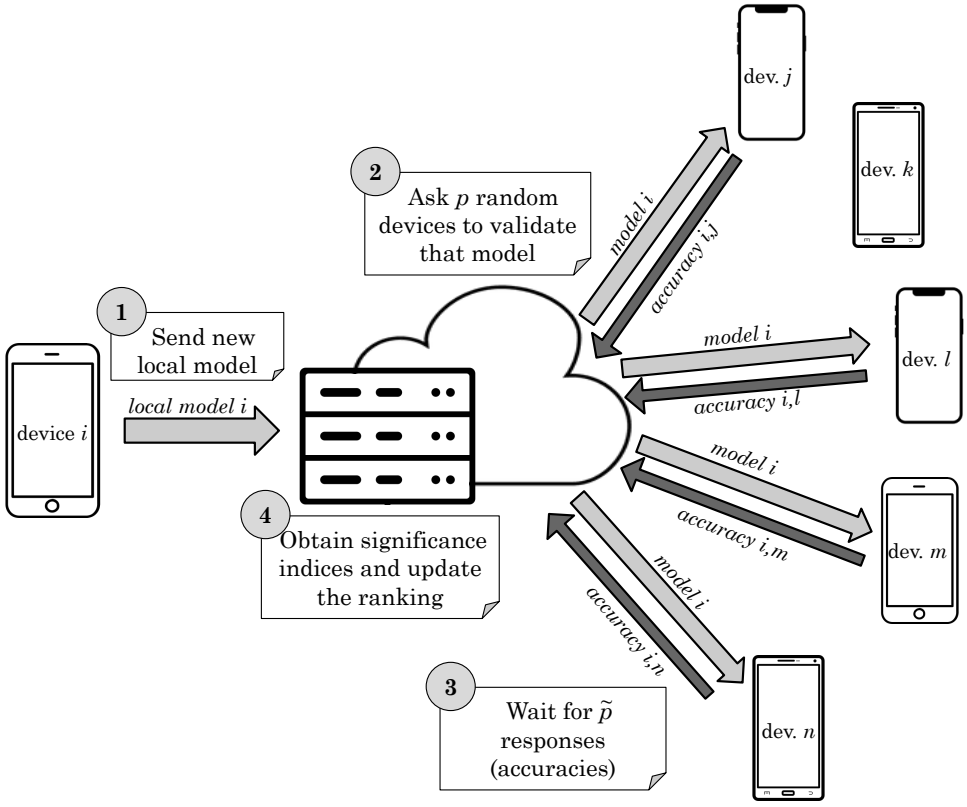


Figure 4.3: Workflow of the Distributed Effective Voting method.

After gathering the $\tilde{p}$ measurements for the current M_G models and the new one, a paired t-test with a significance level of 0.05 is performed for each pair of local models $\mathcal{E}_{l_i}, \mathcal{E}_{l_j}$ so

that:

$$t(\mathcal{E}_{l_i}, \mathcal{E}_{l_j}) = \begin{cases} 1 & \text{if } \mathcal{E}_{l_i} \text{ is significantly better than } \mathcal{E}_{l_j}, \\ -1 & \text{if } \mathcal{E}_{l_j} \text{ is significantly better than } \mathcal{E}_{l_i}, \\ 0 & \text{otherwise.} \end{cases} \tag{4.4}$$

Then, for each model, we calculate its overall significance index:

$$g(\mathcal{E}_{l_i}) = \sum_{j=1}^{M_G+1} t(\mathcal{E}_{l_i}, \mathcal{E}_{l_j}). \tag{4.5}$$

Finally, we select the new M_G models with the highest significance index or score. If there are ties, we break them by selecting the most accurate ones (we compute the mean accuracy from the $\tilde{p}$ available evaluations). As we will see in Section 4.5, we performed several experiments in networks of 10 client devices, so we evaluated ECFL using values for M_G from 3 to 9. For simplicity, we set $p = \tilde{p} = M_G$. In any case, the global ensemble size will be dependent on the problem.

Algorithm 4.3 details the complete server-side behavior of ECFL. Every time a device trains or updates its local model, the changes will be reported to the cloud. The server waits to receive an update (lines 3 and 4). Once this happens, the global model is accordingly modified. If the client that sends the update is already present in the global ensemble, the newer version of that local model replaces the older one (lines 5–7). If the client is not in the global ensemble and its size is still smaller than M_G, then the local model is directly added (lines 8 and 9). Otherwise, our DEV selection method is applied to determine which local models remain in the global ensemble and which one does not (lines 10 and 11). Once the global model is updated, the server shares the new version with all the clients (line 13).

Note that ECFL needs a mechanism for selecting local models to build the global one, but it could be other than DEV. However, designing a good selection method is not straightforward, as some challenges must be addressed. Firstly, local models must be evaluated without centralized data being available anywhere and without even being able to retain all local data. In addition, a balance is required between precision in the selection and efficiency in the overall system. Improving the voting system may involve more communication between server and devices, which limits the scalability of the architecture. The proposed DEV method might select some suboptimal local models from time to time. This makes sense, since it is based just on the voting carried out by a subset of $\tilde{p}$ random clients. Nevertheless, as we will see in Section 4.5, results indicate that the global ensemble always performs close to or better than the best local model.

Algorithm 4.3: ECFL, server side.

Input : List of participant clients $\mathcal{N} = \{1, \ldots, N\}$, maximum size of the global ensemble M_G.
Output : Global model $\mathcal{E}_G$.

$\mathcal{E}_G \leftarrow \{\emptyset\}$ // Initialize the global model (an ensemble, for now empty)
while *true** **do**
  Listen for client updates $\forall j \in \mathcal{N}$
  if $\exists j \in \mathcal{N}$: *new local model* $\mathcal{E}_j^t$ *is received* **then**
    if $\exists \mathcal{E}_j^u : \mathcal{E}_j^u \in \mathcal{E}_G,\ u < t$ **then**
      $\mathcal{E}_G \leftarrow \mathcal{E}_G \setminus \{\mathcal{E}_j^u\}$ // Remove the previous model from user j
      $\mathcal{E}_G \leftarrow \mathcal{E}_G \cup \{\mathcal{E}_j^t\}$ // Add the new one
    else if $|\mathcal{E}_G| < M_G$ **then**
      $\mathcal{E}_G \leftarrow \mathcal{E}_G \cup \{\mathcal{E}_j^t\}$
    else
      $\mathcal{E}_G \leftarrow$ `DistributedEffectiveVoting`$(\mathcal{E}_G, \mathcal{E}_j^t, \mathcal{N})$
    end
    `broadcast`$(\mathcal{E}_G, \mathcal{N})$ // Share the latest version of the global model with all the clients
  end
end

* Given the continual nature, we leave the choice of the stop criteria as a matter of implementation.

4.4 Privacy concerns

An important feature of federated learning, and certainly one of its greatest advantages over other solutions, is its ability to protect data privacy. In our proposal, as is the case with any other FL method, there is no explicit sharing of raw data. In addition, the data are naturally segmented into local storages, from different owners, which makes it more difficult to hack.

Nevertheless, it should be noted that ECFL involves communicating models between the server and the clients. This is something that happens with the majority of FL proposals in their most naive implementation. Unfortunately, it can lead to privacy leaks due to, for example, reverse engineering on the models [189, 190]. To avoid this, there are several protection mechanisms that can be added as additional layers of security. For instance, differential privacy [67, 68, 191] or SMC [192], both already mentioned in Section 2.2.1. Differential privacy could be introduced for client models, so that each participant would have a personalized privacy budget. SMC techniques would enable communications to be secured and protected with cryptographic methods. In particular, homomorphic encryption [71],

would allow to performed many operations directly on encrypted data and models without the need to decrypt them.

In addition to being compatible with the usual FL privacy techniques, ECFL has another very interesting property: The server just acts as a central orchestrator, but it can carry out its tasks without having to access sensitive information, not even local models. This is thanks to the fact that our global aggregation system is an ensemble, and does not require operations on model parameters. Thus, a simple privacy mechanism would be to implement a point-to-point encryption for the local models, combined with their anonymization on the server. In this way, a client's local model would not be decrypted until needed by another client (as part of the global ensemble), which would in any case be unaware of its authorship. We leave the exploration of these privacy issues for future work.

4.5 Experimental results

The aim of this section is to evaluate ECFL, while illustrating how it works. In particular, we are interested in analyzing the performance of the global model, obtained from the consensus of the local devices, in distributed, continual, heterogeneous, and semi-supervised classification scenarios. As mentioned in Section 3.6, activity recognition problems using smartphones fit well within the CFL framework since they involve multiple users and devices gathering data in a regular basis. Therefore, we use this task again to evaluate ECFL. However, there are not many public datasets that allow for experimentation. In the previous chapter, we used Shoaib's HAR dataset [170] to evaluate our CDA-FedAvg method. In this case, in addition to that, we decided to build our own dataset that would allow us to highlight all the properties of ECFL.

Our dataset focuses on the binary task of walking activity recognition. For data collection, we developed an Android application and asked several people to install and run it in the background on their smartphones. The app recorded data continuously while the volunteers were performing their usual routine. With all the data obtained, we created two distinct subsets: one for training, comprising partially labeled data from 10 different people, and the other one for testing, with fully labeled data merged from all other participants. In both cases, several features were extracted from the raw time series. The end result is almost $70,000$ samples for training and $8,000$ for testing. Additional details on data collection, distribution, and preprocessing can be found in Section A.3 in the appendices.

Baseline

Before applying ECFL, we decided to set a baseline. Thus, we trained and fine tuned some of the most popular and widely used supervised classification algorithms: (i) generalized linear model (GLM), (ii) Naïve Bayes (NB), (iii) C5.0 decision tree (C5.0), (iv) support vector machine (SVM), (v) Random Forests (RF), (vi) Stochastic Gradient Boosting (SGB), and (vii) feed-forward neural network (FNN). For that, we used the entire training set, joining the data of the 10 participants. The results of evaluating all the methods on the test set are shown in Table 4.1. We can see that the accuracy of all classifiers ranges between 70% and 90%, but SVM, RF, SGB, and FNN, clearly work better. These are the reference results that could be achieved under ideal conditions, if centralizing user data were possible and there were no temporal constraints. For more details on hyperparameter tuning of these models, see Appendix A.5.

Table 4.1: Performance of several supervised classifiers, trained in ideal conditions.

Method	Balanced Accuracy	Sensitivity	Specificity
GLM	0.722	0.821	0.624
NB	0.795	0.904	0.685
C5.0	0.817	0.884	0.750
SVM	0.846	0.937	0.755
RF	0.855	0.902	0.807
SGB	0.860	0.908	0.811
FNN	0.858	0.913	0.802

Next, we addressed the problem in a federated manner, forcing the data and the learning to be distributed among 10 clients, where each client corresponds to a user-phone pair. In this situation, none of the algorithms from Table 4.1 is directly applicable. Thus, we decided to train a local model for each of the 10 participants and then build a global ensemble using the product rule (since this is the same approach used in ECFL). We also trained two more FNNs using the two most popular FL methods: (a) FedAvg, and (b) FedProx. The performance of each global model is shown in Table 4.2. If we compare these results with those in Table 4.1, we can see some differences. On the one hand, weaker classifiers, such as GLM or Naïve Bayes, have their performance enhanced when used in an ensemble. On the other hand, algorithms that are usually more robust, such as Random Forests or SGB, are weakened by splitting the data among multiple clients. Finally, when using neural networks, both FedAvg

and FedProx are good options, better than an ensemble of networks. Note that we still did not include temporal restrictions, so each client had access to all its data at any time.

Table 4.2: Performance of several supervised classifiers, trained in a distributed way.

Method	Balanced Accuracy	Sensitivity	Specificity
Ensemble (GLM)	0.782	0.959	0.604
Ensemble (NB)	0.819	0.948	0.690
Ensemble (C5.0)	0.809	0.976	0.642
Ensemble (SVM)	0.852	0.956	0.748
Ensemble (RF)	0.819	0.983	0.655
Ensemble (SGB)	0.814	0.959	0.669
Ensemble (FNN)	0.810	0.951	0.669
FedAvg	0.848	0.937	0.758
FedProx	0.844	0.960	0.728

Continual federated setting

Both the results in Table 4.1 and Table 4.2 are useful to get an idea of the performance of the classifiers under ideal conditions. Having this baseline, we introduced the temporal domain to evaluate ECFL. Thus, data acquisition is continuous over time. For simplicity, we assume all clients process data with the same frequency and, therefore, each iteration in our proposal will correspond to a new pattern for all clients. The data stream lasts $10,000$ iterations, given that this is the maximum number of samples collected by each client. Since not all of them reach $10,000$ patterns, those with less data do not start at iteration 1, but join at some point later. The data stream of each participant follows the exact order in which the samples were originally recorded. In this way, we have a realistic data distribution, which will evolve over time, leading to different local concept drifts.

We ran ECFL trying different base classifiers (the same ones from Table 4.1). We also executed FedAvg and FedProx in this setting. In addition, we included CDA-FedAvg, our previous method presented in Chapter 3, in the comparison. Recall that, in continual scenarios, the data stream is of potentially infinite size, so it is impossible to store all incoming data in the memory. For this reason, both CDA-FedAvg and ECFL use a short-term memory of limited size during learning. In the particular case of ECFL, it is a sliding window, W. In order to be on an equal footing, we introduced the same restriction in FedAvg and FedProx, so that in each training round they would use only the data available at that moment in W. We repeated each

execution 10 times, randomly varying the iteration in which each client joins the learning. Table 4.3 compares the average performances of the global models at the end of the data stream. In ECFL, each local model is an ensemble of maximum size M_l, and the global model is composed of up to M_G local models. Besides, there is a minimum amount of labeled data, L, required to train a new learner, and a confidence threshold, γ, used during semi-supervised labeling. The results shown in Table 4.3 were obtained using $M_l = M_G = 5$, $L = 200$, and $\gamma = 0.9$. For more information on the impact of these hyperparameters on the performance of our proposal, see Tables A.5, A.6, and A.7 in Section A.3.

Table 4.3: Average performance of ECFL (using different base classifiers), FedAvg, FedProx, and CDA-FedAvg, trained in a distributed and continual setting.

Method	Balanced Accuracy	Sensitivity	Specificity
ECFL (GLM)	0.743 (±0.066)	0.863 (±0.037)	0.623 (±0.048)
ECFL (NB)	0.789 (±0.014)	0.811 (±0.023)	0.766 (±0.011)
ECFL (C5.0)	0.795 (±0.037)	0.866 (±0.016)	0.726 (±0.021)
ECFL (SVM)	**0.857** (±0.032)	0.871 (±0.007)	0.843 (±0.036)
ECFL (RF)	0.845 (±0.011)	0.911 (±0.014)	0.779 (±0.013)
ECFL (SGB)	0.833 (±0.033)	0.895 (±0.025)	0.771 (±0.019)
ECFL (FNN)	0.803 (±0.027)	0.861 (±0.018)	0.745 (±0.011)
FedAvg	0.806 (±0.016)	0.911 (±0.015)	0.701 (±0.047)
FedProx	0.816 (±0.008)	0.907 (±0.027)	0.725 (±0.035)
CDA-FedAvg	0.831 (±0.012)	0.923 (±0.032)	0.739 (±0.033)

As can be seen in Table 4.3, the best results in this scenario are provided by ECFL, followed by CDA-FedAvg. Nevertheless, the performance of ECFL varies depending on the base classifier. This was to be expected, given that some of them, such as GLM and NB, already provided lower accuracies in the baseline scenario. Note that the direct comparison of the results in Tables 4.1 and 4.2 with those of Table 4.3 is not fair, since this last setting is much more complex, dealing with spatial and temporal constraints at the same time. Even so, we can see that ECFL competes with the baseline models, providing similar performances. Especially noteworthy is the case where SVMs are used as base classifiers, since it outperforms both the single model from Table 4.1 and the ensemble from Table 4.2. We can also see that, while still performing quite well, both FedAvg and FedProx experience some downgrading compared to the previous setting. This is because these methods were not designed for continual learning. It is important to remember that ECFL only uses half of the users to build the global ensemble

model ($M_G = 5$), whereas FedAvg, FedProx, CDA-FedAvg, and all the models in Tables 4.1 and 4.2 are trained using all labeled data from all clients.

To better illustrate the learning process in ECFL, we will now show in detail one of the executions from Table 4.3 as an example. We select the case where we use SVMs as base classifiers. Figure 4.4 details the evolution of the performance per client and over time. The upper graph shows the accuracies of all the models—from each of the 10 users and also the global one—. The thick black line corresponds to the global model, while the rest of the colored lines are each of the 10 clients. As in each iteration the unlabeled local data is labeled with the most recent global model, the more unlabeled data the device has, the more it will be enriched by the global knowledge. The bottom graph shows the local updates and global selection. Once again, each colored line corresponds to one participant. In those places where the line is not drawn it means that the device is not processing data. A circumference ($\circ$) on the line indicates when a drift has been detected and the local model has been updated. If the circumference is filled ($\bullet$), it indicates that the local model is chosen as one of the 5 models of the global ensemble—the other 4 chosen are marked with a cross ($\times$)—. At the end of the process, the global model is composed of the local models of users 2, 4, 5, 7 and 9.

By looking at Figure 4.4, we can see that each client updates their local model between 1 and 5 times. These updates are done based on the concept drift detection and adaptation approach explained in Section 4.3.1. The need for adaptation is conditioned by the evolution of the data distribution of the stream. A more detailed evaluation of the drift detection method is given in Section A.4. Regarding the construction of the global ensemble, it should be reminded that local models are chosen using the DEV method, explained in Section 4.3.2. DEV is based on the voting carried out by a subset of $\tilde{p}$ users randomly chosen (5 in this case, because $\tilde{p} = M_G = 5$). The results presented here, as well as the extended evaluation in Sections A.3 and A.4, demonstrate that the global ensemble provides excellent performance, always similar or greater than that of any local model.

Another important aspect to highlight in Figure 4.4 is the fact that the global model is able to provide good results almost from the beginning. This is due to the great generalization capability that characterizes ensemble methods and classifiers such as SVM, even when the amount of training data is limited. It is something that does not happen using FedAvg or FedProx. Figure 4.5 shows two examples of execution for these two methods. The horizontal axis is represented in terms of federated rounds, and not in terms of samples, since both work with synchronous rounds of local update and global aggregation. We can see how, in both

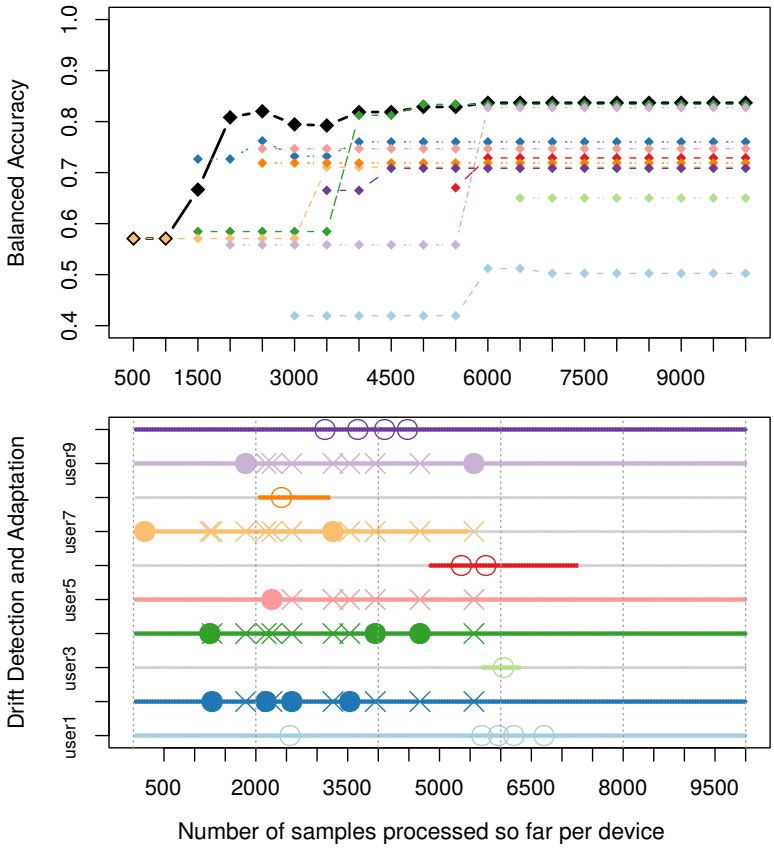
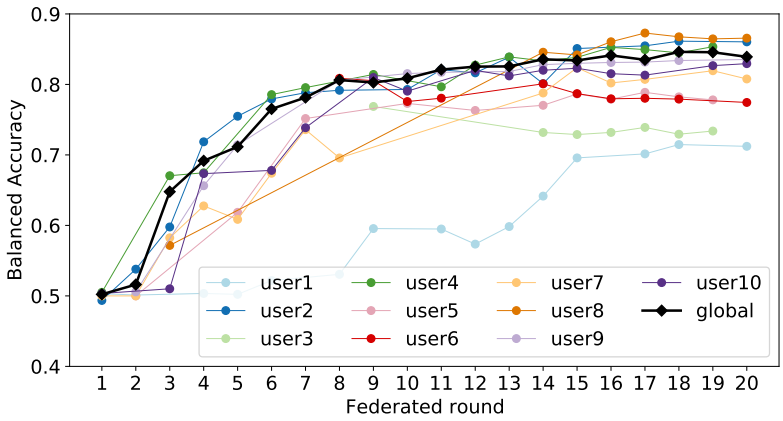


Figure 4.4: Example of executing ECFL with SVM as base classifier. The upper graph shows the evolution of the accuracy over time. The black line corresponds to the global model, while the others are the clients. The bottom graph shows the local updates and global selection.

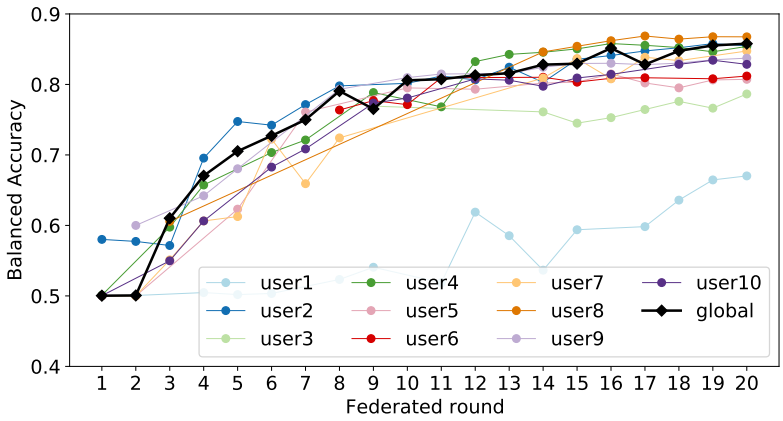
cases, the global model takes approximately 10 rounds to converge, which is roughly the half of the data stream.

Continual federated setting with some users adding noise

To conclude this section, we decided to increase the complexity of the problem a bit more in order to further test our local model selection method, DEV. Suppose now that there are some users who are mislabeling data, whether intentionally or not. To simulate this, we synthetically modified the training data, reversing all the labels provided by 4 of the 10 users. This is the maximum number of clients we can poison to consider them outliers in the system. We chose three very active users (users 1, 2, and 9), and one more not very involved (user 3). Any centralized, distributed or continual algorithm that is unaware of mislabeled data and trained using all available information will give poor results. Our proposal, instead, can detect atypical



(a) FedAvg.



(b) FedProx.

Figure 4.5: Example of execution of FedAvg and FedProx in the federated and continual setting.

participants in an unsupervised manner and exclude them from the global model. Besides, those clients will receive the global model to label the unlabeled samples correctly, thus overcoming the data that was manually mislabeled. Table 4.4 shows the average performance of ECFL, FedAvg, FedProx, and CDA-FedAvg in this continual setting with outliers. As can be appreciated, in this case ECFL far outperforms the results provided by the other approaches.

Figure 4.6 shows an example of executing ECFL in this scenario, in the particular case where SVMs are used as base classifiers. The results are comparable to those shown in Figure 4.4. In this case, at the end of the process, the global model is composed of the local models of users 4, 5, 6, 7 and 10. Figure 4.7 illustrates the process using FedAvg and FedProx. We can see that FedAvg never stabilizes. FedProx, on the other hand, seems to converge more easily. In both cases, it is particularly relevant that users with correctly labeled data do achieve

Table 4.4: Average performance of ECFL, FedAvg, FedProx, and CDA-FedAvg, trained in a distributed and continual setting, when some users mislabel data.

Method	Balanced Accuracy	Sensitivity	Specificity
ECFL (GLM)	0.752 (±0.052)	0.921 (±0.036)	0.583 (±0.050)
ECFL (NB)	0.761 (±0.023)	0.628 (±0.041)	0.893 (±0.028)
ECFL (C5.0)	0.758 (±0.044)	0.831 (±0.027)	0.685 (±0.018)
ECFL (SVM)	0.852 (±0.039)	0.827 (±0.012)	0.876 (±0.021)
ECFL (RF)	0.812 (±0.019)	0.986 (±0.016)	0.638 (±0.011)
ECFL (SGB)	0.822 (±0.015)	0.948 (±0.023)	0.695 (±0.034)
ECFL (NN)	0.799 (±0.026)	0.878 (±0.019)	0.720 (±0.040)
FedAvg	0.669 (±0.189)	0.697 (±0.298)	0.641 (±0.161)
FedProx	0.694 (±0.087)	0.771 (±0.134)	0.617 (±0.139)
CDA-FedAvg	0.729 (±0.099)	0.720 (±0.202)	0.738 (±0.126)

good performance individually. However, the global model is unable to reach the same results since there is no informed selection during the aggregation stage.

Throughout this section we have shown the advantages of ECFL over other state-of-the-art methods. Firstly, as well as CDA-FedAvg, it allows for continual federated learning in non-stationary settings. In fact, the results in Table 4.3 demonstrate that it is able to achieve accuracies similar to those obtained under ideal (stationary and IID) conditions. This is mainly due to the incorporation of drift detection and adaptation mechanisms, which allow it to decide what to learn and when to learn it. Secondly, we have proven good results using different base classifiers. Our architecture is compatible with any classification algorithm, since it combines the outputs of the base learners in a rule-based ensemble. We have seen how some traditionally weaker algorithms, such as GLM or NB, get a boost in performance when used in an ensemble. We have also observed particularly good performance of ECFL when using base classifiers such as SVM, RF, and SGB. Finally, undoubtedly one of the greatest advantages of ECFL compared to the other methods (including CDA-FedAvg) is the mechanism for participant selection: DEV. Normally, FL algorithms are unaware of mislabeled data and perform training using all available information, without applying any further criteria than a random selection of clients per round. ECFL, instead, can detect atypical behaviors and exclude them from the global ensemble. Our last experiment (Table 4.4) perfectly emphasizes the importance of conducting an informed selection of local updates, such as the one we do with DEV.

A more extensive experimental evaluation of ECFL is provided in Appendix A. More specifically, Section A.3 extends the results on walking recognition. It includes details about

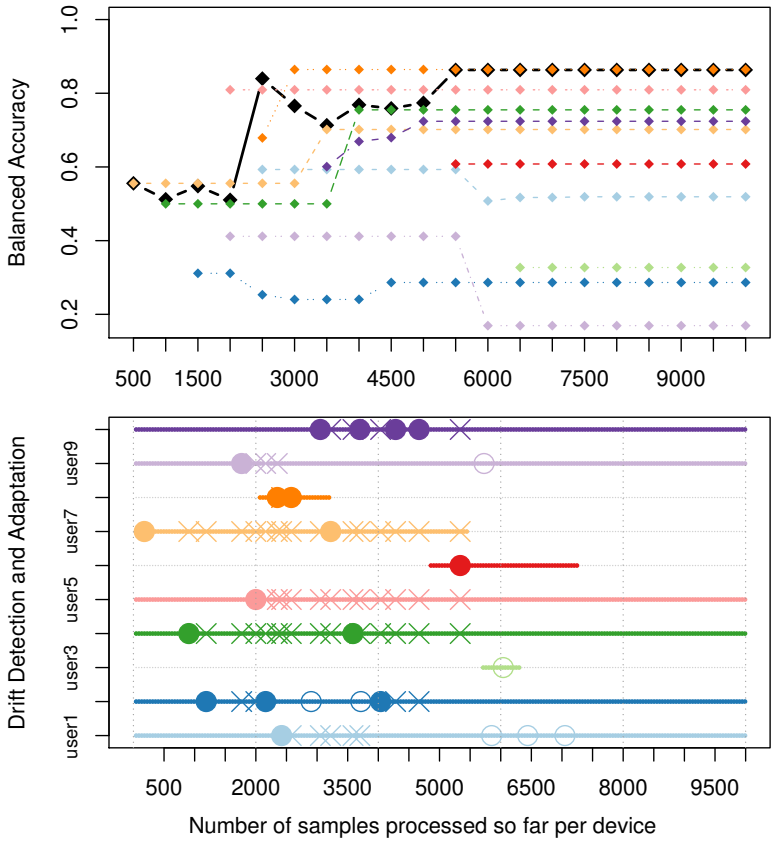
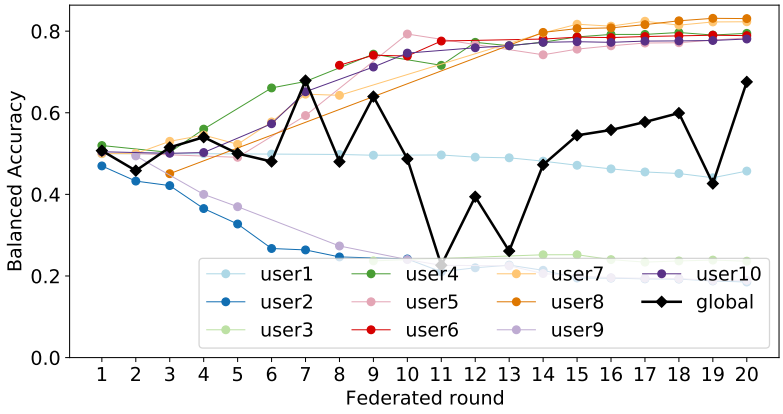
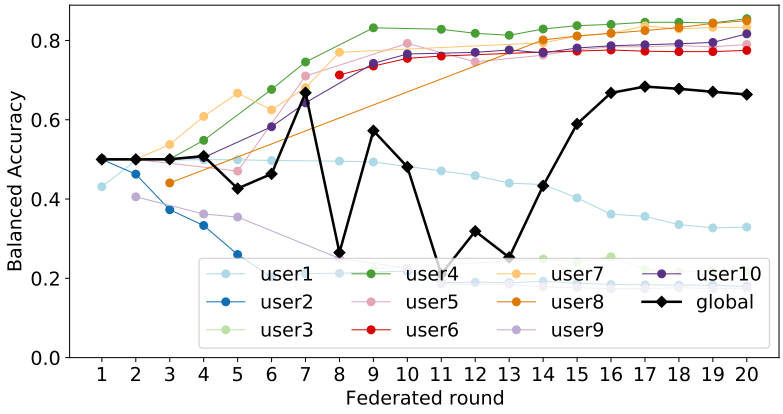


Figure 4.6: Example of executing ECFL when 4 clients mislabel the data. The upper graph shows the evolution of the accuracy. The black line corresponds to the global model, while the others are the clients. The bottom graph shows the local updates and global selection.

data collection, preprocessing and distribution, as well as a comprehensive analysis of the impact of the different hyperparameters on performance. Section A.4 reports new experiments in a different task. In particular, we employ Shoaib's HAR dataset, which was already used to evaluate CDA-FedAvg. Its main advantage is that it contains annotated information on when changes occur, which allows us to better test detection and adaptation to concept drift. Finally, Section A.5 provides additional information (software, hardware, ect.) about the implementation and execution of the experiments.



(a) FedAvg.



(b) FedProx.

Figure 4.7: Example of execution of FedAvg and FedProx in the federated and continual setting when 4 clients mislabel the data.

Chapter 5

Federated Learning from Demonstration: A use case in personal robotics

In Chapters 3 and 4 we presented our main contributions in continual federated learning: CDA-FedAvg and ECFL. For both of them, we performed an experimental evaluation using data taken beforehand, mainly from smartphones. It is true that tasks such as HAR are sufficiently complex and realistic to give us an idea of how the algorithms will work when put into operation. However, it is always preferable to test them directly in real time and on as many problems and hardware as possible. Although CFL have a huge potential for smartphone applications, we believe that there are other devices that can benefit greatly from this technology in the coming years. This is, for instance, the case with *service robots*, i.e., semi or fully autonomous robots that provide useful services for the well-being of people in different contexts such as household, catering, transportation, or healthcare. With the aim of exploring potential applications of our technologies in service robotics, we started a collaboration with Professor Yiannis Demiris, head of the Personal Robotics Lab (PRL) at Imperial College London, in the United Kingdom. The PRL is one of the world's leading laboratories in service robotics research. This chapter presents the last of the contributions of this PhD dissertaion, which is the result of a research visit at the PRL facilities.

Throughout the chapter, we explore the use of federated learning to endow service robots with autonomy. In particular, we focus on the use case of navigation assistance in smart

robotic wheelchairs. We propose Federated Learning from Demonstration (FLfD), a federated approach that allows the wheelchairs to learn how to perform day-to-day tasks from demonstrations provided by human teachers (the end users themselves). Our proposal adopts a collaborative decision making system, in which the user and a DNN share control. The model is trained using sensitive local data (images and laser readings) with privacy guarantees. In our experiments we pose a scenario involving different clients working in heterogeneous domains and show that the federated model is able to generalize to all of them.

Much of the contents of this chapter have been extracted from the following publication (see Appendix B):

> [193] F. E. Casado*, Y. Demiris†. "Federated Learning from Demonstration for Active Assistance to Smart Wheelchair Users," in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2022)*, IEEE, 2022.
> This paper has been accepted and is pending for publication in the conference proceedings. It will be presented in October 2022 in Kyoto, Japan.

A video abstract of this work is available online. It can be accessed by scanning the QR code provided in Figure 5.1. The video shows our proposal in operation in real time. This chapter is organized as follows: Section 5.1 introduces the context and motivation for autonomous navigation of smart wheelchairs. Section 5.2 provides a review of the related work. Section 5.3 details our proposal. Finally, Section 5.4 presents the results.

5.1 Motivation

Mobility aids for the elderly and frail enhance the independence and well-being of their users. Therefore, there is great interest in the research community to develop intelligent robotic systems that can provide such assistance. One notable example are smart robotic wheelchairs, i.e., powered wheelchairs equipped with sensors and a computer [194], such as the one in Fig. 5.2. Using the sensors to gather information about its surroundings, the robot can assist

*Centro Singular de Investigación en Tecnoloxías Intelixentes (CiTIUS), Universidade de Santiago de Compostela, 15782 Santiago de Compostela, Spain.
†Personal Robotics Laboratory (PRL), Dept. of Electrical and Electronic Engineering, Imperial College London, SW7 2AZ, United Kingdom.

Figure 5.1: Frame of the video abstract of the work presented in this chapter. To watch the video, scan the QR code or visit the following URL: https://youtu.be/YHaprlaDk6Q.

the user during navigation in different ways. Typically, the goal is to provide safer or faster navigation, or to reduce the user's cognitive workload. Although this sort of technology may benefit all wheelchair users, people with difficulties in the motor control of the upper limbs would undoubtedly profit the most.

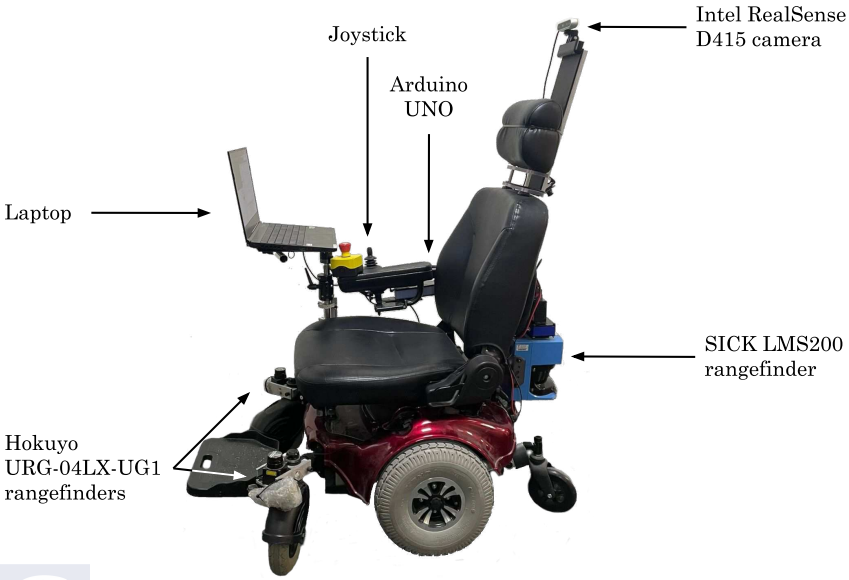


Figure 5.2: Our smart wheelchair in the configuration used for this work.

Empowering the robot with autonomy so that it can provide active navigational assistance is, however, a complex task. Conventional approaches are usually based on the *a priori* map

building of the space in which the wheelchair will move. In this way, different locations of interest can be selected on the map, which will be long-term targets for a path-planning algorithm to reach. However, this is a limitation, since it is infeasible to map and provide targets for all the places a wheelchair user would go in everyday life. Instead, ML-based solutions that can predict the movement the robot should perform *anywhere* are more appealing. In particular, learning from demonstration (LfD) [195] has been shown to lead to significant advances in multiple scenarios involving the use of mobile robots. LfD allows the robot to be instructed by non-expert users and facilitates the learning of adaptive behaviors, so that it can operate in complex and time-varying environments.

In the context of active assistance to smart wheelchair users, LfD could be applied as follows: A human teacher (either the user him/herself, or a professional, like a caretaker or a nurse) drives the wheelchair around the environment while some sensor data (range, vision, etc.) are logged along with the joystick input. Then, a ML model is trained to approximate a mapping function between the sensor data and the expected behavior. Nevertheless, in practice it is challenging to get a model that generalizes and can assist in realistic, dynamic, and complex environments due to several reasons. First, it relies on an expert spending time interacting with the robot, so gathering large amounts of training data takes time. Second, it is very difficult to consider all the possible situations in advance, so usually the dataset is not representative enough. Third, the data needed to represent environment information with accuracy is typically of high dimensionality, which can lead to the curse of dimensionality [196]. These three factors combined can potentially result in model overfitting, which will cause the robot to act unpredictably when facing new scenarios.

We believe that learning from crowds by using FL is an appealing solution for the above shortcomings. The idea is to involve a broad number of teachers, desirably the end users, that would perform the demonstrations in a decentralized way, from home. The data collected from these real environments and working conditions represent an invaluable source of information to train and adapt models and, in consequence, improve robot behavior. Besides, having multiple users providing demonstrations can greatly speed up the learning process. FL is ideally suited to enable this distributed training to take place while keeping private data protected. Thus, in this chapter we explore the possibilities of *Federated Learning from Demonstration* (FLfD) for active assistance to smart wheelchair users. In particular, we propose a collaborative user-robot driving system, in which a federated DNN that integrates vision and range sensing can infer the short-term destinations that the user wants to reach.

For that, we collected data from a driver performing several daily tasks, such as moving along corridors or going through doorways. We worked with several local environments, involving different locations and obstacles. Therefore, data were non-IID [15]. Our model exhibits good generalization capability, providing good performance when facing new domains not seen during training.

5.2 Related work

Assistance for power mobility devices such as smart wheelchairs can be broadly classified into two approaches, namely *reactive* and *active* assistance. Reactive assistance systems [197, 198] tend to keep the user considerably in the loop, taking over control momentarily, with the primary aim of avoiding collisions. On the other hand, in active assistance [199, 200] the user is mostly removed from the control loop and their commands are normally used to infer the desired destination, which the robot will reach by means of autonomous navigation algorithms. Active systems can offer greater autonomy, which makes them more appealing, especially for people with difficulties in the motor control of the upper limbs. Therefore, our work falls within this second approach.

Traditional methods of active assistance rely on considerable spatial maps built beforehand to accommodate long-term assistive planning [199–202]. However, keeping accuracy over time in deployment scenarios demands significant maintenance, because both the environment configuration and user preferences may change. Besides, it is impossible to have mapped all the spaces that the wheelchair user will visit. Hence, it seems more desirable for a system to provide assistance at any location, without the need for a map. This can be achieved adopting the LfD paradigm. The idea is to infer immediate *short-term* destinations the user wants to pass through based only on the environmental information provided by the on-board sensors and previous experiences from a human teacher.

There are a few works that make use of expert demonstrations to learn such short-term navigation behaviors. Chow and Xu [203] achieve this by creating a single-demonstration lookup table consisting of the recorded sensor data and the associated control commands. For each new decision to be taken, the robot has to search in the lookup table for the closest pattern to the real-time-sampled signal. The approach exhibits good performance when evaluated in the same environment used for training. However, the authors do not provide any evidence of the ability to generalize and adapt to new situations. Poon *et al.* [204, 205] present a framework

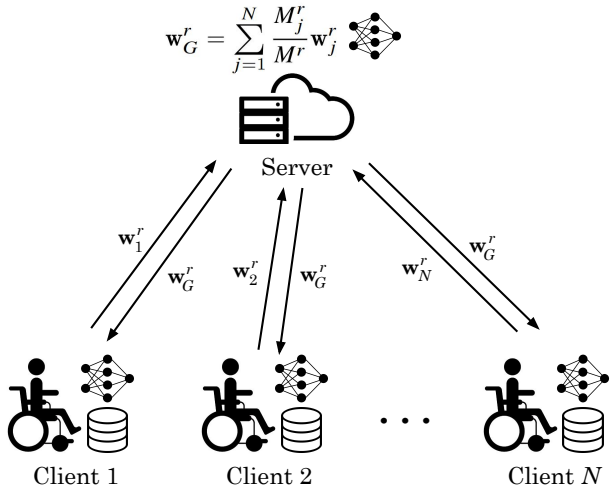
that uses a probabilistic model to estimate short-term user intention. The proposed model, built with Radial Basis Function Networks, takes as input both the on-board sensor data and the direction in which the user wants to move. Training demonstrations are provided by a user driving in a simulated house environment. The results show that this proposal can capture user intentions and produce natural human-like paths in situations not seen during training. Its major constraint is that user intervention is required on a regular basis so that the model can be used.

In this work, we propose a different approach that requires minimal user intervention while providing good generalizability. In our framework, demonstration data can be collected directly by the real end users and the learning process is carried out in a federated way. To the best of our knowledge, our work is one of the first to apply FL techniques in robotics. Li *et al.* [206] introduce a learning architecture for cooperative Simultaneous Localization and Mapping (SLAM) that takes advantage of FL to enhance the performance of visual-LiDAR SLAM in cloud robotic systems. Liu *et al.* [12] present a federated reinforcement learning architecture for navigation that uses transfer learning to make robots quickly adapt to new environments. Finally, regarding federated LfD, Papadopoulos *et al.* [207] propose a conceptualization for developing large-scale multi-agent LfD human-robot collaborative environments by incorporating user profile estimation in order to provide local personalization. Nevertheless, the latter is still a theoretical proposal that needs to be implemented and validated in real-world settings.

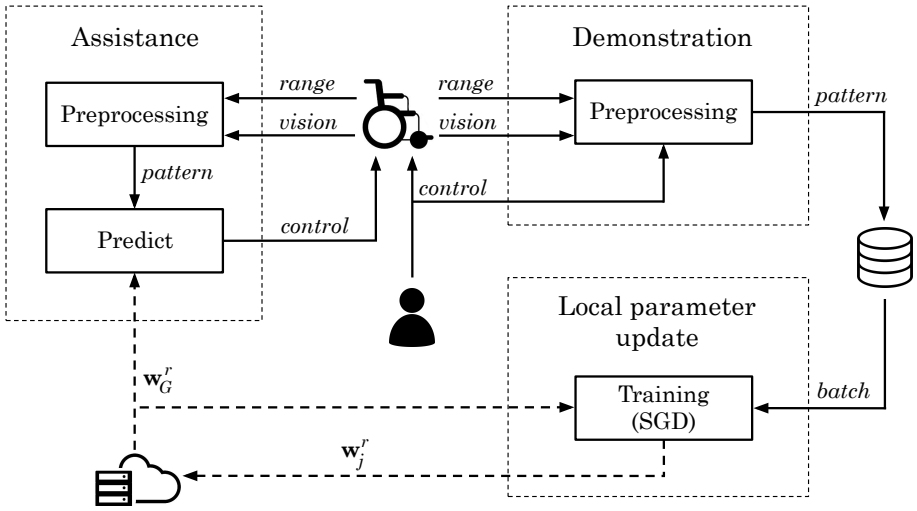
5.3 Methodology: Assisted driving with FLfD

We present a collaborative driving framework which relies on Federated Learning from Demonstration (FLfD) to provide short-term active assistance to smart wheelchair users. Fig. 5.3 gives an overview of our system. We pose a client-server architecture, where the clients are each of the human-robot pairs (Figure 5.3a). Clients and server share and train a common ML model that learns how to assist the users during navigation.

FLfD enables a decentralized and supervised model training. The goal is to infer the most likely joystick commands given the vision and range information from the on-board sensors (Figure 5.3b, right side). Because of the nature of the problem, where each robot processes complex spatio-temporal and multi-sensory information, we apply deep learning. Thus, the proposed model is a DNN, involving convolutional and recurrent layers (see Sec-



(a) Client-server architecture using FedAvg to train the model.



(b) Local workflow for client j.

Figure 5.3: Overview of our FLfD system.

tion 5.3.3). As a first approach, we use FedAvg to train it. Details about the FedAvg algorithm were already provided in Section 2.2.1. Recall that it involves several learning rounds alternating between local updates and global aggregations. In each local step, the clients update the model parameters performing SGD on their private datasets. For global aggregation, the server conducts a weighted average of all updates received.

The model, once trained, is able to assist during navigation (Figure 5.3b, left side). The user can enable and disable the assistance at any time. If it is enabled, human and model share the control of the robot. Human orders always have higher priority. In the absence of any

other indication from the user, the wheelchair follows the joystick commands inferred by the model. In this way, it can complete tasks such as moving along a corridor or going through a doorway without any user intervention. Hence, the wheelchair is fully autonomous in the short term. Note that this does not mean that it is capable of planning a route from beginning to end without any human intervention. This is because, in the absence of other information than the current sensor readings, the model cannot estimate long-term user's intention. Nevertheless, this is not an issue, since control is shared. Thus, the model can do most of the work, while the user will only have to take part occasionally, e.g. to indicate the direction to be followed at junctions.

Each client collects, processes and stores data locally, but never shares them with other clients or the server. Thus, sensitive information, such as camera images, remain safe. As we have already mentioned in previous chapters, there exist additional protection mechanisms that could be added, such as differential privacy [68], or homomorphic encryption [71]. Nonetheless, this is beyond the scope of this work. On the other hand, since this is a first approach, we have trained the federated model offline, but not in real time. We have focused our efforts on creating a dataset that allows the emulation of different distributed environments and facilitates the execution of multiple experiments under several conditions. The construction of this dataset has involved a single robot and a single human, but it includes multiple environments and situations. We have evaluated the assistance provided by the model on that real robot under different conditions (see Section 5.4).

5.3.1 Data collection

For data collection and later experimental evaluation, we used the smart robotic wheelchair from Figure 5.2, internally developed at the PRL, Imperial College London [202]. This robotic platform is based on an electric powered wheelchair with an analog joystick controller. An Arduino UNO interfaces its control bus and reads and writes signals from and to it. Additional sensors were attached to the wheelchair, including a Phidgets spatial 3/3/3 IMU unit, two Hokuyo URG-04LX-UG1 scanning rangefinders situated at the front, one SICK LMS200 rangefinder at the back, and one Intel RealSense D415 camera on top of the headrest. An on-board laptop controls the wheelchair running all the processes included in our Robot Operating System (ROS) framework.

A single individual, experienced in driving the wheelchair, was in charge of performing all the demonstrations. He conducted multiple recording sessions of arbitrary length, but

typically between 1 and 5 minutes. In each session, the raw data from all the sensors and the joystick were processed and stored in a manageable format. The readings from the 3 range sensors were combined into a single 360 degree format. The images from the camera were stored in grayscale and 64×64 resolution (see Sec. 5.3.2). Joystick readings were logged as a 2-dimensional vector consisting of the linear and angular components of the velocity. Finally, the IMU data were recorded as a 6-dimensional vector composed of acceleration and angular velocity. All this information was synchronized using a nearest-sample approach and written to file at a frequency of 20 Hz. The resultant dataset comprises 121 recording sessions. Demonstrations can be grouped into three main blocks: (1) moving along corridors following the wall, (2) going through doorways, and (3) avoiding static and dynamic obstacles. Each recording was annotated with information about the location and the task being performed.

5.3.2 Data preprocessing

The data used for learning comprises range, vision, and joystick twist. The three rangefinders, located at the bottom of the wheelchair, can accurately detect ground-level obstacles (e.g. door frames), but not those that are higher (e.g. tables). The camera, at the top, provides a complementary view of the environment. We want to learn to predict how the wheelchair should move (i.e., the joystick commands) given the information perceived by these sensors.

Each laser measurement is a flat 360 degree scan. A common problem with laser data is that many of the readings may be corrupted. In our system, if the reading is out of the sensor's maximum range, then an `Inf` value is returned, and if it falls within the footprint of the wheelchair, we get a `NaN` value. To avoid discarding samples, we replace all the `Inf` values with the maximum distance value, 10.0 (meters), and all the `NaN`'s with 0.0. In addition, we trim the maximum distance from 10 to 4 meters. Finally, we normalize all the dimensions of the readings between 0 and 1. We denote the resultant set of measurements as $\mathcal{R}$.

The camera used is RGBD and Full HD. To avoid model overfitting due to high dimensionality, we dispense with the depth and process the images as follows. First, they are converted from RGB to a single grayscale channel. Then, they are cropped and resized to 64×64 pixels. In addition, aiming to keep just the most relevant information to facilitate model convergence, Canny edge detector is applied [208]. Finally, each image is normalized between 0 and 1. For the image transformations we use the functions provided by the OpenCV library. We refer to the resulting set of images as $\mathcal{V}$.

The joystick twist readings are 6-dimensional, consisting of the 3-axis linear and angular velocities in the wheelchair's reference system. However, 4 of the 6 values are always 0, because the robot can only move in a plane (forward and backward, rotating on its own axis). Therefore, we get rid of the unnecessary components and we just keep the two non-zero values, where one denotes the movement and the other one the rotation. We normalize both of them between 0 and 1. The result is a set of 2-tuples, that we can call $\mathcal{Y}$.

Each training sample is a tuple $(\mathbf{x}, \mathbf{y})$, where $\mathbf{x} \in \mathcal{X} = \{\mathcal{R} \times \mathcal{V}\}$ is the input, and $\mathbf{y} \in \mathcal{Y}$, the desired output. As we will see in Sec. 5.3.3, the model learns spatio-temporal properties of the input data. Thus, in order to predict $\mathbf{y}_t$, we use as input a vector $\mathbf{x}_t$ composed of the last 10 laser-scan measurements, $(r_{t-9}, r_{t-8}, \ldots, r_{t-1}, r_t) \in \mathcal{R}$, and the last 5 camera images, $(v_{t-8}, v_{t-6}, v_{t-4}, v_{t-2}, v_t) \in \mathcal{V}$. For the laser we sample at a frequency of 20 Hz, while for the images we work at 10 Hz. Thus, each training pattern is composed of the 2 target variables (linear and angular velocity) and 24080 input features (360×10 laser values plus $64 \times 64 \times 5$ pixels).

5.3.3 Model Architecture

The actions taken by the person performing the demonstrations can be modeled by a mapping function:

$$\begin{aligned} f : \mathcal{X} &\longrightarrow \mathcal{Y} \\ \mathbf{x}_t &\mapsto f(\mathbf{x}_t) = \mathbf{y}_t - \theta_t, \end{aligned}$$

where θ represents noise due to lack of accuracy of sensors, errors in the measurements, aliasing, and other random factors. The machine learning task can then be formally defined as learning a regression function $\hat{\mathbf{y}}_t = \hat{f}(\mathbf{x}_t)$, such that $\hat{\mathbf{y}}_t \approx \mathbf{y}_t$, $\forall \mathbf{x}_t \in \mathcal{X}$. To learn $\hat{\mathbf{y}}_t$, we propose to train a DNN with the architecture shown in Fig. 5.4. The total number of parameters is 82562. The model is divided into two main blocks of layers, each associated with one of the inputs. These two blocks are independent of each other and their outputs are combined in the last layer, fully connected.

The first block, at the top in Fig. 5.4, is intended to process the range information, $\mathcal{R}$. For that, we propose to use a total of six 1D convolutional layers, combined with max pooling and batch normalization layers. This first stage allows to automatically extract relevant features from the original signal. Then, we flatten the output of this convolutional series and connect

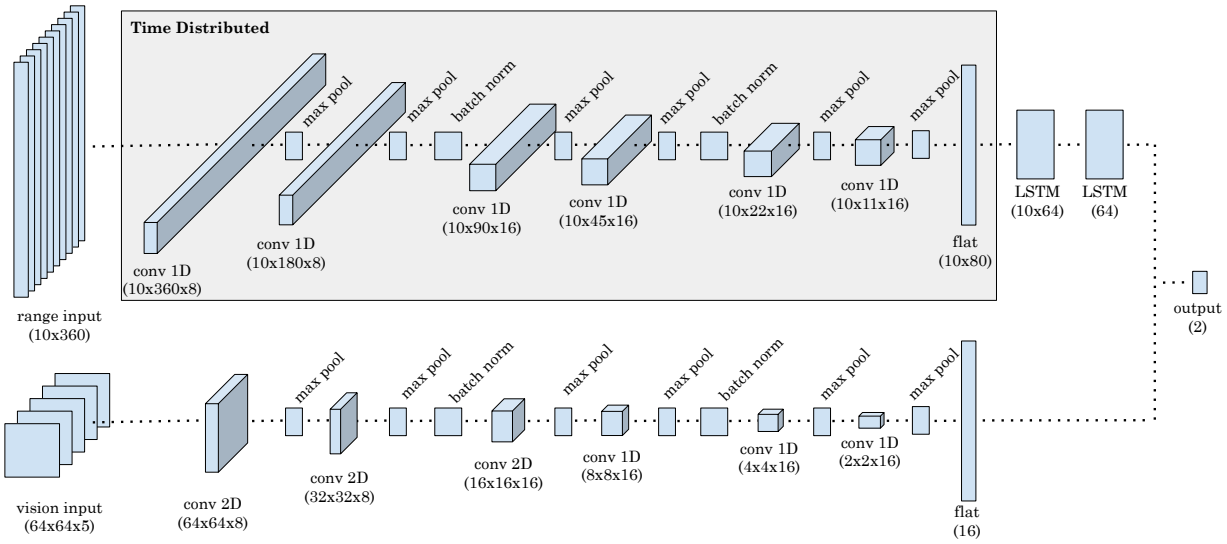


Figure 5.4: Proposed model architecture, integrated in our federated system. The upper block processes the laser information and the lower one processes the images.

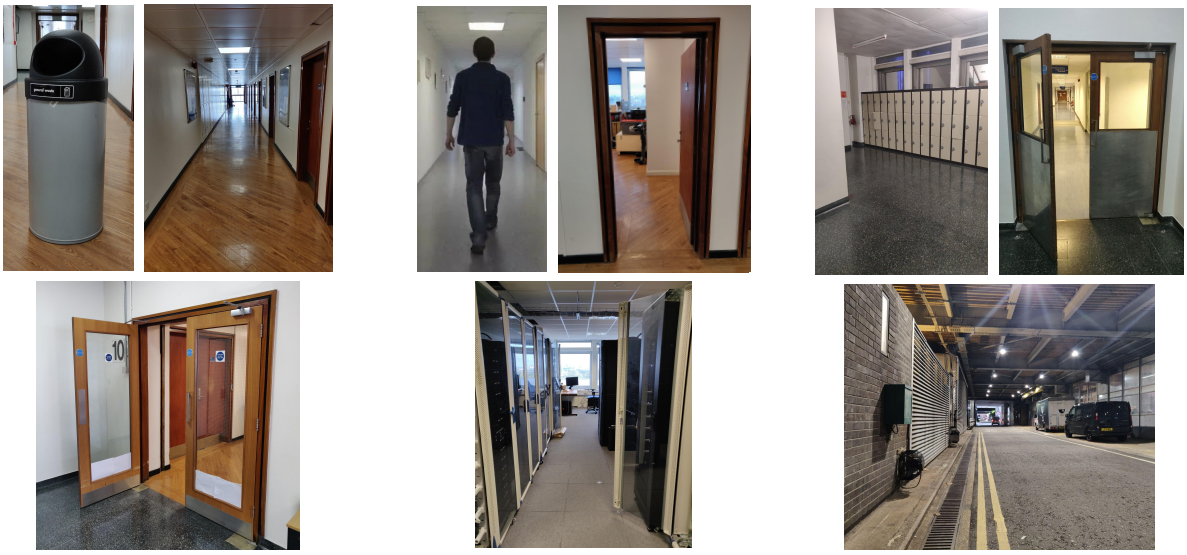
it to two LSTM layers in a row. Note that the convolutional side is *time distributed*, which means that, in each convolutional layer, the same set of weights is used for all 10 timestamps.

The second block, at the bottom in Fig. 5.4, is in charge of processing the visual input, $\mathcal{V}$. It has six 2D convolutional layers, also combined with max pooling and batch normalization. The way in which temporal information is processed in this case is different. Rather than using recurrent layers, we consider the 5 images that compose the pattern as a single 5-channel image. Thus, we feed all 5 images to each layer simultaneously. In this way, we get the network to learn spatio-temporal features from the images at a lower computational cost.

5.4 Experimental results

Given the dataset described in the previous section, we recreated a distributed context for our experiments. To this end, we considered a partitioning of the data into 3 different domains: *A, B,* and *C*, each of them associated to a different client. The 3 domains correspond to data recorded indoors, in corridors and rooms on different floors of a building, although domain C also has recordings from some outdoor areas. The tasks addressed by each client are also quite dissimilar. Figure 5.5 shows examples of locations and obstacles belonging to each domain. Client A has 65 of the 121 recording sessions, with demonstrations of how to move through long corridors, go through double glass doors, and avoid static obstacles (trash bins). Client B holds 43 sessions in which, in contrast, the doors are single, narrower and made of wood,

and the obstacles are dynamic (people walking nearby). Finally, client C has 13 sessions with several types of doors and both static and dynamic obstacles.



(a) Domain A. **(b)** Domain B. **(c)** Domain C.

Figure 5.5: Some pictures from each of the domains used for training and testing.

Note that we could simulate more clients and/or domains with less data associated to each one, but the proposed partition generates three very heterogeneous domains, with non-IID data. This is important because in real-world problems data are rarely IID. We are interested in proving that the federated approach can provide good results even when each local context is very different. We used the majority of the sessions from clients A and B for training, leaving a representative amount for testing. Client C was used exclusively for testing. After applying the preprocessing detailed in Sec. 5.3.2, the total size of the dataset was 199273 samples. Table 5.1 shows the distribution between training and testing for each of the domains.

With the data in Table 5.1, we performed a total of 5 experiments. First, we conducted the federated learning procedure described in Sec. 5.3 involving clients A and B. We ran the learning process for 30 rounds. Both clients participated in all rounds. We used a batch size of 128 and performed a single training epoch on each local parameter update. Then, we repeated the experiment again but setting the number of local epochs per round at 3. To match the number of total epochs to the previous execution, we ran it for 10 rounds. In order to make a comparison, we also trained three other models in a traditional way for 30 epochs. In one of the cases we used only the local data of client A, simulating an isolated training. We did

Table 5.1: Distribution of data by domain and by type of action.

		Task			
		Corridors	Doors	Obstacles	Total
Train	Domain A	28135	46073	17250	91458
	Domain B	11545	17552	15586	44683
	Domain C	0	0	0	0
	Total	39680	63625	32836	**136141**
Test	Domain A	4037	4935	3636	12608
	Domain B	2991	2885	2314	8190
	Domain C	34040	3660	4634	42334
	Total	41068	11480	10584	**63132**

the same for client B. Finally, we trained a model using all available training samples shuffled together. This last model is our baseline, since it was trained under optimal conditions: no privacy issues were considered and all information was centralized on the server, so data were IID. We would like to emphasize that, in a real world scenario, this would not be possible. In this work we have done so for illustrative purposes only. We trained all models in Python 3.9 using the TensorFlow 2.6 library.

Table 5.2 shows the Root Mean Squared Error (RMSE) we obtained with each of the five models when evaluating them on the test data of each domain. We denote as *Fed. (I)* and *Fed. (III)* the federated models in which global aggregation is performed after one and three epochs, respectively. We can observe that the baseline model has the lowest error almost always. This was to be expected, given that it was trained in ideal conditions. However, both federated models, which were trained in a more realistic way, are not far behind. In fact, in some cases, Fed. (I) outperforms the baseline. Between Fed. (I) and Fed. (III), the former generally gives better results. This is reasonable given that the local datasets are non-IID, so performing global aggregation more frequently can facilitate convergence. Both federated options provide good results when evaluated on domain C, which is not used for training. This is a good indicator of their generalization ability. Finally, the local models of clients A and B are the weakest. Both perform quite well when evaluated on their own domain, but the error increases substantially in the others. This may be an evidence of overfitting.

Complementary to the previous evaluation, we designed a set of experiments to test each model in real time using our robotic wheelchair. In each experiment, the robot had to complete a small task successfully. We dispensed with the Fed. (III) model since Fed. (I) performs better.

Table 5.2: RMSE of each model when evaluated in each of the domains.

Domain		Model				
		Baseline	Client A	Client B	Fed. (I)	Fed. (III)
A	Corr.	0.1280	0.1350	0.1718	0.1322	0.1404
	Doors	0.0920	0.1047	0.1145	0.1021	0.0940
	Obst.	0.1016	0.1087	0.1193	0.1049	0.1049
	Overall	**0.1094**	**0.1179**	**0.1396**	**0.1165**	**0.1164**
B	Corr.	0.1107	0.1367	0.1243	0.1132	0.1162
	Doors	0.1357	0.1542	0.1334	0.1448	0.1426
	Obst.	0.1363	0.1418	0.1428	0.1424	0.1458
	Overall	**0.1273**	**0.1445**	**0.1329**	**0.1334**	**0.1346**
C	Corr.	0.0964	0.1120	0.1078	0.1001	0.1022
	Doors	0.1448	0.1605	0.1580	0.1418	0.1460
	Obst.	0.1552	0.1666	0.1839	0.1530	0.1585
	Overall	**0.1070**	**0.1217**	**0.1211**	**0.1093**	**0.1120**
Overall		**0.1103**	**0.1243**	**0.1262**	**0.1141**	**0.1161**

Thus, we repeated each experiment 4 times, one per model. We timed how long it took the wheelchair to complete each task and also logged the incidents that occurred. Table 5.3 shows the times (in seconds) for each of the models. We take as reference the column of the baseline model, where absolute times are shown. In the other three columns, we put the time difference with respect to the first one. We denote with a hyphen (–) those cases where the wheelchair was not able to complete the task. Cells with an asterisk (*) indicate that the robot completed the task, although with minor collisions. As we can see, the models of clients A and B do not provide good results. In the first case, only 70% of the tasks are completed, and in the second, less than 40%. The federated model, on the other hand, is able to complete all tasks. In some cases it manages to finish earlier than the baseline model but, on average, it is 11% slower. Despite this, the federated approach shows the same generalization capability as that obtained in ideal conditions. The experiments summarized in Table 5.3 were recorded. We have posted the video online for those interested in the details (see Figure 5.6).

Table 5.3: Completion times (in seconds) for several tasks.

Task	Model			
	Baseline	Client A	Client B	Federated (I)
Cross a door I (Domain A)	21.26	+0.74	−2.15*	−0.98
Cross a door II (Domain A)	18.20	−0.18	–	+2.06
Change of direction because the door is closed (Domain A)	17.02	−2.03	+1.99	−0.03
Change of direction because the door is blocked by a person (D. A)	14.12	–	+2.89	+2.98
Complete a circuit avoiding static obstacles I (Domain A)	56.08	+10.07	–	+3.17
Complete a circuit avoiding static obstacles II (Domain A)	55.96	−2.86*	–	+10.06
Change of direction at the end of a narrow corridor (Domain B)	34.28	−7.38*	–	+4.01
Move down a corridor avoiding people walking I (Domain B)	16.08	–	–	+5.00
Move down a corridor avoiding people walking II (Domain B)	15.20	–	–	+4.03
Move down a corridor avoiding moving obstacles (Domain B)	23.94	+4.56	+12.08	+5.08
Move down a corridor with glass panes (Domain B)	47.20	−6.01	–	−3.92
Turn at the corner of a corridor (Domain C)	25.91	−7.77	−1.74	−0.77
Cross a door III (Domain C)	17.22	–	–	+6.75
Total	362.47	–	–	+37.44

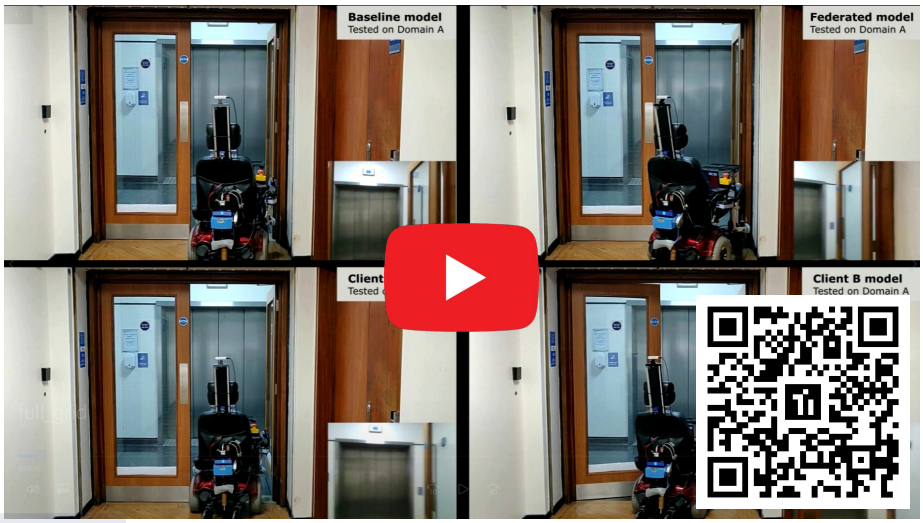


Figure 5.6: Frame of the video containing the full experiments from Table 5.3. To watch the video, scan the QR code or visit the following URL: https://youtu.be/MygKh6ELxQ0.

Chapter 6

Conclusions and future work

Throughout the previous chapters, we introduced the main contributions of this PhD thesis. By way of closing, in the following we present our final remarks and conclusions, as well as some insights on future lines of work. Some of the contents of this chapter have been reproduced under CC BY license from the following publication (see Figure B.2):

> [15] M. F. Criado*, F. E. Casado*, R. Iglesias*, C. V. Regueiro†, and S. Barro*. "Non-IID data and Continual Learning processes in Federated Learning: A long road ahead," *Information Fusion*, Elsevier, vol. 88, pp. 263–280, 2022. DOI: 10.1016/j.inffus.2022.07.024. ISSN: 1566-2535.

Federated learning is a promising approach for multi-device learning, since it poses a distributed, therefore scalable, process while protecting user privacy. However, there is a need for solutions that provide continual and adaptive FL. In many real-world problems, devices gather information on a regular basis, dealing with unbounded sequences of data. In this situations, statistical properties of data may vary in unpredictable ways over time, leading to concept drift. The objective of this PhD thesis was to develop new algorithms that, maintaining all the existing advantages of FL, would allow to handle such continual scenarios with non-stationary data and concept drift.

*Centro Singular de Investigación en Tecnoloxías Intelixentes (CiTIUS), Universidade de Santiago de Compostela, 15782 Santiago de Compostela, Spain.

†CITIC, Computer Architecture Group, Universidade da Coruña, 15071 A Coruña, Spain.

In Chapter 3, we formalized the problem of concept drift in multi-device systems and provided a first approach towards continual federated learning: CDA-FedAvg. Our method is an extension of the original FedAvg to the CFL paradigm. It has the usual advantages of FL, such as privacy protection for clients, but being also capable of detecting virtual concept drift and adapting to it. Each client has a short-term and a long-term memory for different purposes. The short-term memory allows to continuously analyze the data stream in real time. For drift detection, a distribution-based algorithm is used, which relies on a confidence metric to quantify the dissimilarity between the distribution of historical and new data. When a drift happens, the federated model is updated applying rehearsal using the data in the long-term memory. In this way, CDA-FedAvg enables federated training over long periods of time without lowering the performance, knowing exactly what to learn and when to learn it.

In Chapter 4, we moved one step further, introducing a novel ensemble-based CFL architecture: ECFL. Our approach consists of building a global ensemble that combines the outputs of multiple local models, which are trained in a distributed way by the different clients. Each local model is in turn another ensemble, which is built up progressively. When a new local concept drift is identified, a base classifier is trained, becoming part of the local model. This makes it possible to learn over time without forgetting while avoiding storing raw data from past experiences. Furthermore, no restrictions are imposed on the type of algorithm to use to train the base classifiers. In this way, models that do not necessarily rely on gradient descent, such as decision trees, can be combined. This poses further advantages, such as simplicity, flexibility, and generalizability. ECFL is intended for supervised and semi-supervised contexts. The global model can help the clients to label unlabeled samples and improve performance. Another novelty is the incorporation of the DEV system, which allows to select the most suitable participants to form part of the global ensemble.

We performed a thorough evaluation of CDA-FedAvg and ECFL, comparing them with other state-of-the-art methods such as FedAvg and FedProx. We tested the different strategies in several datasets, tasks, and scenarios. Given its relevance and suitability within CFL, we mainly focused on human activity recognition with smartphones, although we also used other common benchmarks, such as handwritten digit recognition. Our main conclusions after the experiments are as follows:

- We pointed out the difficulties that the common FL methods have in learning over time. We observed that, in non-stationary environments, FedAvg and FedProx forget previously learned concepts in order to learn the new ones.

- We empirically demonstrated that both CDA-FedAvg and ECFL are able to learn continuously, without a drop in performance, adapting to unpredictable changes (virtual drifts). They provide similar accuracies in both ideal (stationary) and more complex (non-stationary, with concept drift) situations.

- We proved that CDA-FedAvg outperforms the original FedAvg. Besides, we provided guarantees that our approach helps to reduce storage, communication and computational costs.

- We saw great potential for applying ECFL with many different base algorithms. This is extremely useful for all those problems where the use of DNNs is not an option or, at least, not the most suitable one.

- We demonstrated that ECFL boosts the performance of traditionally weaker classifiers, such as Naïve Bayes.

- Finally, we showed how necessary it can be sometimes to apply participant filtering and how ECFL manages to select the best candidates in order to maintain a good overall performance.

We can affirm that both CDA-FedAvg and ECFL are very capable approaches to address CFL in non-stationary multi-device applications. The choice of one or the other will be conditioned by the specific needs of the problem we want to solve. In some cases, it will be convenient to use DL strategies, so CDA-FedAvg will be the best option. Some examples would be computer vision or NLP tasks. In others, traditional ML algorithms will be more appropriate, so we will use ECFL. Examples of applications falling into this category would be those where the availability of labeled data is limited, involve devices with low computing power, or require to be explainable (as medical diagnosis). Finally, it should be noted that ECFL has other advantages over CDA-FedAvg, such as the labeling system for semi-supervised scenarios, or the informed selection of participants using DEV.

The aforementioned contributions, CDA-FedAvg and ECFL, were evaluated through simulation, and most of the data we used for that were captured with smartphones. While it is true that realistic tasks and complex working conditions were considered, it is always desirable to evaluate algorithms running in real time on a variety of problems and hardware. This was precisely the main objective of the work presented in Chapter 5, in which we proposed a FL

strategy for learning robotic behaviors from crowds. In particular, we developed a collaborative framework for semi-autonomous navigation in smart robotic wheelchairs. Our system relies on learning from demonstration (LfD) to allow the robots to learn to navigate anywhere, without the need to build maps beforehand. What is learned is a DNN that estimates joystick commands based on sensor readings (range and vision). Involving multiple users is desirable, since they can provide large amounts of diverse data. Hence, the model is trained in a federated manner. In our experiments, we worked with both real hardware and scenarios, showing good generalizability to new situations and domains. Although our proposal does not allow for continual adaptation—at least for the time being—, this work has been one of the first to introduce FL in robotics. In addition, the contribution to the field of personal and assistive robotics has been valuable, given that medical professionals are showing a high interest towards even the most basic forms of autonomous driving for medical devices—not only wheelchairs, but also any other movable equipment such as beds.

As a whole, this PhD thesis represents a step forward in multi-device learning research. It has contributed to the maturation and enrichment of the emerging field of federated learning. Starting from the—not yet well established—foundations of this ML paradigm, we moved towards its applicability to real-world problems. This involved dealing with scenarios where data are not static and available from the beginning, but are acquired gradually by the different devices and can evolve in unpredictable ways over time. Thus, we opened a new area of work in continual federated learning (CFL). We mainly focused on addressing the challenge of learning under concept drift with non-stationary data. Nevertheless, we also covered other issues, such as learning from partially labeled data, or saving storage, communications and computational resources in the devices. In addition, our theoretical contributions were also translated into real use cases. On the one hand, we worked with smartphones, addressing human activity recognition. On the other hand, we brought FL to the field of service robotics, successfully developing a solution to assist wheelchair users during navigation.

6.1 Future work

Despite all the contributions mentioned above, we believe that this dissertation is only the starting point for research in CFL. There are several lines of work that we would like to address in the future. These can be grouped into those that concern the federated aspects and those that deal with the continual ones. In other words, we aim to keep working in two distinct

dimensions: *spatial* and *temporal*. This PhD thesis has focused on temporal adaptability, i.e. in the continual dimension, always assuming that all clients share a common goal. Nevertheless, in a network where devices are geographically separated and interact with different users and environments, their learning experiences can be very different. In such situations, it is likely that a single global model is not the best solution and that *spatial adaptability* also needs to be explored. We believe that research in CFL must move forward in both spatial and temporal dimensions in parallel, in order to achieve genuine multi-device learning applicable to real-world problems. In the following, our interests are described in a little more detail.

In the temporal dimension, we have already made significant progress. Proposals such as CDA-FedAvg and ECFL are able to train a federated model over time, from non-stationary data streams under virtual concept drift. Nonetheless, recall that concept drift can be either virtual or real (Section 2.3.1). Virtual drift is caused by variations in the input probability density function, $P^t(\mathbf{x}) \neq P^{t+1}(\mathbf{x})$, from time t to $t+1$. Real drift, instead, involves changes in the conditional probability of the labels with respect to the inputs, $P^t(\mathbf{y}|\mathbf{x}) \neq P^{t+1}(\mathbf{y}|\mathbf{x})$. Some authors have recently started to address real drift in CFL [160, 161], but there is still much room for improvement. Ideally, real drift should be actively detected, just as we do with the virtual one. However, this implies assuming the availability of labeled data that allows to track the error committed by the model. The other, less restrictive option is to perform passive drift management. Instead of analyzing the stream for changes, one can consider updating the model after a specific time interval, assuming that it will eventually become obsolete. This approach, although simpler, is also less efficient in terms of computation and communications. The ultimate goal is to develop algorithms robust to both virtual and real concept drift.

In the spatial dimension, the main challenge lies in dealing with clients whose local datasets follow different, heterogeneous distributions, i.e., they are non-IID. This may be due to different reasons. As we do when analyzing concept drift, it is useful to think in terms of the probability density functions. Thus, we can classify spatial heterogeneity into the following categories:

i) **Differences in perception**. This happens when $P_i(\mathbf{x}) \neq P_j(\mathbf{x})$, but $P_i(\mathbf{y}|\mathbf{x}) = P_j(\mathbf{y}|\mathbf{x})$, for two distinct clients i and j. In this kind of situation, the participants share the same goal, but their datasets contain samples from different input domains. Suppose, for example, that we want to train a federated model for autonomous driving. Some drivers are from Europe and usually pass through traffic circles. Others, instead, are from the United States, and have never seen a traffic circle, but intersections with traffic lights.

That will cause the input space of the different drivers to be skewed, although their goal is the same.

ii) **Differences in behavior**. This means that $P_i(\mathbf{x}) = P_j(\mathbf{x})$, but $P_i(\mathbf{y}|\mathbf{x}) \neq P_j(\mathbf{y}|\mathbf{x})$. This scenario occurs when the input spaces perceived by the clients are analogous, but their outputs are not. This is because sometimes individual interests may differ. Back to the autonomous driving use case, this could happen in a yellow traffic light, where a cautious client would stop, while a more "hurried" one would accelerate before the light turns red. Note that, in these situations, a single global model cannot fit different client behaviors.

Differences in perception and behavior can also occur simultaneously, as is the case in the temporal dimension with the *total* (real and virtual) concept drift. Over the past two years, research on FL with non-IID clients has gained considerable attention [83–86]. The most frequent approach is to personalize the model for each client, trying to balance general and local knowledge. This is known as *personalized federated learning*, and it is a promising line of work, since it allows for adaptation to local particularities without losing generalizability. We are interested in exploring personalization within ECFL. Our architecture has a distinctive feature: the global model is an ensemble consisting of independent local models. This opens up new ways of personalization never before considered. For example, each client could choose at any time which model to use, its own or the global ensemble. To make this decision, several criteria could be taken into account, such as the evolution of the error in the local data, or whether the local model is accepted or rejected as part of the global ensemble.

Another line in which we would like to move forward is what we call *group-level personalization*. The state of the art in personalized FL poses adaptation at the client level, so that each participant can get to have its own customized model. This, although effective, may not be efficient. We believe that between having a single global model for all clients and having N personalized models (one per client), there is a spectrum of intermediate possibilities. We suggest that the number of models to customize is given according to the particular needs of the problem to be solved and should be determined dynamically. In many scenarios, it is likely that there will be groups of users who respond to the same behavior and who can therefore share the same personalized model. One way to address this could be to implement a clustering system to efficiently group clients. In this way, Q models could be personalized, being Q the number of clusters and, in general, $Q \ll N$. The model of each cluster could be learned independently of the others, or all of them could be adapted from a common global

model. The groups could also be adapted over time, allowing new clusters to be created and others to be destroyed.

As can be noted, in multi-device learning the distribution of data may vary in many different ways, both over time and among different clients. To be more precise, a CFL problem can evolve following 16 different courses. We represent all of the possibilities in Figure 6.1. In real-life problems, it will be unlikely to know in advance how this evolution will take place. It is therefore of interest to develop CFL algorithms that are capable of dealing with as many scenarios as possible. Nevertheless, facing both spatial and temporal heterogeneity at the same time is a major challenge. Some of the cases in Figure 6.1 can be solved without too much hassle, but others may require certain conditions to be verified.

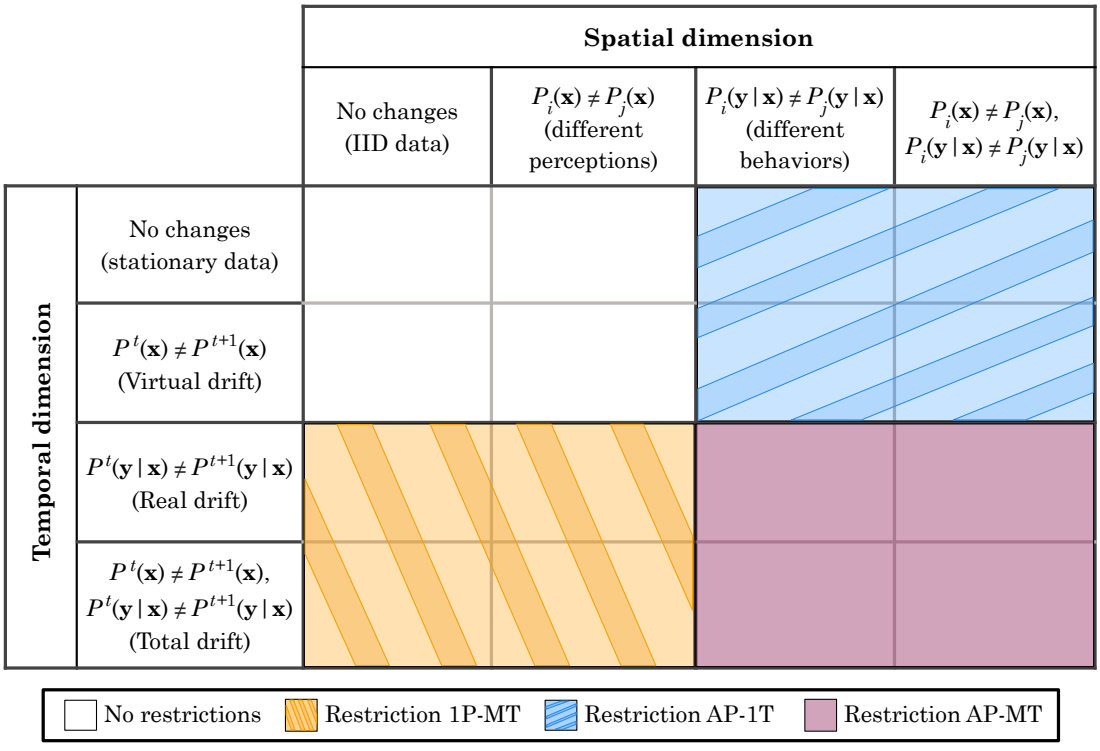


Figure 6.1: Required restrictions for the non-IID learning scenarios.

Variations in the marginal input probabilities, $P(\mathbf{x})$, either in the spatial or temporal dimension, can be faced without needing supplementary information (upper left quadrant in Figure 6.1). In other words, we can tackle them in an unsupervised manner, by simply analyzing the input patterns. This is, for instance, the case of our virtual drift detection algorithm, introduced in Chapter 3. On the contrary, if we seek to detect changes in the

conditional probability, $P(\mathbf{y}|\mathbf{x})$, labels are required. With this in mind, we define the minimum conditions that must be met in order to address each of the scenarios in Figure 6.1:

1. **Restriction 1P-MT** (one participant, many times). If behavior changes over time but not between clients (lower left quadrant in Figure 6.1), then we need labeled data of at least one of the clients from time to time. Since all participants share a common goal, this condition is enough to determine the evolution in the behavior of all of them, detecting and adapting to real concept drift.

2. **Restriction AP-1T** (all participants, one time). If the clients have different conditional probabilities but these do not change over time (upper right quadrant in Figure 6.1), then enough labeled data from each of them is required at the beginning of the process. In this way, it is possible to determine the different behaviors.

3. **Restriction AP-MT** (all participants, many times). If conditional probabilities vary both in the spatial and temporal dimensions (lower right quadrant in Figure 6.1), then we need enough labeled data from all clients from time to time. This is the only way to identify the different client behaviors and how they evolve over time. The more frequently labels are obtained, the greater the capacity to adapt to change. This is the hardest restriction, so that any other scenario considered in Figure 6.1 will also be solvable if it is fulfilled.

Note that, in the process of designing CFL solutions, the federated and continual dimensions can be handled jointly or separately. Today, there are already several algorithms that specialize in one or the other field. These proposals are respectively orthogonal, i.e., they do not interfere on each other. This is very positive, as it allows modularization and recycling of solutions. For instance, a sliding window-based strategy for drift detection could be combined with any personalization technique for spatial adaptation. This modularity is something we already explored in ECFL, where any of its components (local training, drift detection and reaction, global selection of candidates, etc.) could be upgraded or replaced without affecting the rest of the system.

Finally, it is also important to promote the transfer of these technologies to real-world problems. While it is true that nowadays CFL is still in a very early stage, we believe that it will progressively be introduced in more and more applications. Perhaps the most immediate opportunities will be in the field of smartphones and wearables. In this sense, starting from

what we have already done in human activity recognition, we would like to explore real use cases related to health monitoring, sports practice, or indoor positioning. In the longer term, we believe that these technologies will eventually be integrated into other fields, such as industrial and service robotics. We already conducted a first study in the context of assistance to smart robotic wheelchair users. We would like to keep working in this line, carrying out a real-time deployment involving multiple users and robots. Any other application related to human-device interaction could also benefit greatly. We refer to problems involving a large number of people interacting regularly with their devices, thus generating large amounts of decentralized and private data over time. This could include areas such as NLP (conversational agents, text predictors, translators, etc.), autonomous driving, cybersecurity, or domotics, among others. All in all, we foresee a future full of challenges and opportunities for CFL.

Bibliography

[1] S. Li, L. Da Xu, and S. Zhao, "5G Internet of Things: A survey," *Journal of Industrial Information Integration*, vol. 10, pp. 1–9, 2018.

[2] B. Custers, A. M. Sears, F. Dechesne, I. Georgieva, T. Tani, and S. van der Hof, *EU Personal Data Protection in Policy and Practice*. Springer, 2019.

[3] B. M. Gaff, H. E. Sussman, and J. Geetter, "Privacy and big data," *Computer*, vol. 47, no. 6, pp. 7–9, 2014.

[4] G. Ananthanarayanan *et al.*, "Real-time video analytics: The killer app for edge computing," *computer*, vol. 50, no. 10, pp. 58–67, 2017.

[5] U. Montanaro *et al.*, "Towards connected autonomous driving: Review of use-cases," *Vehicle system dynamics*, vol. 57, no. 6, pp. 779–814, 2019.

[6] W. Y. B. Lim *et al.*, "Federated learning in mobile edge networks: A comprehensive survey," *IEEE Communications Surveys & Tutorials*, vol. 22, no. 3, pp. 2031–2063, 2020.

[7] Q. Li *et al.*, "A survey on federated learning systems: Vision, hype and reality for data privacy and protection," *IEEE Transactions on Knowledge and Data Engineering*, 2021.

[8] A. Hard *et al.*, "Federated learning for mobile keyboard prediction," *arXiv preprint arXiv:1811.03604*, 2018.

[9] V. Rey, P. M. S. Sánchez, A. H. Celdrán, and G. Bovet, "Federated learning for malware detection in IoT devices," *Computer Networks*, p. 108 693, 2022.

[10] Y. M. Saputra, D. T. Hoang, D. N. Nguyen, E. Dutkiewicz, M. D. Mueck, and S. Srikanteswara, "Energy demand prediction with federated learning for electric vehicle networks," in *2019 IEEE Global Communications Conference (GLOBECOM)*, IEEE, 2019, pp. 1–6.

[11] T. S. Brisimi, R. Chen, T. Mela, A. Olshevsky, I. C. Paschalidis, and W. Shi, "Federated learning of predictive models from federated electronic health records," *International journal of medical informatics*, vol. 112, pp. 59–67, 2018.

[12] B. Liu, L. Wang, and M. Liu, "Lifelong federated reinforcement learning: A learning architecture for navigation in cloud robotic systems," *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 4555–4562, 2019.

[13] L. U. Khan, W. Saad, Z. Han, E. Hossain, and C. S. Hong, "Federated learning for internet of things: Recent advances, taxonomy, and open challenges," *IEEE Communications Surveys & Tutorials*, 2021.

[14] Y. Chen, Y. Ning, M. Slawski, and 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.

[15] M. F. Criado, F. E. Casado, R. Iglesias, C. V. Regueiro, and S. Barro, "Non-IID data and Continual Learning processes in Federated Learning: A long road ahead," *Information Fusion*, vol. 88, pp. 263–280, 2022.

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

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

[18] A. Garg, N. Shukla, L. Marla, and S. Somanchi, "Distribution Shift in Airline Customer Behavior during COVID-19," *arXiv preprint arXiv:2111.14938*, 2021.

[19] R. Kemker, M. McClure, A. Abitino, T. Hayes, and C. Kanan, "Measuring catastrophic forgetting in neural networks," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, 2018.

[20] G. I. Webb, R. Hyde, H. Cao, H. L. Nguyen, and F. Petitjean, "Characterizing concept drift," *Data Mining and Knowledge Discovery*, vol. 30, no. 4, pp. 964–994, 2016.

[21] J. Konečný, B. McMahan, and D. Ramage, "Federated optimization: Distributed optimization beyond the datacenter," *arXiv preprint arXiv:1511.03575*, 2015.

[22] H. B. McMahan, E. Moore, D. Ramage, and B. A. y Arcas, "Federated learning of deep networks using model averaging," *arXiv preprint arXiv:1602.05629v1*, 2016.

[23] A. L. Samuel, "Some studies in machine learning using the game of checkers. II—Recent progress," *IBM Journal of research and development*, vol. 11, no. 6, pp. 601–617, 1967.

[24] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain," *Psychological review*, vol. 65, no. 6, p. 386, 1958.

[25] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The bulletin of mathematical biophysics*, vol. 5, no. 4, pp. 115–133, 1943.

[26] P. Werbos, "Beyond regression: New tools for prediction and analysis in the behavioral sciences," *Ph. D. dissertation, Harvard University*, 1974.

[27] T. Cover and P. Hart, "Nearest neighbor pattern classification," *IEEE transactions on information theory*, vol. 13, no. 1, pp. 21–27, 1967.

[28] M. Waltz and K. Fu, "A heuristic approach to reinforcement learning control systems," *IEEE Transactions on Automatic Control*, vol. 10, no. 4, pp. 390–398, 1965.

[29] J. MacQueen, "Classification and analysis of multivariate observations," in *5th Berkeley Symp. Math. Statist. Probability*, 1967, pp. 281–297.

[30] J. R. Quinlan, "Induction of decision trees," *Machine learning*, vol. 1, no. 1, pp. 81–106, 1986.

[31] C. Cortes and V. Vapnik, "Support-vector networks," *Machine learning*, vol. 20, no. 3, pp. 273–297, 1995.

[32] E. S. Robert, "A brief introduction to boosting," in *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, vol. 1369, 1999.

[33] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[34] G. E. Hinton, "Learning multiple layers of representation," *Trends in cognitive sciences*, vol. 11, no. 10, pp. 428–434, 2007.

[35] B. Apolloni, A. Ghosh, F. Alpaslan, and S. Patnaik, *Machine learning and robot perception*. Springer Science & Business Media, 2005, vol. 7.

[36] K. Ikeuchi, *Computer vision: A reference guide*. Springer, 2021.

[37] E. Kumar, *Natural language processing*. IK International Pvt Ltd, 2013.

[38] K. P. Kalyanathaya, D. Akila, and P. Rajesh, "Advances in natural language processing–a survey of current research trends, development tools and industry applications," *International Journal of Recent Technology and Engineering*, vol. 7, no. 5C, pp. 199–202, 2019.

[39] L. Gyorfi, G. Ottucsak, and H. Walk, *Machine learning for financial engineering*. World Scientific, 2012, vol. 8.

[40] Y. Gong and W. Xu, *Machine learning for multimedia content analysis*. Springer Science & Business Media, 2007, vol. 30.

[41] J. Yu and D. Tao, *Modern machine learning techniques and their applications in cartoon animation research*. John Wiley & Sons, 2013.

[42] S.-I. Ao, B. B. Rieger, and M. Amouzegar, *Machine learning and systems engineering*. Springer Science & Business Media, 2010, vol. 68.

[43] T. J. Cleophas, A. H. Zwinderman, and H. I. Cleophas-Allers, *Machine learning in medicine*. Springer, 2013, vol. 9.

[44] J. D. Malley, K. G. Malley, and S. Pajevic, *Statistical learning for biomedical data*. Cambridge University Press, 2011.

[45] D. Peteiro-Barral and B. Guijarro-Berdiñas, "A survey of methods for distributed machine learning," *Progress in Artificial Intelligence*, vol. 2, no. 1, pp. 1–11, 2013.

[46] J. Verbraeken, M. Wolting, J. Katzy, J. Kloppenburg, T. Verbelen, and J. S. Rellermeyer, "A survey on distributed machine learning," *ACM Computing Surveys (CSUR)*, vol. 53, no. 2, pp. 1–33, 2020.

[47] P. Brazdil, M. Gams, S. Sian, L. Torgo, and W. Velde, "Learning in distributed systems and multi-agent environments," in *European Working Session on Learning*, Springer, 1991, pp. 412–423.

[48] H. Robbins and S. Monro, "A stochastic approximation method," *The annals of mathematical statistics*, pp. 400–407, 1951.

[49] M. Zinkevich, M. Weimer, L. Li, and A. Smola, "Parallelized stochastic gradient descent," *Advances in neural information processing systems*, vol. 23, 2010.

[50] M. Zinkevich, J. Langford, and A. Smola, "Slow learners are fast," *Advances in neural information processing systems*, vol. 22, 2009.

[51] A. Agarwal, M. J. Wainwright, and J. C. Duchi, "Distributed dual averaging in networks," *Advances in Neural Information Processing Systems*, vol. 23, 2010.

[52] F. Seide, H. Fu, J. Droppo, G. Li, and D. Yu, "1-bit stochastic gradient descent and its application to data-parallel distributed training of speech DNNs," in *Fifteenth annual conference of the international speech communication association*, Citeseer, 2014.

[53] N. Strom, "Scalable distributed DNN training using commodity GPU cloud computing," in *Sixteenth annual conference of the international speech communication association*, 2015.

[54] E. Wei and A. Ozdaglar, "Distributed alternating direction method of multipliers," in *2012 IEEE 51st IEEE Conference on Decision and Control (CDC)*, IEEE, 2012, pp. 5445–5450.

[55] O. Shamir, N. Srebro, and T. Zhang, "Communication-efficient distributed optimization using an approximate newton-type method," in *International conference on machine learning*, PMLR, 2014, pp. 1000–1008.

[56] Y. Zhang and X. Lin, "DiSCO: Distributed optimization for self-concordant empirical loss," in *International conference on machine learning*, PMLR, 2015, pp. 362–370.

[57] O. Sagi and L. Rokach, "Ensemble learning: A survey," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 8, no. 4, e1249, 2018.

[58] P. K. Chan, S. J. Stolfo, *et al.*, "Toward parallel and distributed learning by meta-learning," in *AAAI workshop in Knowledge Discovery in Databases*, vol. 227, 1993, p. 240.

[59] G. Tsoumakas and I. Vlahavas, "Effective stacking of distributed classifiers," in *Ecai*, vol. 2002, 2002, pp. 340–344.

[60] Y. Guo and J. Sutiwaraphun, "Probing knowledge in distributed data mining," in *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, Springer, 1999, pp. 443–452.

[61] L. Breiman, "Pasting small votes for classification in large databases and on-line," *Machine learning*, vol. 36, no. 1, pp. 85–103, 1999.

[62] A. Lazarevic and Z. Obradovic, "Boosting algorithms for parallel and distributed learning," *Distributed and parallel databases*, vol. 11, no. 2, pp. 203–229, 2002.

[63] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Artificial intelligence and statistics*, PMLR, 2017, pp. 1273–1282.

[64] R. Johnson and T. Zhang, "Accelerating stochastic gradient descent using predictive variance reduction," *Advances in neural information processing systems*, vol. 26, 2013.

[65] J. W. Bos, K. Lauter, and M. Naehrig, "Private predictive analysis on encrypted medical data," *Journal of biomedical informatics*, vol. 50, pp. 234–243, 2014.

[66] S. Truex *et al.*, "A hybrid approach to privacy-preserving federated learning," in *Proceedings of the 12th ACM Workshop on Artificial Intelligence and Security*, 2019, pp. 1–11.

[67] S. Augenstein *et al.*, "Generative Models for Effective ML on Private, Decentralized Datasets," in *International Conference on Learning Representations*, 2019.

[68] K. Wei *et al.*, "Federated learning with differential privacy: Algorithms and performance analysis," *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 3454–3469, 2020.

[69] C. Zhao *et al.*, "Secure multi-party computation: Theory, practice and applications," *Information Sciences*, vol. 476, pp. 357–372, 2019.

[70] Y. Liu, Y. Kang, C. Xing, T. Chen, and Q. Yang, "A secure federated transfer learning framework," *IEEE Intelligent Systems*, vol. 35, no. 4, pp. 70–82, 2020.

[71] Y. Aono, T. Hayashi, L. Wang, S. Moriai, *et al.*, "Privacy-preserving deep learning via additively homomorphic encryption," *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 5, pp. 1333–1345, 2017.

[72] E. Bagdasaryan, A. Veit, Y. Hua, D. Estrin, and V. Shmatikov, "How to backdoor federated learning," in *International Conference on Artificial Intelligence and Statistics*, PMLR, 2020, pp. 2938–2948.

[73] A. N. Bhagoji, S. Chakraborty, P. Mittal, and S. Calo, "Analyzing federated learning through an adversarial lens," in *International Conference on Machine Learning*, PMLR, 2019, pp. 634–643.

[74] H. Kim, J. Park, M. Bennis, and S.-L. Kim, "Blockchained on-device federated learning," *IEEE Communications Letters*, vol. 24, no. 6, pp. 1279–1283, 2019.

[75] Q. Yang, Y. Liu, T. Chen, and Y. Tong, "Federated machine learning: Concept and applications," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 10, no. 2, pp. 1–19, 2019.

[76] M. Chen, N. Shlezinger, H. V. Poor, Y. C. Eldar, and S. Cui, "Communication-efficient federated learning," *Proceedings of the National Academy of Sciences*, vol. 118, no. 17, 2021.

[77] N. Guha, A. Talwalkar, and V. Smith, "One-shot federated learning," *arXiv preprint arXiv:1902.11175*, 2019.

[78] X. Yao, C. Huang, and L. Sun, "Two-stream federated learning: Reduce the communication costs," in *2018 IEEE Visual Communications and Image Processing (VCIP)*, IEEE, 2018, pp. 1–4.

[79] M. Yurochkin, M. Agarwal, S. Ghosh, K. Greenewald, N. Hoang, and Y. Khazaeni, "Bayesian nonparametric federated learning of neural networks," in *International Conference on Machine Learning*, PMLR, 2019, pp. 7252–7261.

[80] H. Li and T. Han, "An End-to-End Encrypted Neural Network for Gradient Updates Transmission in Federated Learning," in *2019 Data Compression Conference (DCC)*, IEEE, 2019, pp. 589–589.

[81] P. Jiang and L. Ying, "An optimal stopping approach for iterative training in federated learning," in *2020 54th Annual Conference on Information Sciences and Systems (CISS)*, IEEE, 2020, pp. 1–6.

[82] Y. Wang, Y. Xu, Q. Shi, and T.-H. Chang, "Quantized federated learning under transmission delay and outage constraints," *IEEE Journal on Selected Areas in Communications*, vol. 40, no. 1, pp. 323–341, 2021.

[83] T. Li, A. K. Sahu, M. Zaheer, M. Sanjabi, A. Talwalkar, and V. Smith, "Federated optimization in heterogeneous networks," *Proceedings of Machine Learning and Systems*, vol. 2, pp. 429–450, 2020.

[84] Y. Deng, M. M. Kamani, and M. Mahdavi, "Adaptive personalized federated learning," *arXiv preprint arXiv:2003.13461*, 2020.

[85] C. T. Dinh, N. Tran, and J. Nguyen, "Personalized federated learning with moreau envelopes," *Advances in Neural Information Processing Systems*, vol. 33, pp. 21 394–21 405, 2020.

[86] M. G. Arivazhagan, V. Aggarwal, A. K. Singh, and S. Choudhary, "Federated learning with personalization layers," *arXiv preprint arXiv:1912.00818*, 2019.

[87] E. Bakopoulou, B. Tillman, and A. Markopoulou, "A federated learning approach for mobile packet classification," *arXiv preprint arXiv:1907.13113*, 2019.

[88] H. Ludwig *et al.*, "IBM federated learning: An enterprise framework white paper v0.1," *arXiv preprint arXiv:2007.10987*, 2020.

[89] A. Soliman, S. Girdzijauskas, M.-R. Bouguelia, S. Pashami, and S. Nowaczyk, "Decentralized and adaptive k-means clustering for non-IID data using hyperloglog counters," in *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, Springer, 2020, pp. 343–355.

[90] M. Servetnyk, C. C. Fung, and Z. Han, "Unsupervised federated learning for unbalanced data," in *GLOBECOM 2020-2020 IEEE Global Communications Conference*, IEEE, 2020, pp. 1–6.

[91] X. Fan, Y. Ma, Z. Dai, W. Jing, C. Tan, and B. K. H. Low, "Fault-tolerant federated reinforcement learning with theoretical guarantee," *Advances in Neural Information Processing Systems*, vol. 34, pp. 1007–1021, 2021.

[92] K. Weiss, T. M. Khoshgoftaar, and D. Wang, "A survey of transfer learning," *Journal of Big data*, vol. 3, no. 1, pp. 1–40, 2016.

[93] D. Leroy, A. Coucke, T. Lavril, T. Gisselbrecht, and J. Dureau, "Federated learning for keyword spotting," in *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, IEEE, 2019, pp. 6341–6345.

[94] S. Ramaswamy, R. Mathews, K. Rao, and F. Beaufays, "Federated learning for emoji prediction in a mobile keyboard," *arXiv preprint arXiv:1906.04329*, 2019.

[95] J. Feng, C. Rong, F. Sun, D. Guo, and Y. Li, "PMF: A privacy-preserving human mobility prediction framework via federated learning," *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 4, no. 1, pp. 1–21, 2020.

[96] K. Sozinov, V. Vlassov, and S. Girdzijauskas, "Human activity recognition using federated learning," in *2018 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Ubiquitous Computing & Communications, Big Data & Cloud Computing, Social Computing & Networking, Sustainable Computing & Communications (ISPA/IUCC/BDCloud/SocialCom/SustainCom)*, IEEE, 2018, pp. 1103–1111.

[97] F. E. Casado, D. Lema, R. Iglesias, C. V. Regueiro, and S. Barro, "Concept drift detection and adaptation for robotics and mobile devices in federated and continual settings," in *Workshop of physical agents*, Springer, 2020, pp. 79–93.

[98] F. E. Casado, D. Lema, M. F. Criado, R. Iglesias, C. V. Regueiro, and S. Barro, "Concept drift detection and adaptation for federated and continual learning," *Multimedia Tools and Applications*, vol. 81, no. 3, pp. 3397–3419, 2021.

[99] L. Huang, A. L. Shea, H. Qian, A. Masurkar, H. Deng, and D. Liu, "Patient clustering improves efficiency of federated machine learning to predict mortality and hospital stay time using distributed electronic medical records," *Journal of biomedical informatics*, vol. 99, p. 103 291, 2019.

[100] S. Silva, B. A. Gutman, E. Romero, P. M. Thompson, A. Altmann, and M. Lorenzi, "Federated learning in distributed medical databases: Meta-analysis of large-scale subcortical brain data," in *2019 IEEE 16th international symposium on biomedical imaging (ISBI 2019)*, IEEE, 2019, pp. 270–274.

[101] D. Gao, C. Ju, X. Wei, Y. Liu, T. Chen, and Q. Yang, "HHHFL: Hierarchical heterogeneous horizontal federated learning for electroencephalography," *arXiv preprint arXiv:1909.05784*, 2019.

[102] J. Lee, J. Sun, F. Wang, S. Wang, C.-H. Jun, X. Jiang, *et al.*, "Privacy-preserving patient similarity learning in a federated environment: Development and analysis," *JMIR medical informatics*, vol. 6, no. 2, e7744, 2018.

[103] D. Liu, D. Dligach, and T. Miller, "Two-stage Federated Phenotyping and Patient Representation Learning," in *Proceedings of the conference. Association for Computational Linguistics. Meeting*, vol. 2019, 2019, pp. 283–291.

[104] X. Liang, Y. Liu, T. Chen, M. Liu, and Q. Yang, "Federated transfer reinforcement learning for autonomous driving," *arXiv preprint arXiv:1910.06001*, 2019.

[105] Y. Li, X. Tao, X. Zhang, J. Liu, and J. Xu, "Privacy-preserved federated learning for autonomous driving," *IEEE Transactions on Intelligent Transportation Systems*, 2021.

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

[107] G. I. Parisi, R. Kemker, J. L. Part, C. Kanan, and S. Wermter, "Continual lifelong learning with neural networks: A review," *Neural Networks*, 2019.

[108] Z. Chen and B. Liu, "Lifelong machine learning," *Synthesis Lectures on Artificial Intelligence and Machine Learning*, vol. 12, no. 3, pp. 1–207, 2018.

[109] A. Gepperth and B. Hammer, "Incremental learning algorithms and applications," in *European symposium on artificial neural networks (ESANN)*, 2016.

[110] S. Thrun and T. M. Mitchell, "Lifelong robot learning," *Robotics and autonomous systems*, vol. 15, no. 1-2, pp. 25–46, 1995.

[111] Z. Chen and B. Liu, "Topic modeling using topics from many domains, lifelong learning and big data," in *International conference on machine learning*, PMLR, 2014, pp. 703–711.

[112] C. Tessler, S. Givony, T. Zahavy, D. Mankowitz, and S. Mannor, "A deep hierarchical approach to lifelong learning in Minecraft," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 31, 2017.

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

[114] T. Mitchell *et al.*, "Never-ending learning," *Communications of the ACM*, vol. 61, no. 5, pp. 103–115, 2018.

[115] S. A. Rebuffi, A. Kolesnikov, G. Sperl, and C. H. Lampert, "iCaRL: Incremental classifier and representation learning," in *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, 2017, pp. 2001–2010.

[116] M. B. Ring *et al.*, "Continual learning in reinforcement environments," 1994.

[117] S. Thrun, "Is learning the n-th thing any easier than learning the first?" *Advances in neural information processing systems*, vol. 8, 1995.

[118] D. L. Silver and R. E. Mercer, "The parallel transfer of task knowledge using dynamic learning rates based on a measure of relatedness," in *Learning to learn*, Springer, 1996, pp. 213–233.

[119] D. L. Silver and R. E. Mercer, "The task rehearsal method of life-long learning: Overcoming impoverished data," in *Conference of the Canadian Society for Computational Studies of Intelligence*, Springer, 2002, pp. 90–101.

[120] M. Wang and C. Wang, "Learning from adaptive neural dynamic surface control of strict-feedback systems," *IEEE transactions on neural networks and learning systems*, vol. 26, no. 6, pp. 1247–1259, 2014.

[121] S. Wang, Z. Chen, and B. Liu, "Mining aspect-specific opinion using a holistic lifelong topic model," in *Proceedings of the 25th international conference on world wide web*, 2016, pp. 167–176.

[122] Q. Liu, B. Liu, Y. Zhang, D. S. Kim, and Z. Gao, "Improving opinion aspect extraction using semantic similarity and aspect associations," in *Thirtieth AAAI conference on artificial intelligence*, 2016.

[123] Z. K. Malik, A. Hussain, and J. Wu, "An online generalized eigenvalue version of laplacian eigenmaps for visual big data," *Neurocomputing*, vol. 173, pp. 127–136, 2016.

[124] F. Tanaka and M. Yamamura, "An approach to lifelong reinforcement learning through multiple environments," in *6th European Workshop on Learning Robots*, 1997, pp. 93–99.

[125] M. B. Ring, "CHILD: A first step towards continual learning," in *Learning to learn*, Springer, 1998, pp. 261–292.

[126] F. Fernández and M. Veloso, "Learning domain structure through probabilistic policy reuse in reinforcement learning," *Progress in Artificial Intelligence*, vol. 2, no. 1, pp. 13–27, 2013.

[127] M. P. Deisenroth, P. Englert, J. Peters, and D. Fox, "Multi-task policy search for robotics," in *2014 IEEE international conference on robotics and automation (ICRA)*, IEEE, 2014, pp. 3876–3881.

[128] Y. Amirat, D. Daney, S. Mohammed, A. Spalanzani, A. Chibani, and O. Simonin, "Assistance and service robotics in a human environment," *Robotics and Autonomous Systems*, vol. 75, no. PA, pp. 1–3, 2016.

[129] V. Lomonaco and D. Maltoni, "Core50: a new dataset and benchmark for continuous object recognition," in *Conference on Robot Learning*, PMLR, 2017, pp. 17–26.

[130] A. Mozaffari, M. Vajedi, and N. L. Azad, "A robust safety-oriented autonomous cruise control scheme for electric vehicles based on model predictive control and online sequential extreme learning machine with a hyper-level fault tolerance-based supervisor," *Neurocomputing*, vol. 151, pp. 845–856, 2015.

[131] M. A. A. Dewan, E. Granger, G.-L. Marcialis, R. Sabourin, and F. Roli, "Adaptive appearance model tracking for still-to-video face recognition," *Pattern recognition*, vol. 49, pp. 129–151, 2016.

[132] L. Liu, X. Bai, H. Zhang, J. Zhou, and W. Tang, "Describing and learning of related parts based on latent structural model in big data," *Neurocomputing*, vol. 173, pp. 355–363, 2016.

[133] H. C. Carneiro, F. M. França, and P. M. Lima, "Multilingual part-of-speech tagging with weightless neural networks," *Neural Networks*, vol. 66, pp. 11–21, 2015.

[134] G. Yin, Y.-T. Zhang, Z.-N. Li, G.-Q. Ren, and H.-B. Fan, "Online fault diagnosis method based on incremental support vector data description and extreme learning machine with incremental output structure," *Neurocomputing*, vol. 128, pp. 224–231, 2014.

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

[136] S. Grossberg, "Nonlinear neural networks: Principles, mechanisms, and architectures," *Neural networks*, vol. 1, no. 1, pp. 17–61, 1988.

[137] R. M. French, "Catastrophic forgetting in connectionist networks," *Trends in cognitive sciences*, vol. 3, no. 4, pp. 128–135, 1999.

[138] J. Gama, I. Žliobaitė, A. Bifet, M. Pechenizkiy, and A. Bouchachia, "A survey on concept drift adaptation," *ACM computing surveys (CSUR)*, vol. 46, no. 4, pp. 1–37, 2014.

[139] J. Lu, A. Liu, F. Dong, F. Gu, J. Gama, and G. Zhang, "Learning under concept drift: A review," *IEEE Transactions on Knowledge and Data Engineering*, vol. 31, no. 12, pp. 2346–2363, 2018.

[140] I. Khamassi, M. Sayed-Mouchaweh, M. Hammami, and K. Ghédira, "Discussion and review on evolving data streams and concept drift adapting," *Evolving systems*, vol. 9, no. 1, pp. 1–23, 2018.

[141] G. Ditzler and R. Polikar, "Incremental learning of concept drift from streaming imbalanced data," *IEEE transactions on knowledge and data engineering*, vol. 25, no. 10, pp. 2283–2301, 2012.

[142] J. C. Schlimmer and R. H. Granger, "Incremental learning from noisy data," *Machine learning*, vol. 1, no. 3, pp. 317–354, 1986.

[143] M. S. Hammoodi, F. Stahl, and A. Badii, "Real-time feature selection technique with concept drift detection using adaptive micro-clusters for data stream mining," *Knowledge-Based Systems*, vol. 161, pp. 205–239, 2018.

[144] J. Shao, Z. Ahmadi, and 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.

[145] A. Dries and U. Rückert, "Adaptive concept drift detection," *Statistical Analysis and Data Mining: The ASA Data Science Journal*, vol. 2, no. 5-6, pp. 311–327, 2009.

[146] J. Gama, P. Medas, G. Castillo, and P. Rodrigues, "Learning with drift detection," in *Brazilian symposium on artificial intelligence*, Springer, 2004, pp. 286–295.

[147] I. Frias-Blanco, J. del Campo-Ávila, G. Ramos-Jimenez, R. Morales-Bueno, A. Ortiz-Diaz, and Y. Caballero-Mota, "Online and non-parametric drift detection methods based on Hoeffding's bounds," *IEEE Transactions on Knowledge and Data Engineering*, vol. 27, no. 3, pp. 810–823, 2014.

[148] A. Bifet and R. Gavalda, "Adaptive learning from evolving data streams," in *International Symposium on Intelligent Data Analysis*, Springer, 2009, pp. 249–260.

[149] K. Nishida and K. Yamauchi, "Detecting concept drift using statistical testing," in *International conference on discovery science*, Springer, 2007, pp. 264–269.

[150] S. H. Bach and M. A. Maloof, "Paired learners for concept drift," in *2008 Eighth IEEE International Conference on Data Mining*, IEEE, 2008, pp. 23–32.

[151] A. Bifet and R. Gavalda, "Learning from time-changing data with adaptive windowing," in *Proceedings of the 2007 SIAM international conference on data mining*, SIAM, 2007, pp. 443–448.

[152] Y. Sun, K. Tang, Z. Zhu, and X. Yao, "Concept drift adaptation by exploiting historical knowledge," *IEEE transactions on neural networks and learning systems*, vol. 29, no. 10, pp. 4822–4832, 2018.

[153] A. Bifet, G. Holmes, and B. Pfahringer, "Leveraging bagging for evolving data streams," in *Joint European conference on machine learning and knowledge discovery in databases*, Springer, 2010, pp. 135–150.

[154] F. Chu and C. Zaniolo, "Fast and light boosting for adaptive mining of data streams," in *Pacific-Asia conference on knowledge discovery and data mining*, Springer, 2004, pp. 282–292.

[155] P. Domingos and G. Hulten, "Mining high-speed data streams," in *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, 2000, pp. 71–80.

[156] L. Rutkowski, M. Jaworski, L. Pietruczuk, and P. Duda, "A new method for data stream mining based on the misclassification error," *IEEE transactions on neural networks and learning systems*, vol. 26, no. 5, pp. 1048–1059, 2014.

[157] J. Yoon, W. Jeong, G. Lee, E. Yang, and S. J. Hwang, "Federated continual learning with weighted inter-client transfer," in *International Conference on Machine Learning*, PMLR, 2021, pp. 12 073–12 086.

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

[159] T. J. Park, K. Kumatani, and D. Dimitriadis, "Tackling dynamics in federated incremental learning with variational embedding rehearsal," *arXiv preprint arXiv:2110.09695*, 2021.

[160] G. Canonaco, A. Bergamasco, A. Mongelluzzo, and M. Roveri, "Adaptive federated learning in presence of concept drift," in *2021 International Joint Conference on Neural Networks (IJCNN)*, IEEE, 2021, pp. 1–7.

[161] E. Jothimurugesan, K. Hsieh, J. Wang, G. Joshi, and P. B. Gibbons, "Federated learning under distributed concept drift," *arXiv preprint arXiv:2206.00799*, 2022.

[162] M. Baron, "Convergence rates of change-point estimators and tail probabilities of the first-passage-time process," *Canadian Journal of Statistics*, vol. 27, no. 1, pp. 183–197, 1999.

[163] A. Haque, L. Khan, and M. Baron, "SAND: Semi-supervised adaptive novel class detection and classification over data stream," in *Thirtieth AAAI Conference on Artificial Intelligence*, 2016, pp. 1652–1658.

[164] K. Bowman and L. Shenton, "Estimation: Method of moments," *Wiley StatsRef: Statistics Reference Online*, 2014.

[165] G. M. van de Ven and A. S. Tolias, "Three scenarios for continual learning," *arXiv preprint arXiv:1904.07734*, 2019.

[166] S. J. Raudys and A. K. Jain, "Small sample size effects in statistical pattern recognition: Recommendations for practitioners," *IEEE Transactions on Pattern Analysis & Machine Intelligence*, vol. 13, no. 3, pp. 252–264, 1991.

[167] L. Armijo, "Minimization of functions having lipschitz continuous first partial derivatives," *Pacific Journal of mathematics*, vol. 16, no. 1, pp. 1–3, 1966.

[168] P. Kairouz *et al.*, "Advances and open problems in federated learning," *Foundations and Trends® in Machine Learning*, vol. 14, no. 1–2, pp. 1–210, 2021.

[169] J. Konečný, H. B. McMahan, D. Ramage, and P. Richtárik, "Federated optimization: Distributed machine learning for on-device intelligence," *arXiv preprint arXiv: 1610.02527*, 2016.

[170] M. Shoaib, S. Bosch, O. D. Incel, H. Scholten, and P. J. Havinga, "Fusion of smartphone motion sensors for physical activity recognition," *Sensors*, vol. 14, no. 6, pp. 10 146–10 176, 2014.

[171] F. E. Casado, G. Rodríguez, R. Iglesias, C. V. Regueiro, S. Barro, and A. Canedo-Rodríguez, "Walking recognition in mobile devices," *Sensors*, vol. 20, no. 4, p. 1189, 2020.

[172] L. N. Tong, J. J. He, and L. Peng, "CNN-based PD hand tremor detection using inertial sensor," *IEEE Sensors Letters*, 2021.

[173] F. E. Casado, D. Lema, R. Iglesias, C. V. Regueiro, and S. Barro, "Federated and continual learning for classification tasks in a society of devices," *arXiv preprint arXiv:2006.07129*, 2020.

[174] R. Schwartz, J. Dodge, N. A. Smith, and O. Etzioni, "Green AI," *Communications of the ACM*, vol. 63, no. 12, pp. 54–63, 2020.

[175] A. B. Arrieta *et al.*, "Explainable Artificial Intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI," *Information Fusion*, vol. 58, pp. 82–115, 2020.

[176] M. B. Fazi, "Beyond human: Deep learning, explainability and representation," *Theory, Culture & Society*, vol. 38, no. 7-8, pp. 55–77, 2021.

[177] N. O'Mahony *et al.*, "Deep learning vs. traditional computer vision," in *Science and Information Conference*, Springer, 2019, pp. 128–144.

[178] I. Androutsopoulos, J. Koutsias, K. V. Chandrinos, G. Paliouras, and C. D. Spyropoulos, "An evaluation of naive bayesian anti-spam filtering," in *Proc. of the Workshop on Machine Learning in the New Information Age, 11th European Conference on Machine Learning (ECML 2000), May*, 2000.

[179] E. Strubell, A. Ganesh, and A. McCallum, "Energy and Policy Considerations for Deep Learning in NLP," in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 2019, pp. 3645–3650.

[180] T. G. Dietterich *et al.*, "Ensemble learning," *The handbook of brain theory and neural networks*, vol. 2, pp. 110–125, 2002.

[181] Z.-H. Zhou, "Ensemble learning," *Encyclopedia of biometrics*, vol. 1, pp. 270–273, 2009.

[182] J. Hamer, M. Mohri, and A. T. Suresh, "FedBoost: A Communication-Efficient Algorithm for Federated Learning," in *International Conference on Machine Learning*, PMLR, 2020, pp. 3973–3983.

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

[184] J. Czyz, J. Kittler, and L. Vandendorpe, "Multiple classifier combination for face-based identity verification," *Pattern recognition*, vol. 37, no. 7, pp. 1459–1469, 2004.

[185] J. Kittler, M. Hatef, R. P. Duin, and J. Matas, "On combining classifiers," *IEEE transactions on pattern analysis and machine intelligence*, vol. 20, no. 3, pp. 226–239, 1998.

[186] K. Tumer and J. Ghosh, "Error correlation and error reduction in ensemble classifiers," *Connection science*, vol. 8, no. 3-4, pp. 385–404, 1996.

[187] D. H. Wolpert, "Stacked generalization," *Neural networks*, vol. 5, no. 2, pp. 241–259, 1992.

[188] G. Tsoumakas, L. Angelis, and I. Vlahavas, "Clustering classifiers for knowledge discovery from physically distributed databases," *Data & Knowledge Engineering*, vol. 49, no. 3, pp. 223–242, 2004.

[189] M. Nasr, R. Shokri, and A. Houmansadr, "Comprehensive privacy analysis of deep learning: Passive and active white-box inference attacks against centralized and federated learning," in *2019 IEEE symposium on security and privacy (SP)*, IEEE, 2019, pp. 739–753.

[190] Z. Wang, M. Song, Z. Zhang, Y. Song, Q. Wang, and H. Qi, "Beyond inferring class representatives: User-level privacy leakage from federated learning," in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*, IEEE, 2019, pp. 2512–2520.

[191] C. Dwork, "Differential privacy: A survey of results," in *International conference on theory and applications of models of computation*, Springer, 2008, pp. 1–19.

[192] R. Canetti, U. Feige, O. Goldreich, and M. Naor, "Adaptively secure multi-party computation," in *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, 1996, pp. 639–648.

[193] F. E. Casado and Y. Demiris, "Federated learning from demonstration for active assistance to smart wheelchair users," in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2022.

[194] J. Leaman and H. M. La, "A comprehensive review of smart wheelchairs: Past, present, and future," *IEEE Transactions on Human-Machine Systems*, vol. 47, no. 4, pp. 486–499, 2017.

[195] H. Ravichandar, A. S. Polydoros, S. Chernova, and A. Billard, "Recent advances in robot learning from demonstration," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 3, pp. 297–330, 2020.

[196] M. Verleysen and D. François, "The curse of dimensionality in data mining and time series prediction," in *International work-conference on artificial neural networks*, Springer, 2005, pp. 758–770.

[197] H. Soh and Y. Demiris, "When and how to help: An iterative probabilistic model for learning assistance by demonstration," in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE, 2013, pp. 3230–3236.

[198] H. Soh and Y. Demiris, "Learning assistance by demonstration: Smart mobility with shared control and paired haptic controllers," *Journal of Human-Robot Interaction*, vol. 4, no. 3, pp. 76–100, 2015.

[199] A. Hüntemann, E. Demeester, E. Vander Poorten, H. Van Brussel, and J. De Schutter, "Probabilistic approach to recognize local navigation plans by fusing past driving information with a personalized user model," in *2013 IEEE International Conference on Robotics and Automation*, IEEE, 2013, pp. 4376–4383.

[200] T. Matsubara, J. V. Miro, D. Tanaka, J. Poon, and K. Sugimoto, "Sequential intention estimation of a mobility aid user for intelligent navigational assistance," in *2015 24th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN)*, IEEE, 2015, pp. 444–449.

[201] R. C. Simpson and S. P. Levine, "Automatic adaptation in the NavChair assistive wheelchair navigation system," *IEEE Transactions on Rehabilitation Engineering*, vol. 7, no. 4, pp. 452–463, 1999.

[202] T. Carlson and Y. Demiris, "Human-wheelchair collaboration through prediction of intention and adaptive assistance," in *2008 IEEE International Conference on Robotics and Automation*, IEEE, 2008, pp. 3926–3931.

[203] H. N. Chow and Y. Xu, "Learning human navigational skill for smart wheelchair in a static cluttered route," *IEEE Transactions on Industrial Electronics*, vol. 53, no. 4, pp. 1350–1361, 2006.

[204] J. Poon, Y. Cui, J. V. Miro, T. Matsubara, and K. Sugimoto, "Local driving assistance from demonstration for mobility aids," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2017, pp. 5935–5941.

[205] J. Poon, Y. Cui, J. V. Miro, and T. Matsubara, "Learning from demonstration for locally assistive mobility aids," *International Journal of Intelligent Robotics and Applications*, vol. 3, no. 3, pp. 255–268, 2019.

[206] Z. Li, L. Wang, L. Jiang, and C.-Z. Xu, "FC-SLAM: Federated learning enhanced distributed visual-LiDAR SLAM in cloud robotic system," in *2019 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, IEEE, 2019, pp. 1995–2000.

[207] G. T. Papadopoulos, A. Leonidis, M. Antona, and C. Stephanidis, "User profile-driven large-scale multi-agent learning from demonstration in federated human-robot collaborative environments," *arXiv preprint arXiv:2103.16434*, 2021.

[208] J. Canny, "A computational approach to edge detection," *IEEE Transactions on pattern analysis and machine intelligence*, no. 6, pp. 679–698, 1986.

[209] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

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

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

[212] P. Roy, S. Ghosh, S. Bhattacharya, and U. Pal, "Effects of degradations on deep neural network architectures," *arXiv preprint arXiv:1807.10108*, 2018.

[213] F. M. Carlucci, P. Russo, T. Tommasi, and B. Caputo, "Hallucinating agnostic images to generalize across domains," in *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)*, IEEE, 2019, pp. 3227–3234.

[214] X. Peng, Q. Bai, X. Xia, Z. Huang, K. Saenko, and B. Wang, "Moment matching for multi-source domain adaptation," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2019, pp. 1406–1415.

[215] S. Dutta, A. Chatterjee, and S. Munshi, "An automated hierarchical gait pattern identification tool employing cross-correlation-based feature extraction and recurrent neural network based classification," *Expert systems*, vol. 26, no. 2, pp. 202–217, 2009.

[216] C. Zhu and W. Sheng, "Recognizing human daily activity using a single inertial sensor," in *Intelligent Control and Automation (WCICA), 2010 8th World Congress on*, Citeseer, Jinan, China, Jul. 2010, pp. 282–287.

[217] M. J. Mathie, A. C. Coster, N. H. Lovell, and B. G. Celler, "Accelerometry: Providing an integrated, practical method for long-term, ambulatory monitoring of human movement," *Physiological measurement*, vol. 25, no. 2, R1, 2004.

[218] Y. Ren, Y. Chen, M. C. Chuah, and J. Yang, "User verification leveraging gait recognition for smartphone enabled mobile healthcare systems," *IEEE Transactions on Mobile Computing*, vol. 14, no. 9, pp. 1961–1974, 2015.

[219] R. Harle, "A survey of indoor inertial positioning systems for pedestrians," *IEEE Communications Surveys & Tutorials*, vol. 15, no. 3, pp. 1281–1293, 2013.

[220] G. Rodríguez, F. E. Casado, R. Iglesias, C. V. Regueiro, and A. Nieto, "Robust step counting for inertial navigation with mobile phones," *Sensors*, vol. 18, no. 9, p. 3157, 2018.

[221] M. Kuhn *et al.*, "Package 'caret'," *The R Journal*, 2020.

Appendix A

Extended results and additional experiments

In Chapters 3 and 4 we presented CDA-FedAvg and ECFL, respectively. In both cases, we performed an experimental evaluation comparing them with other state-of-the-art methods. For ease of reading, we summarized or omitted several details related to data collection, preprocessing, implementation, and results. In the same way, we focused on evaluating the proposals on a single task. In the present appendix, we provide all the information that was not included in those central chapters. In addition, we report further results obtained when evaluating the algorithms on other tasks and datasets. The supplementary descriptions included here help to understand in depth how each of the experiments was conducted. The extended and additional results provide further evidence for the robustness, performance improvements, and cost reduction that we achieve with CDA-FedAvg and ECFL.

Some of the contents of this appendix have been reproduced under CC BY license from the following publications (see Appendix B, Figures B.1, B.2, and B.3):

[15] M. F. Criado*, F. E. Casado*, R. Iglesias*, C. V. Regueiro†, and S. Barro*. "Non-IID data and Continual Learning processes in Federated Learning: A long road ahead," *Information Fusion*, Elsevier, vol. 88, pp. 263–280, 2022. DOI: 10.1016/j.inffus.2022.07.024. ISSN: 1566-2535.

> [98] F. E. Casado*, D. Lema*, M. F. Criado*, R. Iglesias*, C. V. Regueiro†, and S. Barro*. "Concept drift detection and adaptation for federated and continual learning," *Multimedia Tools and Applications*, Springer, vol. 81, no. 3, pp. 3397–3419, 2021. DOI: 10.1007/s11042-021-11219-x. ISSN: 1380-7501.

> [171] F. E. Casado*, G. Rodríguez‡, R. Iglesias*, C. V. Regueiro†, S. Barro*, and A. Canedo-Rodríguez‡. "Walking recognition in Mobile Devices," *Sensors*, MDPI, vol. 20, no. 4, 1189, 2020. DOI: 10.3390/s20041189. ISSN: 1424-8220.

> [173] F. E. Casado*, D. Lema*, R. Iglesias*, C. V. Regueiro†, and S. Barro*. "Ensemble and continual federated learning for classification tasks," *Machine Learning*, Springer. ISSN: 0885-6125.
> This paper is currently under review (second round).

The appendix is structured as follows: Section A.1 extends the results obtained with CDA-FedAvg on HAR. Section A.2 provides additional experiments conducted on the task of digit recognition, comparing CDA-FedAvg with FedAvg. Section A.3 presents the extended results of the performance of ECFL on the walking recognition task, as well as more details on how the data were collected and preprocessed. In Section A.4, we evaluate ECFL in the HAR multi-class problem. Finally, Section A.5 provides additional details related to software, hardware, and other implementation matters.

A.1 CDA-FedAvg: Extended results on activity recognition

In the following we provide an extended version of the results from Section 3.6, where we evaluated both FedAvg and CDA-FedAvg using the Shoaib's HAR dataset [170]:

*Centro Singular de Investigación en Tecnoloxías Intelixentes (CiTIUS), Universidade de Santiago de Compostela, 15782 Santiago de Compostela, Spain.
†CITIC, Computer Architecture Group, Universidade da Coruña, 15071 A Coruña, Spain.
‡Situm Technologies S.L., 15782 Santiago de Compostela, Spain.

- Table A.1 extends Table 3.1. It shows the final accuracies achieved when running FedAvg in ideal conditions, so that all clients collect 5000 data samples in a stationary and IID manner. Each of the rows provide the results of each execution, training with 9 of the 10 clients and testing on the dataset of the remaining one. For each execution, we provide the accuracy we got at the en of the process for each of the phone position (belt, pockets, etc.), as well as the average.

- Table A.2 extends Table 3.2. In this case, it reports the results obtained when evaluating FedAvg on the non-stationary setting in which the users gradually change the position of the phone (from the belt to the left pocket, from the left to the right one, etc.). In total, there are 4 shifts in the distribution of the data. The table provides the final accuracies at the end of the process.

- Finally, Table A.3 extends Table 3.3. It provides the full results of evaluating our method, CDA-FedAvg in the previous scenario.

Table A.1: Final global accuracies obtained for all the executions of standard FedAvg in a stationary and IID setting, after all clients have processed 5000 data samples.

	Accuracy					
Test set	Belt	Left pocket	Right pocket	Upper arm	Wrist	Overall
User 1	0.878	0.911	0.862	0.909	0.802	0.872
User 2	0.726	0.981	0.982	0.720	0.803	0.842
User 3	0.891	0.951	0.898	0.741	0.898	0.876
User 4	0.906	0.982	0.984	0.555	0.909	0.867
User 5	0.925	0.981	0.984	0.753	0.919	0.912
User 6	0.672	0.981	0.970	0.709	0.824	0.831
User 7	0.606	0.834	0.977	0.915	0.776	0.822
User 8	0.881	0.993	0.992	0.673	0.924	0.893
User 9	0.445	0.992	0.989	0.635	0.819	0.776
User 10	0.744	0.711	0.907	0.791	0.910	0.813
Average	0.767	0.932	0.955	0.740	0.858	0.850

Table A.2: Final global accuracies for all the executions of FedAvg in a non-stationary setting, after all clients have processed 5000 data samples.

	Accuracy					
Test set	Belt	Left pocket	Right pocket	Upper arm	Wrist	Overall
User 1	0.349	0.725	0.626	0.586	0.776	0.612
User 2	0.358	0.598	0.735	0.633	0.773	0.619
User 3	0.498	0.375	0.780	0.610	0.758	0.604
User 4	0.359	0.659	0.642	0.404	0.873	0.587
User 5	0.349	0.666	0.807	0.706	0.978	0.701
User 6	0.352	0.531	0.758	0.460	0.936	0.607
User 7	0.329	0.547	0.921	0.644	0.926	0.673
User 8	0.375	0.531	0.873	0.525	0.934	0.648
User 9	0.144	0.642	0.799	0.567	0.950	0.620
User 10	0.505	0.415	0.913	0.634	0.830	0.659
Average	0.362	0.569	0.785	0.577	0.873	0.633

Table A.3: Final global accuracies for all the executions of CDA-FedAvg in a non-stationary setting, after all clients have processed 5000 data samples.

	Accuracy					
Test set	Belt	Left pocket	Right pocket	Upper arm	Wrist	Overall
User 1	0.832	0.950	0.777	0.771	0.864	0.839
User 2	0.786	0.955	0.669	0.691	0.797	0.780
User 3	0.834	0.894	0.838	0.771	0.809	0.829
User 4	0.936	0.983	0.971	0.613	0.908	0.884
User 5	0.972	0.990	0.992	0.855	0.676	0.897
User 6	0.776	0.941	0.722	0.712	0.908	0.812
User 7	0.585	0.668	0.969	0.864	0.909	0.799
User 8	0.821	0.980	0.987	0.698	0.912	0.878
User 9	0.294	0.964	0.982	0.591	0.884	0.743
User 10	0.748	0.749	0.561	0.684	0.883	0.725
Average	0.758	0.907	0.847	0.725	0.855	0.819

A.2 CDA-FedAvg: Additional experiments on digit recognition

We further provide additional results comparing CDA-FedAvg with FedAvg in a different task and data. We chose one of the most usual benchmarks in computer vision: digit recognition in images. In order to force non-stationary data streams, we need a variety of images. For

that, one option would have been to take a classic dataset, such as MNIST [209], and perform several modifications by adding different types of noise to the images. However, to make the experiment more realistic, we considered it better to combine data from different sources. In this way, we used up to five different public datasets, including:

1. **MNIST** (Figure A.1a) [209]. This dataset includes 28×28 grayscale images of digits from 0 to 9, which were handwritten by 250 different people and then scanned. The total number of samples amounts to $70,000$.

2. **MNIST-M** (Figure A.1b) [210]. It was created by combining the original MNIST digits with random colored patches. It contains a total of $149,002$ images of 28×28 pixels.

3. **SVHN** (Figure A.1c) [211]. This is a real-world dataset, consisting of 32×32 pictures obtained from house numbers in Google Street View. If we compare it with MNIST, SVHN incorporates an order of magnitude more labeled data ($630,420$ digit images) and comes from a harder, real world problem (recognizing numbers in natural scene images).

4. **Synthetic Digits** (Figure A.1d) [212]. This is the synthetic counterpart of SVHN. It consist of $488,953$ samples of digits embedded on random and noisy backgrounds. The images are 32×32 and were generated with varying fonts, colors and rotations.

5. **USPS** (Figure A.1e) [213]. It contains $9,298$ images of digits scanned from envelopes by the U.S. Postal Service. Samples are centered, normalized and show a broad range of font styles. All of them are grayscale, with a resolution of 16×16.

(a) MNIST. **(b)** MNISTM. **(c)** SVHN. **(d)** Synth. Digits. **(e)** USPS.

Figure A.1: Some examples of images from each of the datasets used for the experiments.

In the literature, the aggregation of these five datasets is often referred to as the *Digit-five* dataset [214]. As can be seen in Figure A.1, each of them differs greatly from the others. When combining them, we resized all images to 32×32 pixels. Besides, we balanced all the sources

by keeping $60,000$ samples from each one. In the case of USPS, we applied oversampling. Thus, we ended up with a fully labeled dataset of $300,000$ patterns. To recreate a federated setting, we distributed the data among 50 artificial clients, assigning $6,000$ samples to each one ($1,200$ per source). We used 35 of these 50 clients for training, and left the data from the other 15 for testing.

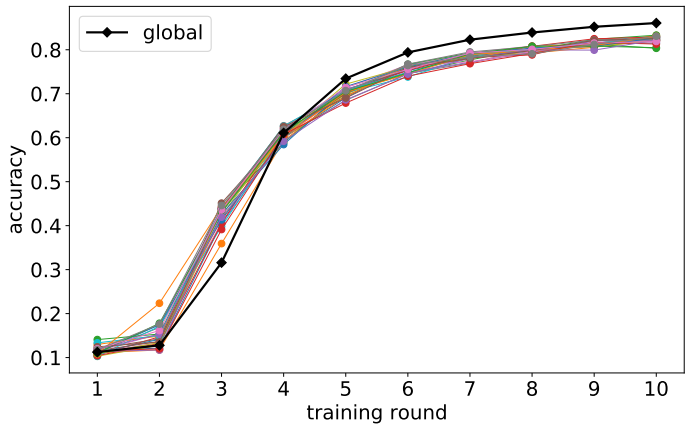
Similar to what we did with HAR in Section 3.6, we performed experiments in two different CFL scenarios. The model architecture we employed was the same in all of them, consisting of a simple CNN with 4 convolutional layers followed by 3 dense layers (see Section A.5 for implementation details). We ran each experiment multiple times to make sure the results were statistically significant and no artifacts had been produced. Recall that FedAvg was not originally designed to deal with continual problems, but with static datasets. Hence, to adapt it to CFL, enabling it to process data continuously while performing the learning rounds. For simplicity, we allow the algorithm to keep the entire data stream in memory, without having to forget old samples. This places CDA-FedAvg in disadvantage, since it does restrict the storage. In spite of this, as we will see below, our method provides very good results.

Scenario A: Stationary baseline

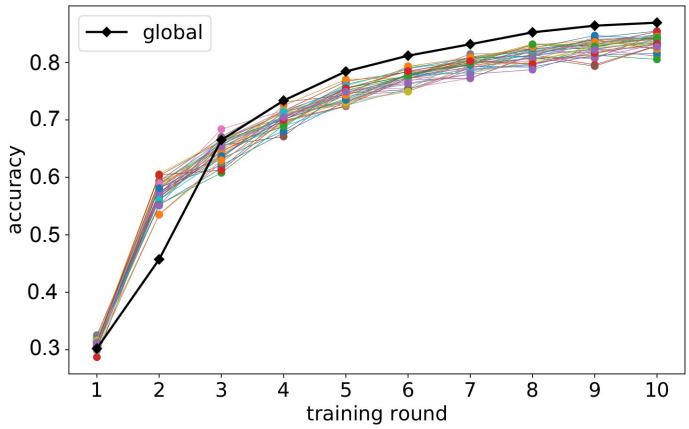
In this first experiment, we randomly shuffled the data of each client. Therefore, the different sources were all mixed together and all the data streams were IID and stationary. We executed FedAvg and CDA-FedAvg forcing a gradual acquisition of data and allowing them to run a maximum of 10 learning rounds until they reached the end of the data stream. Figure A.2 shows the average results for both methods. The thin colored lines indicate the accuracy of each of the local versions of the model after the parameter update. The thick black line represents the accuracy after the global aggregation. As can be seen, in this scenario both FedAvg and CDA-FedAvg perform similarly, reaching an accuracy of approximately 84% in both cases.

Scenario B: Non-stationary stream under virtual drift

In this other setting, we forced virtual concept drifts in all of the clients simultaneously. For that, instead of shuffle the data, we ordered it by source: The first 1200 samples are from MNIST, the next $1,200$ are from MNIST-M, and so on. Hence, the local data streams were non-stationary. Again, we ran both FedAvg and FedAvg. Figure A.3 shows the results. In this



(a) FedAvg.



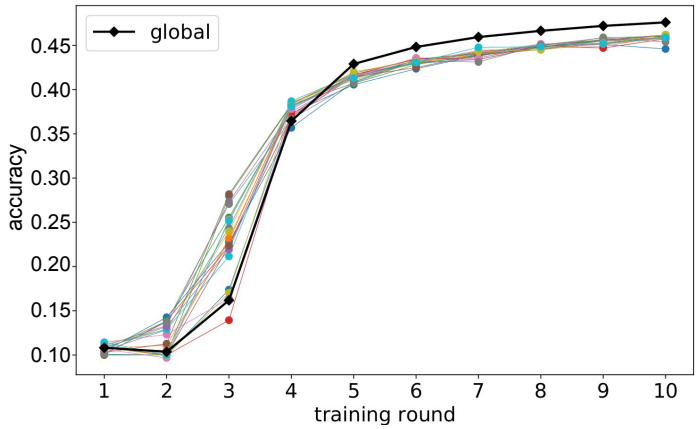
(b) CDA-FedAvg.

Figure A.2: Performances of FedAvg and CDA-FedAvg in an ideal (stationary, IID) scenario. The thin colored lines show the accuracy after each local update, while the thick black one indicates the accuracy after the global aggregation.

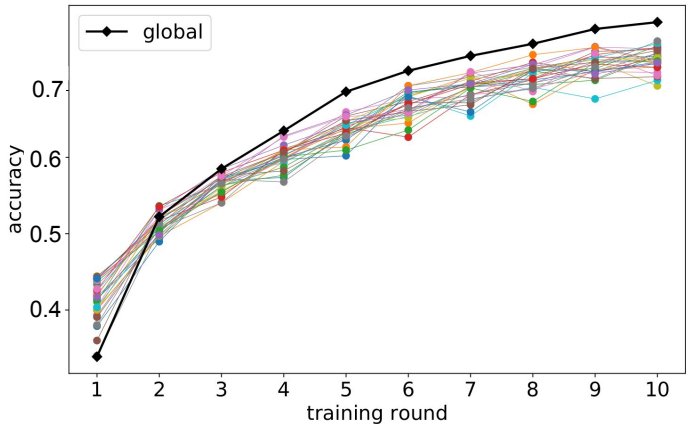
case, we can appreciate how FedAvg is not able to learn gradually in the face of concept drifts, lowering its accuracy to 48%. On the other hand, CDA-FedAvg can perfectly handle this situation, providing similar results to those achieved in the stationary case, approaching 80%.

A.3 ECFL: Extended results on walking recognition

In this section we provide further details on the experiments performed with ECFL on the task of walking recognition, which were omitted in Section 4.5 for ease of reading.



(a) Virtual drift – FedAvg.



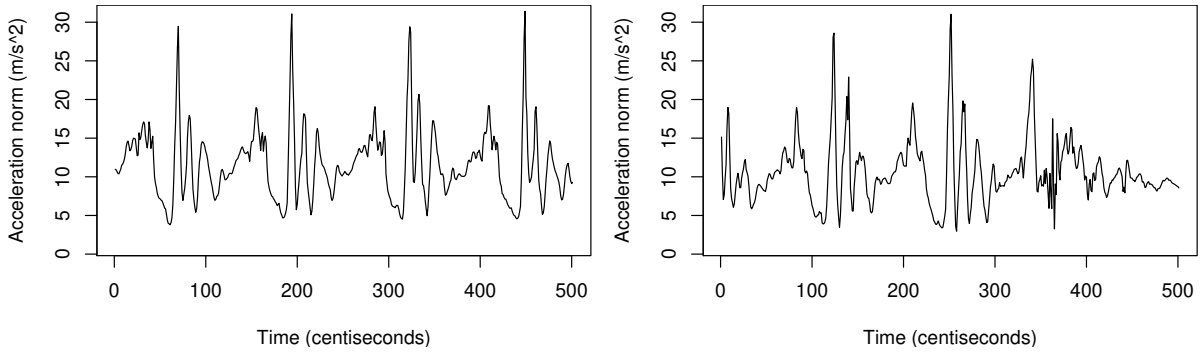
(b) Virtual drift – CDA-FedAvg.

Figure A.3: Performances of FedAvg and CDA-FedAvg in the different Temporal non-IID Scenarios. The black thick line represents the global model accuracy, whereas the other ones represent the accuracy of the model of each client.

A.3.1 Relevance and complexity of the task

Knowing when a person is walking or not is of great utility in many contexts. Some examples are medical diagnosis [215], elderly assistance [216], monitoring systems [217], biometric identification [218], or indoor localization [219]. Several strategies involving different hardware have been explored to monitor this activity, such as cameras for visual odometry or pressure sensors attached to the shoes. Nevertheless, the placement of sensors on the body or clothing greatly restricts their applicability. Instead, using the sensors integrated in the smartphone (accelerometer, gyroscope and magnetometer) is much more appealing, since very little physical infrastructure is required and more and more people are carrying one of these devices.

Achieving a robust recognition system using the smartphone is more complex than it might seem. It is relatively easy to identify when a person walks carrying the smartphone in a fixed position. However, in real life, the orientation of the device with respect to the body, as well as its location (hand, bag, pocket, etc.), may change constantly as the person moves, thus making the task more challenging. Figure A.4 illustrates this by showing the acceleration norm experienced by a smartphone while the owner is doing two different activities. The signal presented in Figure A.4a corresponds to the person walking with the device in the pocket. On the other hand, Figure A.4b reflects a situation in which the user is standing still, but gesticulating while holding the phone. The person and the device are the same in both cases.



(a) The user is walking with the device inside the pocket.

(b) The user is not walking, but gesticulating with the device in the hand.

Figure A.4: Acceleration norm experienced by the smartphone in two very different situations.

Some works have addressed walking recognition by applying standard and centralized ML processes [171, 220]. However, these approaches involve months of work, collecting data from different volunteers—with the privacy restrictions that this may entail—, re-training and fine-tuning a model until it is finally put into exploitation. Besides, no matter how complete we think the training data is, there will often be a situation, a user, or a device, for which the model fails to generalize well enough. A CFL approach can help avoid such issues.

A.3.2 Data collection, preprocessing and distribution

The following explains how the data used in the experiments of Section 4.5 was obtained. We did an important effort to collect data in a wide variety of cases or situations and from as many people as possible. The process was performed in two stages: the first one for gathering the data used for testing, and the second one for that used for training. Both the training and

testing datasets contain tri-axis accelerometer and gyroscope raw signals recorded at 100 Hz by different smartphones while their users performed different activities. They also contain annotated information about what users are doing in each moment.

Test dataset

We built a fully labeled dataset that includes recordings from 77 different people. We asked the participants to walk under natural conditions, following some basic premises, while carrying a smartphone. They walked in different environments, at different speeds, accomplishing different goals (walking in corridors, going upstairs, etc.), and varying the position of the device (holding it in the hand, pocket, backpack, etc.). On average, each volunteer walked about 2 minutes. We developed an Android application that samples and logs inertial data (accelerometer and gyroscope) continuously. The smartphone had that app running in background as the user performed the tasks. However, obtaining a reliable ground truth was not straightforward. We were interested on annotating in the recorded signal each step taken by the participant. One option would have been manual labeling. Nevertheless, this is error prone, especially if we consider the amount of data collected. Therefore, we opted for volunteers to carry a set of additional devices in their legs. In particular, we tied another smartphone to each leg using sport armbands, as shown in Figure A.5. These extra devices recorded footstep impacts and communicated the ground truth to the main smartphone.

Figure A.5: Sports armbands holding the smartphones used to obtain the ground truth.

Training dataset

We collected partially labeled data from 10 different people. We developed another Android application that, in addition to recording inertial data, had a graphical interface that the user could interact with to indicate whether he/she was walking or not. This was

optional so, depending on the user's willingness to participate, more or less labeled data were gathered. The app also labeled autonomously some examples applying a series of heuristic rules when dealing with clearly identifiable positives or negatives (e.g., when the phone was idle). Participants installed our application in their own smartphones and recorded data continuously while performing their usual routine.

Preprocessing

Both training and test datasets were subjected to signal preprocessing. We followed the steps we originally proposed in a previous work [171], given the good performance demonstrated in the past. Basically, the raw inertial recordings were filtered, centered, and split into windows of 250 measurements (2.5 seconds). Then, feature extraction was applied, obtaining 21 state-of-the-art features for each window. After this process, we obtained the final sets used in our experiments. Table A.4 summarizes the data distribution in both datasets, attending to the class labels. For training data, a breakdown by user is provided.

Table A.4: Train and test data distribution for the experiments on walking recognition.

		Label			
		Walking	Not walking	Unlabeled	Total
Training set	User 1	3130	2250	4620	10000
	User 2	2519	4359	3122	10000
	User 3	186	325	125	636
	User 4	2432	2455	5113	10000
	User 5	233	2785	6982	10000
	User 6	554	1821	69	2444
	User 7	2582	2052	769	5403
	User 8	232	678	229	1139
	User 9	1151	2669	6180	10000
	User 10	2329	2669	5002	10000
	Total	15348	22063	32211	69622
Test set		6331	1586	0	7917

A.3.3 Full results

In the following, we provide an comprehensive evaluation of ECFL on the walking recognition problem, for different configurations and hyperparameters. In particular, we show the impact of building the local models using 7 different base classifiers, and varying the size of the local and global ensembles (M_l and M_G, respectively), as well as the confidence threshold (γ) and the minimum amount of labeled data required for training (L). We ran each experiment 10 times, randomly varying the iteration in which each client joins the learning. For each setting, we provide the mean and standard deviation of the balanced accuracy.

Table A.5 is an extended version of Table 4.3 (Section 4.5), and shows the influence of the base classifier selection in the results. We provide the average accuracies obtained at the end of the data stream for the global model, but also for each of the local ones. regarding the base classifier used. All other parameters were kept constant, being $M_l = M_G = 5$, $\gamma = 0.9$, and $L = 200$. Table A.5 empirically demonstrates that the global model always performs similar or better than the best of the local ones.

Table A.5: Average accuracies of local and global models in ECFL using different base classifiers.

Base classifier	User 1	User 2	User 3	User 4	User 5	User 6
GLM	0.501 (±0.001)	0.747 (±0.039)	0.700 (±0.005)	0.663 (±0.105)	0.795 (±0.013)	0.661 (±0.035)
Naïve Bayes	0.763 (±0.004)	0.575 (±0.000)	0.787 (±0.001)	0.500 (±0.000)	0.831 (±0.000)	0.823 (±0.004)
C5.0 Tree	0.503 (±0.000)	0.752 (±0.009)	0.639 (±0.039)	0.504 (±0.063)	0.812 (±0.012)	0.566 (±0.008)
SVM	0.490 (±0.001)	0.712 (±0.015)	0.609 (±0.000)	0.741 (±0.008)	0.816 (±0.000)	0.615 (±0.000)
RF	0.503 (±0.000)	0.834 (±0.005)	0.715 (±0.000)	0.819 (±0.028)	0.799 (±0.000)	0.676 (±0.000)
SGB	0.502 (±0.001)	0.692 (±0.031)	0.730 (±0.062)	0.773 (±0.167)	0.817 (±0.000)	0.618 (±0.039)
FNN	0.504 (±0.001)	0.808 (±0.010)	0.706 (±0.013)	0.635 (±0.029)	0.793 (±0.002)	0.652 (±0.007)

Base classifier	User 7	User 8	User 9	User 10	Global
GLM	0.736 (±0.008)	0.737 (±0.047)	0.625 (±0.066)	0.672 (±0.100)	0.743 (±0.066)
Naïve Bayes	0.500 (±0.000)	0.838 (±0.019)	0.761 (±0.008)	0.679 (±0.010)	0.789 (±0.014)
C5.0 Tree	0.533 (±0.111)	0.733 (±0.023)	0.583 (±0.006)	0.547 (±0.084)	0.795 (±0.037)
SVM	0.708 (±0.002)	0.863 (±0.001)	0.684 (±0.149)	0.684 (±0.022)	0.857 (±0.032)
RF	0.682 (±0.006)	0.815 (±0.004)	0.677 (±0.014)	0.769 (±0.010)	0.845 (±0.011)
SGB	0.667 (±0.106)	0.775 (±0.044)	0.697 (±0.075)	0.726 (±0.124)	0.833 (±0.033)
FNN	0.763 (±0.013)	0.703 (±0.094)	0.663 (±0.051)	0.688 (±0.042)	0.803 (±0.027)

It is important to mention that, despite the good performance, the use of Random Forests or SGB as base classifiers in ECFL does not seem to be optimal. This is because these algorithms are already ensembles. For these cases, there are probably much more efficient combination strategies that could be tested. For example, when using Random Forests, it might be better to combine the decisions of all the trees of each forest, at a lower level, instead of just aggregating

the final decisions of each of the forests. Proposals like this are out of the scope of this PhD thesis, but are of interest for future work.

Table A.6 shows the impact of local and global ensemble size on model accuracy. In this case, all executions were performed using SVMs as base classifiers and $\gamma = 3$ and $L = 200$. As we can see, in general the results are better the larger the size of both local and global models. However, it seems that the optimal global ensemble size is 7. Those rows with value 5+ for M_l denote that further increasing the local ensemble size has no impact on the performance. This is because, in this problem, no client detects more than 4 drifts, so no more than 5 base classifiers are ever trained.

Table A.6: Average accuracies of local and global models in ECFL varying the ensemble sizes.

M_G	M_l	User 1	User 2	User 3	User 4	User 5	User 6
1	1	0.488 (±0.037)	0.578 (±0.034)	0.641 (±0.038)	0.688 (±0.126)	0.774 (±0.006)	0.599 (±0.091)
1	3	0.516 (±0.014)	0.665 (±0.050)	0.688 (±0.055)	0.706 (±0.116)	0.774 (±0.006)	0.612 (±0.081)
1	5+	0.516 (±0.014)	0.696 (±0.030)	0.688 (±0.055)	0.706 (±0.116)	0.774 (±0.006)	0.612 (±0.081)
3	1	0.489 (±0.002)	0.565 (±0.006)	0.609 (±0.026)	0.683 (±0.104)	0.816 (±0.000)	0.626 (±0.025)
3	3	0.497 (±0.001)	0.645 (±0.003)	0.631 (±0.000)	0.738 (±0.007)	0.816 (±0.000)	0.624 (±0.020)
3	5+	0.491 (±0.001)	0.711 (±0.015)	0.631 (±0.000)	0.738 (±0.007)	0.816 (±0.000)	0.624 (±0.020)
5	1	0.490 (±0.002)	0.575 (±0.017)	0.609 (±0.000)	0.697 (±0.122)	0.816 (±0.000)	0.593 (±0.053)
5	3	0.497 (±0.000)	0.651 (±0.005)	0.609 (±0.000)	0.741 (±0.008)	0.816 (±0.000)	0.615 (±0.000)
5	5+	0.493 (±0.001)	0.712 (±0.015)	0.609 (±0.000)	0.741 (±0.008)	0.816 (±0.000)	0.615 (±0.000)
7	1	0.489 (±0.001)	0.592 (±0.012)	0.609 (±0.000)	0.738 (±0.093)	0.816 (±0.000)	0.615 (±0.000)
7	3	0.496 (±0.000)	0.655 (±0.009)	0.609 (±0.000)	0.734 (±0.005)	0.816 (±0.000)	0.615 (±0.000)
7	5+	0.490 (±0.001)	0.714 (±0.011)	0.609 (±0.000)	0.734 (±0.005)	0.816 (±0.000)	0.615 (±0.000)
9	1	0.489 (±0.001)	0.564 (±0.011)	0.609 (±0.000)	0.724 (±0.094)	0.816 (±0.000)	0.615 (±0.000)
9	3	0.496 (±0.001)	0.647 (±0.010)	0.609 (±0.000)	0.732 (±0.006)	0.816 (±0.000)	0.615 (±0.000)
9	5+	0.491 (±0.001)	0.709 (±0.013)	0.609 (±0.000)	0.732 (±0.006)	0.816 (±0.000)	0.615 (±0.000)

M_G	M_l	User 7	User 8	User 9	User 10	Global
1	1	0.728 (±0.032)	0.789 (±0.008)	0.680 (±0.085)	0.637 (±0.041)	0.734 (±0.053)
1	3	0.732 (±0.031)	0.865 (±0.010)	0.666 (±0.115)	0.665 (±0.063)	0.769 (±0.069)
1	5+	0.856 (±0.031)	0.865 (±0.010)	0.720 (±0.047)	0.674 (±0.051)	0.769 (±0.069)
3	1	0.710 (±0.030)	0.759 (±0.012)	0.518 (±0.014)	0.691 (±0.008)	0.791 (±0.053)
3	3	0.711 (±0.007)	0.864 (±0.000)	0.525 (±0.006)	0.671 (±0.043)	0.800 (±0.051)
3	5+	0.761 (±0.007)	0.864 (±0.000)	0.589 (±0.079)	0.697 (±0.035)	0.840 (±0.034)
5	1	0.708 (±0.007)	0.760 (±0.012)	0.513 (±0.005)	0.684 (±0.010)	0.845 (±0.047)
5	3	0.752 (±0.000)	0.863 (±0.002)	0.612 (±0.001)	0.673 (±0.134)	0.852 (±0.043)
5	5+	0.752 (±0.002)	0.863 (±0.001)	0.684 (±0.149)	0.689 (±0.022)	0.857 (±0.032)
7	1	0.707 (±0.010)	0.777 (±0.014)	0.528 (±0.026)	0.689 (±0.013)	0.856 (±0.005)
7	3	0.743 (±0.001)	0.863 (±0.002)	0.650 (±0.166)	0.661 (±0.052)	0.855 (±0.018)
7	5+	0.743 (±0.001)	0.863 (±0.002)	0.714 (±0.124)	0.691 (±0.027)	0.868 (±0.018)
9	1	0.704 (±0.006)	0.761 (±0.013)	0.512 (±0.005)	0.689 (±0.009)	0.840 (±0.009)
9	3	0.706 (±0.005)	0.864 (±0.002)	0.576 (±0.059)	0.633 (±0.017)	0.847 (±0.014)
9	5+	0.740 (±0.004)	0.863 (±0.001)	0.664 (±0.022)	0.663 (±0.009)	0.849 (±0.012)

Finally, in Table A.7 we present the influence of the confidence threshold, γ, and the minimum amount of training data, L. Generally, a higher value for L is a guarantee of better

performance. However, the trade-off is that a longer waiting time is required to obtain the minimum amount of data needed for training. In fact, this waiting can be infinite if the clients get very few labeled data. This is precisely what happens to user 5 when $L \geq 400$, which never gets to train a local model. We can also appreciate that, at least for this problem, varying γ does not seem to affect significantly. This may be because the local models selected to be part of the global ensemble are usually those that provide the best results, which tend to belong to clients that already have a significant amount of labeled data.

Table A.7: Average accuracies of local and global models in ECFL varying γ and L.

γ	L	User 1	User 2	User 3	User 4	User 5	User 6
0.85	100	0.495 (±0.001)	0.606 (±0.008)	0.725 (±0.000)	0.700 (±0.006)	0.529 (±0.000)	0.714 (±0.006)
0.85	200	0.490 (±0.001)	0.709 (±0.015)	0.609 (±0.000)	0.730 (±0.007)	0.816 (±0.000)	0.615 (±0.000)
0.85	300	0.486 (±0.000)	0.767 (±0.006)	0.637 (±0.000)	0.823 (±0.032)	0.729 (±0.008)	0.750 (±0.012)
0.85	400	0.486 (±0.000)	0.761 (±0.000)	0.676 (±0.016)	0.778 (±0.012)	–	0.729 (±0.004)
0.85	500	0.501 (±0.001)	0.792 (±0.001)	0.689 (±0.026)	0.735 (±0.014)	–	0.710 (±0.012)
0.90	100	0.495 (±0.001)	0.606 (±0.008)	0.725 (±0.000)	0.698 (±0.003)	0.529 (±0.000)	0.715 (±0.006)
0.90	200	0.490 (±0.001)	0.712 (±0.015)	0.609 (±0.000)	0.741 (±0.008)	0.816 (±0.000)	0.615 (±0.000)
0.90	300	0.486 (±0.000)	0.771 (±0.009)	0.637 (±0.000)	0.809 (±0.013)	0.732 (±0.000)	0.746 (±0.002)
0.90	400	0.486 (±0.000)	0.761 (±0.000)	0.667 (±0.014)	0.771 (±0.007)	–	0.727 (±0.000)
0.90	500	0.501 (±0.001)	0.792 (±0.001)	0.664 (±0.020)	0.731 (±0.006)	–	0.719 (±0.001)
0.95	100	0.495 (±0.001)	0.606 (±0.008)	0.725 (±0.000)	0.698 (±0.003)	0.529 (±0.000)	0.715 (±0.006)
0.95	200	0.491 (±0.001)	0.713 (±0.013)	0.609 (±0.000)	0.738 (±0.006)	0.816 (±0.000)	0.615 (±0.000)
0.95	300	0.486 (±0.000)	0.766 (±0.015)	0.637 (±0.000)	0.812 (±0.012)	0.732 (±0.000)	0.746 (±0.002)
0.95	400	0.486 (±0.000)	0.761 (±0.000)	0.666 (±0.006)	0.766 (±0.000)	–	0.727 (±0.000)
0.95	500	0.501 (±0.001)	0.793 (±0.002)	0.672 (±0.016)	0.722 (±0.020)	–	0.728 (±0.012)

γ	L	User 7	User 8	User 9	User 10	Global
0.85	100	0.609 (±0.015)	0.826 (±0.045)	0.587 (±0.089)	0.577 (±0.003)	0.689 (±0.062)
0.85	200	0.707 (±0.005)	0.863 (±0.001)	0.615 (±0.085)	0.670 (±0.009)	0.861 (±0.013)
0.85	300	0.668 (±0.005)	0.869 (±0.000)	0.808 (±0.001)	0.733 (±0.005)	0.812 (±0.032)
0.85	400	0.777 (±0.013)	0.873 (±0.008)	0.649 (±0.006)	0.800 (±0.010)	0.878 (±0.006)
0.85	500	0.781 (±0.006)	0.874 (±0.002)	0.795 (±0.003)	0.807 (±0.005)	0.873 (±0.003)
0.90	100	0.607 (±0.015)	0.826 (±0.045)	0.581 (±0.094)	0.577 (±0.003)	0.680 (±0.047)
0.90	200	0.708 (±0.002)	0.863 (±0.001)	0.684 (±0.149)	0.684 (±0.022)	0.857 (±0.032)
0.90	300	0.666 (±0.009)	0.867 (±0.003)	0.809 (±0.002)	0.733 (±0.005)	0.825 (±0.019)
0.90	400	0.770 (±0.002)	0.869 (±0.001)	0.650 (±0.006)	0.802 (±0.006)	0.874 (±0.009)
0.90	500	0.776 (±0.004)	0.873 (±0.000)	0.795 (±0.004)	0.806 (±0.005)	0.874 (±0.001)
0.95	100	0.607 (±0.015)	0.826 (±0.045)	0.581 (±0.094)	0.577 (±0.003)	0.680 (±0.047)
0.95	200	0.707 (±0.001)	0.863 (±0.002)	0.592 (±0.084)	0.677 (±0.006)	0.854 (±0.026)
0.95	300	0.668 (±0.002)	0.869 (±0.000)	0.808 (±0.002)	0.733 (±0.006)	0.819 (±0.029)
0.95	400	0.770 (±0.007)	0.871 (±0.003)	0.651 (±0.009)	0.800 (±0.008)	0.878 (±0.004)
0.95	500	0.772 (±0.005)	0.874 (±0.002)	0.796 (±0.004)	0.807 (±0.000)	0.874 (±0.002)

A.4 ECFL: Additional experiments on the HAR multi-class dataset

We further provide additional results in a different task and data. We selected Shoaib's HAR dataset [170], which was already used to evaluate CDA-FedAvg in Sections 3.6 and A.1. Recall

that this one contains records of 10 different people carrying the smartphone while performing seven activities: walking, going upstairs, going downstairs, sitting, standing, jogging, and biking. The data were collected at 50 Hz and includes readings from accelerometer, gyroscope, and magnetometer of the smartphone. The data is fully labeled for all the participants. For each activity, the smartphone was placed in 5 different locations (see Figure 3.1).

We posed the problem as a multiclass classification task where the goal is to correctly predict which of the 7 activities is being performed by the user. We split the raw inertial signals into windows of 124 samples (2.5 seconds). We decided to just keep the accelerometer and gyroscope channels and apply the same preprocessing and feature extraction used in the previous task (Appendix A.3.2). After that, each client has a total of $5,000$ samples, $1,000$ for each phone location. In all the experiments that we will show below, we performed leave-one-out cross-validation at the client level. That is, we used 9 of the clients for training and the remaining one for testing. Thus, each experiment was repeated 10 times, employing $45,000$ samples for training and $5,000$ for testing.

Baseline

Similarly to what we did in Section 4.5, we first provide a baseline to get an idea of the performance that could be achieved under ideal conditions. For that, we joined the data from all clients, shuffled it randomly to have a totally IID dataset, and trained and fine-tuned several classifiers. Then, we also applied the two most common state-of-the-art federated methods: FedAvg and FedProx. Table A.8 shows the average results. In it, we can see that classifiers such as Random Forests and SVM are once again leading the ranking. It can also be noticed that, for this task, the performance of federated methods lags slightly behind that of centralized methods.

Continual federated setting

Next, in order to test ECFL, we configured a CFL setting, generating an evolving data stream for each of the clients. All of them worked at the same frequency, starting at the same time, so the data streams lasted $5,000$ iterations. We sorted the data according to the phone position in the same way for all users: 1st belt, 2nd left pocket, 3rd right pocket, 4th upper arm, and 5th wrist. In this way, we forced a non-stationary distribution that changes a total of 4 times. We ran ECFL using different base classifiers (the same as those evaluated under ideal

Table A.8: Average accuracies of several classifiers, trained and tested in ideal conditions (static, IID).

Method	Accuracy
Naïve Bayes	0.793 (±0.064)
C5.0 Decision Tree	0.834 (±0.032)
Support vector machine (SVM)	0.887 (±0.042)
Random Forests (RF)	0.894 (±0.041)
Stochastic Gradient Boosting (SGB)	0.833 (±0.053)
Feed-forward neural network (FNN)	0.881 (±0.037)
FedAvg	0.836 (±0.038)
FedProx	0.821 (±0.041)

conditions, Table A.8). The hyperparameters were set to their default values: $M_l = M_G = 5$, $\gamma = 0.9$, and $L = 200$. We also executed FedAvg, FedProx, and CDA-FedAvg in this scenario. Table A.9 shows the average accuracies of the global and local models at the end of the data stream. We can see that, although this setting is much more complex, some ECFL configurations provide similar results to those achieved under ideal conditions. For example, ECFL using Random Forests is able to compete with the baseline. CDA-FedAvg also gives good performance, in this case similar to that provided by FedAvg in the baseline. In contrast, FedAvg and FedProx suffer a drop in performance with respect to the previous setting, since they are not intended to work on continual problems.

To conclude this section, in Figure A.6 we present an example of execution of ECFL using Random Forests as base classifier. In this trial, the data from client 5 were reserved for testing, and the other 9 users participated in the training. The upper graph shows the evolution of the accuracies. The thick black line corresponds to the global model, while the rest of the colored lines are each of the clients. The middle graph shows the local updates and global selection. The vertical dashed lines indicate where changes in the position of the smartphone occur. A circumference (∘) on the line means that a drift has been detected and the local model has been updated. If the circumference is filled (•) it indicates that the local model is chosen to be in the global ensemble —the other 4 are marked with a cross (×)—. The bottom graph shows the amount of data stored in each of the clients at any given time.

By looking at Figure A.6, we can see that the first local models are trained around iteration 600. Then, most of the clients perform a total of 4 updates. This is to be expected, given that we have forced 4 drifts to occur. Client 7 only detects 2 of these changes. This can be

Table A.9: Average accuracies on HAR dataset for local and global models in ECFL (using different base classifiers), FedAvg, FedProx, and CDA-FedAvg.

Method	User 1	User 2	User 3	User 4	User 5	User 6
ECFL (NB)	0.648 (±0.049)	0.725 (±0.046)	0.705 (±0.043)	0.632 (±0.066)	0.662 (±0.060)	0.707 (±0.041)
ECFL (C5.0)	0.718 (±0.055)	0.698 (±0.051)	0.728 (±0.076)	0.762 (±0.042)	0.683 (±0.077)	0.641 (±0.046)
ECFL (SVM)	0.758 (±0.076)	0.771 (±0.038)	0.785 (±0.048)	0.792 (±0.043)	0.771 (±0.032)	0.747 (±0.040)
ECFL (RF)	0.827 (±0.076)	0.867 (±0.022)	0.855 (±0.041)	0.878 (±0.052)	0.824 (±0.035)	0.825 (±0.067)
ECFL (SGB)	0.773 (±0.061)	0.775 (±0.051)	0.787 (±0.058)	0.704 (±0.048)	0.621 (±0.081)	0.634 (±0.047)
ECFL (FNN)	0.768 (±0.036)	0.781 (±0.033)	0.762 (±0.047)	0.749 (±0.052)	0.765 (±0.63)	0.761 (±0.045)
FedAvg	0.765 (±0.028)	0.741 (±0.062)	0.754 (±0.049)	0.744 (±0.045)	0.745 (±0.043)	0.723 (±0.059)
FedProx	0.755 (±0.011)	0.737 (±0.011)	0.753 (±0.012)	0.746 (±0.011)	0.745 (±0.010)	0.713 (±0.010)
CDA-FedAvg	0.784 (±0.112)	0.797 (±0.090)	0.742 (±0.121)	0.693 (±0.123)	0.802 (±0.069)	0.793 (±0.107)

Method	User 7	User 8	User 9	User 10	Global
ECFL (NB)	0.726 (±0.040)	0.691 (±0.033)	0.654 (±0.053)	0.720 (±0.059)	0.736 (±0.039)
ECFL (C5.0)	0.649 (±0.073)	0.605 (±0.047)	0.709 (±0.046)	0.698 (±0.049)	0.750 (±0.036)
ECFL (SVM)	0.754 (±0.062)	0.758 (±0.056)	0.738 (±0.057)	0.735 (±0.053)	0.815 (±0.022)
ECFL (RF)	0.796 (±0.043)	0.800 (±0.091)	0.837 (±0.040)	0.863 (±0.032)	**0.884** (±0.053)
ECFL (SGB)	0.703 (±0.056)	0.736 (±0.045)	0.745 (±0.043)	0.696 (±0.066)	0.793 (±0.041)
ECFL (FNN)	0.736 (±0.035)	0.726 (±0.038)	0.713 (±0.057)	0.717 (±0.054)	0.790 (±0.021)
FedAvg	0.771 (±0.059)	0.742 (±0.057)	0.744 (±0.045)	0.747 (±0.060)	0.788 (±0.053)
FedProx	0.762 (±0.011)	0.738 (±0.012)	0.731 (±0.011)	0.751 (±0.013)	0.790 (±0.009)
CDA-FedAvg	0.813 (±0.087)	0.793 (±0.086)	0.779 (±0.123)	0.799 (±0.090)	0.839 (±0.036)

explained mainly because this is the participant with the worst local model. In the case of client 4, which only detects 3 drifts, the explanation could be just the opposite: it has the best local model, so it is able to generalize enough that the last change does not have an impact on the confidence of the model. At the end of the process, the global ensemble is composed of the local models of users 1, 3, 6, 8 and 10. We can see that the global model provides always similar or greater performance than that of any of the local ones.

A.5 Implementation details

Here we provide additional details on the execution of the experiments conducted in Chapters 3 and 4, as well as those described in the previous sections of the present appendix.

A.5.1 Hardware

All experiments were carried out on a desktop computer running Ubuntu 18.04 and equipped with Intel® Core™ i7-4790 processor, Intel® Haswell Desktop graphics, and 27.3 GiB of DDR3 RAM. In addition, data collection for the task of walking recognition was done using

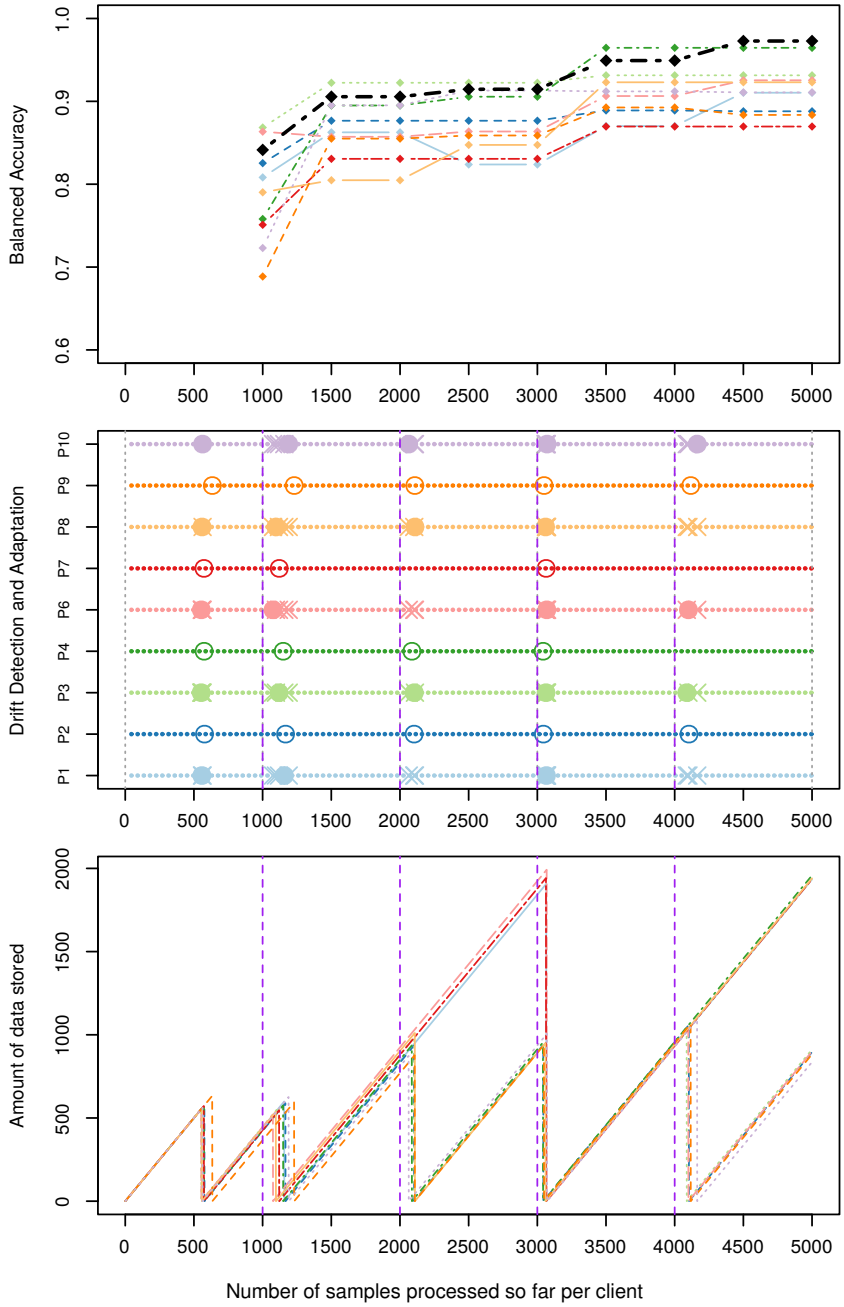


Figure A.6: Example of executing ECFL in the HAR task. The black line corresponds to the global model, while the rest of the coloured lines are associated with one client.

13 mid-range smartphones from different manufacturers and running Android OS in different versions (between 7.0 and 9.0).

A.5.2 Software

The apps used to record the data for walking recognition were implemented in Android SDK API 25. Data preprocessing, model training, and evaluation, were performed using different frameworks and programming languages:

- **Experiments with FedAvg, FedProx, and CDA-FedAvg**. We used Python 3.6 together with TensorFlow 2.5 library to train and validate all the models.
- **Experiments with ECFL and traditional classifiers**. In this case, we opted the R language. In particular, for the training of the baseline models (Naïve Bayes, GLM, C5.0, etc.) we used the algorithm implementations already provided by the `caret` package [221]. The ECFL framework was also developed in R using `caret` to train the base classifiers.

A.5.3 Model architectures and hyperparameters

Depending on the task and method to be evaluated, different model architectures and hyperparameters were used.

Experiments with CDA-FedAvg

As with most federated methods, FedAvg and CDA-FedAvg work well with neural networks. In the experiments performed with these two methods in Chapter 3 and in this appendix, we employed convolutional architectures. This decision is justified based on the good performance of this type of network in vision and time series processing problems [36, 171, 172]. We used a particular configuration depending on the task:

- **Human activity recognition using Shoaib's dataset**. We worked with six-dimensional signal windows consisting of 124 measurements. Each of the six dimensions was associated to one of the axes of the acceleration and angular velocity. Hence, the proposed architecture had 6 input channels. It consisted of two 1D convolutional layers, one max-pooling layer, one flattening layer, two dropout layers, and two fully connected layers. The total number of learnable parameters was $764,399$. Table A.10 shows the details of the architecture. We used a dropout rate of 0.2 in both dropout layers and applied the *softmax* activation function to the output. For training, we employed a batch size of 100 windows.

- **Digit recognition with the Digit-five dataset**. In this other problem, we worked with RGB images of 32×32 pixels. The proposed architecture had 3 input channels and consisted of four 2D convolutional layers, four max-pooling layers, one flattening layer, one dropout layers, and three fully connected layers. The total number of trainable parameters was $33,050$. Table A.11 shows the details of the architecture. We used a dropout rate of 0.2 and applied the *Rectified Linear Unit (ReLu)* activation function to the outputs of the convolutional layers. We also applied *softmax* activation to the output of the model. For training, we employed a batch size of 50 images.

Table A.10: Details of the CNN architecture used for the experiments on HAR with FedAvg and CDA-FedAvg.

Layer name	Kernel size	# kernels	Stride	Feature map.	# params
input	–	–	–	124×6	0
conv1	10	100	1	115×100	$6,100$
conv2	10	100	1	106×100	$100,100$
max_pool	2	–	2	53×100	0
dropout1	–	–	–	53×100	0
flattening	–	–	–	$1 \times 5,300$	0
fully_con1	–	–	–	1×124	$657,324$
dropout2	–	–	–	1×124	0
fully_con2	–	–	–	1×7	875

Experiments with ECFL

To train the baseline models (Naïve Bayes, GLM, C5.0, etc.) from Tables 4.1 and A.8, a grid search for the optimal hyperparameters was performed using the methods already provided by the `caret` package. In the case of the base classifiers for ECFL, no hyperparameter tuning was applied, maintaining the default values proposed by `caret`. All SVMs used a radial basis function (RBF) kernel. All feed-forward neural networks had 3 hidden layers, with 32 neurons in each. When FedAvg and FedProx were tested under ideal conditions (Tables 4.2 and A.8) they were allowed to train for 30 rounds, performing 3 local epochs per round and using all local models for global aggregation.

Table A.11: Details of the CNN architecture used for the experiments on digit recognition with FedAvg and CDA-FedAvg.

Layer name	Kernel size	# kernels	Stride	Feature map.	# params
input	–	–	–	$32 \times 32 \times 3$	0
conv1	3×3	16	1×1	$32 \times 32 \times 16$	448
max_pool1	2×2	–	2×2	$16 \times 16 \times 16$	0
conv2	3×3	32	1×1	$16 \times 16 \times 32$	4,640
max_pool2	2×2	–	2×2	$8 \times 8 \times 32$	0
conv3	3×3	32	1×1	$8 \times 8 \times 32$	9,248
max_pool3	2×2	–	2×2	$4 \times 4 \times 32$	0
conv4	3×3	32	1×1	$4 \times 4 \times 32$	9,248
max_pool4	2×2	–	2×2	$2 \times 2 \times 32$	0
flattening	–	–	–	1×128	0
fully_con1	–	–	–	1×64	8,256
dropout2	–	–	–	1×64	0
fully_con2	–	–	–	1×16	1,040
fully_con3	–	–	–	1×10	170

A.5.4 Data and other supplementary resources

The walking recognition dataset used for the experiments with ECFL (Sections 4.5 and A.3 was released, so that the scientific community can use it as benchmark in future research. It can be found on the CiTIUS website: https://citius.usc.es/t/30. We provide both raw and processed data. In addition, the Android app that the volunteers installed on their smartphones for data collection is available in the Google Play store: https://play.google.com/store/apps/details?id=es.usc.citius.inertialnav.

The other datasets used in the experimentation with CDA-FedAvg and ECFL are all public:

- Shoaib's HAR dataset [170] can be downloaded from the University of Twente website: https://www.utwente.nl/en/eemcs/ps/research/dataset/.
- The original MNIST [209] is available on Yann LeCun's personal website: http://yann.lecun.com/exdb/mnist/.
- MNIST-M [210] can be found in GitHub: https://github.com/zumpchke/keras_mnistm.

- SVHN [211] is posted in the Stanford University repository: http://ufldl.stanford.edu/housenumbers/.
- Synthetic Digits [212] is available at Kaggle website: https://www.kaggle.com/datasets/prasunroy/synthetic-digits.
- Finally, USPS [213] is also posted at Kaggle: https://paperswithcode.com/dataset/usps.

All URLs provided in this section were last accessed on August 2, 2022.

Appendix B

Publications

This appendix contains a list of those publications resulting from the research carried out over the last 4 years, from which some of the contents of this PhD dissertation have been extracted.

Journals

Publication J–01

> [98] F. E. Casado, D. Lema, M. F. Criado, R. Iglesias, C. V. Regueiro, and S. Barro. "Concept drift detection and adaptation for federated and continual learning," *Multimedia Tools and Applications*, Springer, vol. 81, no. 3, pp. 3397-3419, 2021. DOI: 10.1007/s11042-021-11219-x. ISSN: 1380–7501.

Quality indicators

- Journal Citation Reports (JCR) (2021):
  - Impact factor: 2.577.
  - Quartile: Q2 in categories *Computer Science, Theory & Methods* (42/109) and *Computer Science, Software Engineering* (48/110).
- Scopus Scientific Journal Rankings (SJR) (2021):
  - Cite score: 0.716.

– Quartile: Q1 in category *Media Technology* (12/55).

PhD candidate contribution

Conceptualization, formal analysis, investigation, methodology, data curation, visualization, validation, and writing.

Reproduction rights

This publication was partially reproduced in Chapter 3 and Appendix A under CC BY license, as shown in Figure B.1.

Figure B.1: CC BY license for publication J-01 (98).

Publication J-02

[173] F. E. Casado, D. Lema, R. Iglesias, C. V. Regueiro, and S. Barro. "Ensemble and continual federated learning for classification tasks," *Machine Learning*, Springer. ISSN: 0885-6125.
This paper is currently under review (second round).

Quality indicators

- JCR (2021):
  - Impact factor: 5.414.
  - Quartile: Q2 in category *Computer Science, Artificial Intelligence* (40/144).
- SJR (2021):
  - Cite score: 1.64.
  - Quartile: Q1 in categories *Artificial Intelligence* (41/239) and *Software* (41/358).

PhD candidate contribution

Conceptualization, formal analysis, investigation, methodology, software, data curation, visualization, validation, and writing.

Reproduction rights

This article was partially reproduced in Chapter 4 and Appendix A, and it is currently under review.

Publication J–03

> [15] M. F. Criado, F. E. Casado, R. Iglesias, C. V. Regueiro, and S. Barro. "Non-IID data and Continual Learning processes in Federated Learning: A long road ahead," *Information Fusion*, Elsevier, vol. 88, pp. 263–280, 2022. DOI: 10.1016/j.inffus.2022.07.024. ISSN: 1566-2535.

Quality indicators

- JCR (2021):
  - Impact factor: 17.564.
  - Quartile: Q1 in categories *Computer Science, Theory & Methods* (1/110) and *Computer Science, Artificial Intelligence* (4/144).
- SJR (2021):
  - Cite score: 4.56.

– Quartile: Q1 in categories *Information Systems* (4/335), *Signal Processing* (2/95), *Software* (7/358), and *Hardware and Architecture* (2/149).

PhD candidate contribution

Conceptualization, formal analysis, investigation, methodology, software, data curation, and visualization.

Reproduction rights

This publication was partially reproduced in Chapter 6 and Appendix A under CC BY license, as shown in Figure B.2.

Figure B.2: CC BY license for Publication J–03 (98).

Publication J–04

[171] F. E. Casado, G. Rodríguez, R. Iglesias, C. V. Regueiro, S. Barro, and A. Canedo-Rodríguez. "Walking recognition in Mobile Devices," *Sensors*, MDPI, vol. 20, no. 4, 1189, 2020. DOI: 10.3390/s20041189. ISSN: 1424-8220.

Quality indicators

- JCR (2020):
  - Impact factor: 3.576.
  - Quartile: Q1 in category *Instruments & Instrumentation* (14/64).
- SJR (2020):
  - Cite score: 0.716.
  - Quartile: Q2 in categories *Information Systems* (84/337), *Electrical and Electronic Engineering* (166/638), and *Instrumentation* (34/131).

PhD candidate contribution

Conceptualization, formal analysis, investigation, methodology, software, data curation, visualization, validation, and writing.

Reproduction rights

This publication was partially reproduced in Chapter 3 and Appendix A under CC BY license, as shown in Figure B.3.


Open Access Article

Walking Recognition in Mobile Devices

by Fernando E. Casado [1,*], Germán Rodríguez [2], Roberto Iglesias [1,*], Carlos V. Regueiro [3], Senén Barro [1] and Adrián Canedo-Rodríguez [2]

[1] CiTIUS (Centro Singular de Investigación en Tecnoloxías Intelixentes), Universidade de Santiago de Compostela, 15782 Santiago de Compostela, Spain
[2] Situm Technologies S.L., 15782 Santiago de Compostela, Spain
[3] CITIC, Computer Architecture Group, Universidade da Coruña, 15071 A Coruña, Spain
[*] Authors to whom correspondence should be addressed.

Sensors **2020**, *20*(4), 1189; **https://doi.org/10.3390/s20041189**

Received: 21 January 2020 / Revised: 14 February 2020 / Accepted: 18 February 2020 / Published: 21 February 2020

(This article belongs to the Special Issue **Human and Animal Motion Tracking Using Inertial Sensors**)

Figure B.3: CC BY license for publication J-04 (171).

Conferences

Publication C-01

> [193] F. E. Casado, Y. Demiris. "Federated Learning from Demonstration for Active Assistance to Smart Wheelchair Users," in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2022)*, IEEE, 2022. This paper has been accepted and is pending for publication in the conference proceedings. It will be presented in October 2022 in Kyoto, Japan.

Quality indicators

- GII-GRIN-SCIE (GGS) Conference Rating (2021):
  - Class: 1.
  - Rating: A+.
- Computing Research & Education (CORE) Ranking (2021):
  - Rank: A.
- Microsoft Academic Conference Ranks (2021):
  - Class: A++.

PhD candidate contribution

Conceptualization, formal analysis, investigation, methodology, data curation, visualization, validation, and writing.

Reproduction rights

This paper was partially reproduced in Chapter 5 and is pending for publication in the conference proceedings.

Federated learning (FL) is a machine learning paradigm that allows training models in a distributed way, among multiple devices, without compromising user privacy. In this PhD thesis, we propose new FL strategies that, while maintaining all the advantages that this technology already provides, also allow us to handle continual scenarios with non-stationary data and concept drift. We formulate two complementary strategies: CDA-FedAvg, which enables the training of a global deep neural network, and ECFL, which poses the model as an ensemble of local learners. We evaluated our solutions in different use cases, including activity recognition in smartphones and assistance to robotic wheelchair users. The results highlight the relevance of continual FL and, in particular, the advantages and impact of our contributions.