Ensemble and continual federated learning for classification tasks

Fernando E. Casado[1] · Dylan Lema[1] · Roberto Iglesias[1] · Carlos V. Regueiro[2] · Senén Barro[1]

Abstract
Federated learning is the state-of-the-art paradigm for training a learning model collaboratively across multiple distributed devices while ensuring data privacy. Under this framework, different algorithms have been developed in recent years and have been successfully applied to real use cases. The vast majority of work in federated learning assumes static datasets and relies on the use of deep neural networks. However, in real-world problems, it is common to have a continual data stream, which may be non-stationary, leading to phenomena such as concept drift. Besides, there are many multi-device applications where other, non-deep strategies are more suitable, due to their simplicity, explainability, or generalizability, among other reasons. In this paper we present Ensemble and Continual Federated Learning, a federated architecture based on ensemble techniques for solving continual classification tasks. We propose the global federated model to be an ensemble, consisting of several independent learners, which are locally trained. Thus, we enable a flexible aggregation of heterogeneous client models, which may differ in size, structure, or even algorithmic family. This ensemble-based approach, together with drift detection and adaptation mechanisms, also allows for continual adaptation in situations where data distribution changes over time. In order to test our proposal and illustrate how it works, we have evaluated it in different tasks related to human activity recognition using smartphones.

Keywords Federated learning · Ensemble learning · Concept drift · Semi-supervised classification · Smartphones

1 Introduction

Smartphones, wearables, robots, or "things" from the Internet of Things (IoT) are already counted in millions. These devices benefit from increasingly better sensorization, as well as intercommunication capabilities (Li et al., 2018a). In this context, *multi-device*

✉ Fernando E. Casado
  fernando.estevez.casado@usc.es

Extended author information available on the last page of the article

machine learning is an opportunity to develop new and sophisticated applications in all human domains: education, health, banking, sport, etc. The idea is that, although each individual device may have limitations, these can be overcome by collaborating as a society. One important challenge is non-stationary data. Each device will likely collect data on a continual basis, leading to a potentially unbounded data stream (Ramírez-Gallego et al., 2017). Thus, it may be impossible to process and store all incoming information. Moreover, access to true class labels can be limited or delayed, and the data distribution may change in unforeseen ways, causing the phenomenon known as *concept drift* (Widmer & Kubat, 1996).

The most immediate way to perform multi-device learning would be a centralized approach, i.e., uploading and processing all the data jointly on a cloud server using any machine learning (ML) technique. Nevertheless, this can lead to scalability issues (Ananthanarayanan et al., 2017; Bagui & Nguyen, 2015), and may put data privacy into risk (Custers et al., 2019; Gaff et al., 2014). In contrast, distributed approaches are much more appealing. In this sense, the main paradigm is *federated learning* (FL) (McMahan et al., 2016). It consists of alternating local learning in the devices with global consensus in the cloud. Under this approach, no sensitive data is shared, just the learning model. FL has been successfully applied in several real-world tasks, such as mobile keyboard prediction (Hard et al., 2018), malware detection (Rey et al., 2022), energy demand prediction (Saputra et al., 2019), medical diagnosis (Brisimi et al., 2018), or autonomous navigation in robotics (Liu et al., 2019).

The vast majority of work in FL has relied on the parallel training of a deep neural network (DNN). That is, all client devices locally update the same model using their private data. However, although deep learning provides good results in a large number of applications, it also has shortcomings that sometimes limit its scope, e.g. computational complexity (Schwartz et al., 2020), model opacity (Arrieta et al., 2020; Fazi, 2021), or tendency to overfitting (O'Mahony et al., 2019). On top of this, it is well known that DNNs are very prone to suffer from what is known as *catastrophic forgetting* (Lesort et al., 2020), i.e., the inability to retain old information in the presence of new one. This issue is critical when data is processed continuously.

In view of the limitations of standard FL, new solutions that do not depend on deep learning have started to be explored (Bakopoulou et al., 2019; Ludwig et al., 2020). Typically, these are adaptations of classical ML algorithms to the federated setting. Nevertheless, the common global aggregation mechanism of FL is not always possible using these methods, forcing to look for specific alternatives. This leads to a lack of homogeneity in solution designs, and makes it difficult to extrapolate them to new settings. Besides, almost no progress has been made in providing these FL algorithms with the capacity for continual adaptation.

In this work, we present *Ensemble and Continual Federated Learning* (ECFL), a general and flexible architecture intended to solve multi-device classification tasks using any ML method in a federated and continual fashion. Our approach strongly relies on ensemble techniques. *Ensemble learning* (Sagi & Rokach, 2018) consists of training multiple *base learners* and combining their outputs to obtain 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. Besides, ECFL includes concept drift handling mechanisms, allowing the devices to detect changes and update their local models accordingly. Keeping the global model up to date is as straightforward as replacing learners in the ensemble. This makes it possible to tackle *continual single-task problems*, i.e., the task remains the same but the

underlying distribution of data might change over time. Finally, ECFL is also prepared to work in semi-supervised scenarios, by using the global model for labeling local data. This is very useful because, in continual learning, true class labels may be scarce or arrive with delay. 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 (Androutsopoulos et al., 2000).

Our approach has the following strengths:

- It allows to work with local learners different from DNNs. This has several advantages in terms of *simplicity*, as it opens the door to employ methods that have fewer parameters, require smaller amounts of data, and involve lighter computations. Thus, it is more environmentally friendly, in line with the *green ML* initiative (Schwartz et al., 2020; Strubell et al., 2019). Furthermore, there are other benefits regarding *explainability*, since it is possible to use interpretable learners such as decision trees (Arrieta et al., 2020). This is very useful in applications such as medical diagnosis or autonomous driving, where it is critical to know why each decision is made.
- 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 brings *flexibility*, and *autonomy*. It 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.
- Local learning also integrates concept drift detection and semi-supervised labeling, thus promoting continual *adaptability*. 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. Our approach can deal with continual single-task problems without experiencing phenomena such as catastrophic forgetting.
- The choice of ensemble learning for global aggregation also enhances *generalization*. It has been shown the ability of ensembles to reduce bias and variance (Dietterich et al., 2002; Zhou, 2009). 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 rest of the paper is structured as follows: Sect. 2 provides a review of the state of the art. Section 3 introduces the challenges of continual learning in federated settings. In Sect. 4, ECFL is presented. Section 5 explains in detail the local learning, carried out on the devices. Section 6 exposes global consensus stage, performed in the cloud. The experimental results are shown in Sect. 7. In Sect. 8, some thoughts about privacy are given. Finally, our conclusions are exposed in Sect. 9.

2 Related work

Federated learning (McMahan et al., 2016; Li et al., 2019; Lim et al., 2019) is a distributed machine learning framework that allows the training of a model across multiple decentralized client devices. Its core idea is simple: solving local problems on the clients and combining their knowledge in the cloud without exchanging any raw data. In a standard federated scenario, the model is initialized in a central server and shared with the participants.

Then, several learning rounds are conducted, alternating between *local training* and *global aggregation*. In each round, the clients receive the latest version of the model from the server, perform the local training using their local datasets, and send back their updates. After that, the individual contributions are merged on the server, thus upgrading the model.

The federated learning literature is closely related to deep learning. Most of the methods that have been proposed in recent years are designed to collaboratively train a DNN. The most popular one is FedAvg (McMahan et al., 2016). At each FedAvg round, each client performs the local learning stage by conducting stochastic gradient descent (SGD) on its private dataset to adjust the learnable parameters (weights and biases) of the network. On the server side, the aggregation of all local parameter updates is carried out by means of a weighted arithmetic mean, which relies on the quantity of data employed in each update. That is, the more data a participant uses for local training, the more influence it has on the global model. FedProx (Li et al., 2018b) is another popular method that can be viewed as a generalization of FedAvg. By adding a proximal term during optimization, it helps to improve stability in model convergence when the local datasets are heterogeneous.

Although there is a large amount of work on FL with DNNs, there are very few proposals based on other, non-deep algorithms. In the same way, little research has been done on continual adaptation of federated learning in non-stationary data streams. In the following, we give an overview of the studies in the literature that focus specifically on these aspects.

2.1 Ensemble federated learning

There are some federated methods that are adaptations of classical ML algorithms. Soliman et al. (2020) introduce a federated k-means to address decentralized clustering by integrating HyperLogLog counters with a P2P architecture. Bakopoulou et al. (2019) propose a federated SVM with linear kernels to solve the problem of mobile packet classification. Ludwig et al. (2020) implement a federated ID3 decision tree that grows at the server side while the clients only perform light computations. All the above methods are characterized by having been designed to solve a specific problem, so their use in new applications is limited. This is mainly because the ML methods on which they are based do not naturally lend themselves to federation, since most of them are not based on gradient descent and, therefore, the common global aggregation approaches (such as the weighted average) are not feasible.

The idea of *ensemble federated learning* (EFL) has been recently studied in the literature and promises to be a more generalizable approach. Hamer et al. (2020) propose Fed-Boost, an algorithm for learning an ensemble of pre-trained base predictors via federated learning. In this work, the authors use a weighted ensemble, where the output of each predictor has a certain weight. The goal is to learn the set of weights in a federated fashion, so that the performance of the ensemble is maximized. Although the method is interesting, its application is very limited, as it assumes that the server is provided with a set of pre-trained models.

Lin et al. (2020) develop an algorithm for ensemble distillation in FL settings. They suggest that clients train local models that are then merged on the server by distillation. 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. (2019) present *one-shot* federated learning. Their aim is to learn a global model over a network of devices in a single round of communication. For that, they propose an ensemble of locally trained SVMs. They suggest several strategies to select a subset with the best local models to add to the global ensemble and thus limit its size. Although it is a preliminary work, the performed experiments suggest that ensemble-based federated methods have great potential. Nevertheless, we believe that the efficiency in communications achieved with one-shot learning does not compensate for the sacrifice in performance.

2.2 Continual federated learning

Continual learning (CL) (Lesort et al., 2020; Parisi et al., 2019) is the machine learning paradigm that seeks to train models in environments that evolve over time. The goal is to smoothly update the learner to take into account new data distributions or tasks while retaining prior knowledge. There is a great deal of work in CL posed in a centralized way, involving a single learner. However, there is not much research on CL in federated settings.

Yoon et al. (2021) propose FedWeIT, a federated framework for inter-client knowledge transfer in continual learning. They pose a scenario where each client learns on a sequence of tasks. FedWeIT decomposes the model weights into global and task-specific parameters, allowing each participant to adaptively train benefiting from the knowledge of other clients that share common tasks. Although the idea is interesting, the work does not focus on continual learning challenges. The training is addressed in a classical manner, prefixing a number of communication rounds and assuming that all tasks are known in advance. In real-world problems, however, tasks often arise and evolve unpredictably, leading to concept drift, so it is important to have mechanisms for detecting and adapting to change.

Park et al. (2021) present FIL-VER, an algorithm designed to address incremental single-task FL problems. FIL-VER is based on applying rehearsal to avoid catastrophic forgetting. For that, the authors use variational embedding encoders. The algorithm is able to deal with different scenarios 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 is a limitation for its implementation in real problems.

To the best of our knowledge, there are no proposals in the literature for federated and continual learning with explicit drift detection, nor based on methods other than DNNs. The present work is in line with the aforementioned studies in EFL (Guha et al., 2019; Lin et al., 2020), relying on an ensemble of local learners as a means of global aggregation, but also taking advantage of ensemble techniques to enable continual adaptation.

3 Learning under data streams and concept drift in federated settings

A *data stream* is a potentially unbounded sequence of data arriving over time (Ramírez-Gallego et al., 2017). Learning under data streams imposes several constraints that cannot be fulfilled by standard ML algorithms, thus appearing new solutions that fall within the field of CL. We highlight the following key aspects:

- Data is not given beforehand, but become available over time.
- The size of the stream may be infinite, so it can be infeasible to store all data in memory.
- Each instance may be accessed a limited number of times to guarantee storage space.
- The amount of labeled samples could be small because of the high cost of querying the label for each incoming instance.
- The access to the labels might be delayed by hours, days, or even months, depending on the problem to be treated.
- Statistical properties of data may vary over time unpredictably, leading to concept drift.

Special attention should be paid to concept drift, since it can cause that the inducted knowledge of past data may not be relevant anymore, leading to poor predictions or decision outcomes. Formally, we can define concept drift as follows: Given a time period [0, t], a set of samples, denoted as $S_{0,t} = \{s_0, \dots, s_t\}$, where $s_i = (x_i, y_i)$ is one data instance, x_i is the feature vector, y_i is the label, and $S_{0,t}$ has a certain joint probability distribution of x and y, $P_t(x, y)$. Therefore, concept drift can be defined as a change in the joint probability at timestamp t, such that $\exists t : P_t(x, y) \neq P_{t+1}(x, y)$. Given that the joint probability can be factorized as $P(x, y) = P(x) \cdot P(y \mid x)$, we can categorize concept drift into two types (Webb et al., 2016): (1) virtual, and (2) real. *Virtual concept drift* implies shifts in the input probability, $P(x)$, whereas *real concept drift* is caused by novelty on data, which has an effect on posterior class probabilities, $P(y \mid x)$.

The above applies to a single data stream. Nevertheless, when we talk about multi-device learning, each of the participants will have an independent data stream, with different properties, being able to drop in or out at any time. Thus, the problem becomes more complex. Now, the goal is to train a global model in a distributed and parallel way using the local data of the D available clients. Each client will have a different bias because of the conditions of its local environment, and likewise, its data stream may change in different ways over time. Therefore, there may occur concept drifts that affect all clients, some of them, or just a single one. Thus, we can generalize the problem in the following way: Given a time period [0, t], a set of clients $\{d_1, \dots, d_D\}$, and a set of local samples for each client, denoted as $S_{0,t}^k = \{s_0^k, \dots, s_t^k\}$, where $s_i^k = (x_i^k, y_i^k)$ is one data instance from client d_k, x_i^k is the feature vector, and y_i^k is the label. Each local dataset $S_{0,t}^k$ has a certain joint probability $P_t^k(x, y)$. A *local concept drift* occurs at timestamp t for client d_k if $\exists t, k : P_t^k(x, y) \neq P_{t+1}^k(x, y)$. In the same way, we can define a *global concept drift* as a probability change at time t such that $\exists t : P_t^G(x, y) \neq P_{t+1}^G(x, y)$, where $P_t^G(x, y)$ is the global joint probability of all clients $\{d_1, \dots, d_C\}$. Both local and global drift can be either virtual or real. Note that a global drift does not necessarily mean that a local drift will occur for all the clients at exactly the same time t.

If there is a real and local drift in only one or a few clients, they will start to behave very differently from the others and, hence, a global model will not provide a good performance for them. To deal with this kind of problems, there are already several personalization techniques in the literature (Sattler et al., 2019; Wang et al., 2019a). However, in this work we are not interested in that situation, but we want to deal with scenarios where changes affect all participants. Thus, we focus on global concept drift. In our experiments we assume that change occurs locally in all clients closely in time. Besides, we consider changes in the

input data, $P(x)$, i.e., virtual drifts. This is because we pose drift handling in a totally unsupervised manner, which is not possible when working with real drift.

Algorithms that deal with data streams and concept drift can be categorized according to different criteria (Lu et al., 2018; Ditzler & Polikar, 2012):

- 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.
- 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.
- Finally, 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 to collect a representative amount of data 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. In this work, we have designed a federated ensemble-based architecture that learns in batches and provides active drift detection in all the devices. The use of ensembles has already been justified in the previous sections by the fact that they are easily adaptable to federated environments, providing an alternative and general way to perform global aggregation different from weighted average. The preference for batch processing and active drift detection is motivated by the efficiency of client–server communications. Performing passive and online learning on the devices would imply training and sending updates to the server continuously and at a high frequency. This would mean a high workload for the clients and could also lead to bottlenecks on the server side. Instead, there are significant benefits in applying active drift detection together with batch learning. Explicit drift detection answers two key questions: *what* to learn and *when* to learn it. In a FL setting, this information is even more relevant, since it determines not only when a model should be locally updated, but also uploaded to the cloud. Hence, drift detection together with batch learning may be lighter in terms of number of communications and computational cost. In particular, when the data distribution changes little, the number of operations will be very low, since training is conducted only when necessary.

4 Ensemble and continual federated learning: an overview

We present Ensemble and Continual Federated Learning (ECFL), a new FL architecture based on ensemble learning and suitable for dealing with continual data streams and concept drift. Figure 1 shows a high-level diagram of our proposal. As we can see, it involves a cyclical learning. In the figure there are smartphones, but we could think of any other set of devices, either homogeneous or heterogeneous, including wearables, robots, etc. Each client device is able to perceive its environment through its sensors and is connected to the cloud.
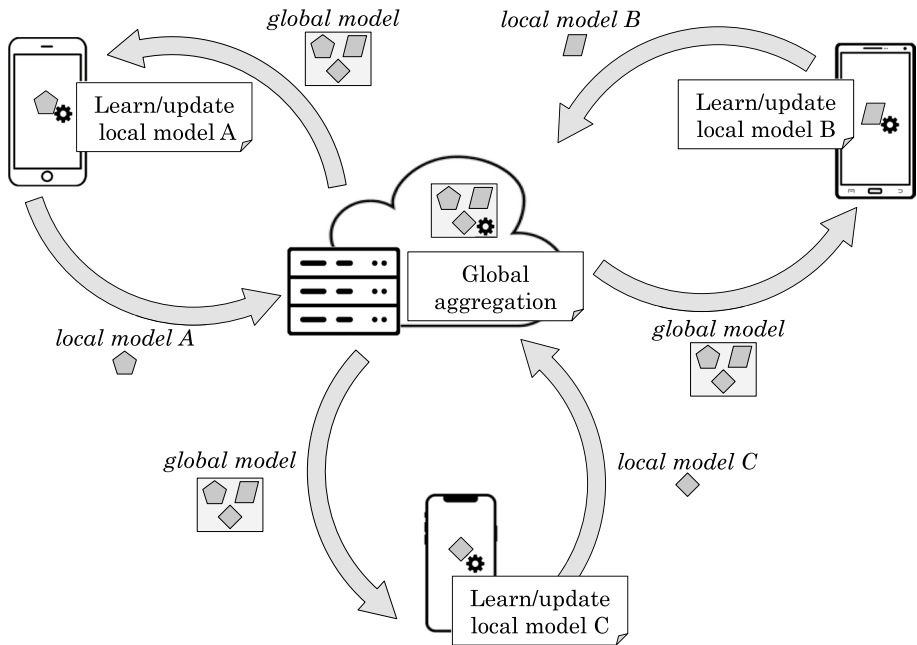
Fig. 1 High-level diagram of ECFL

Iteratively, each of the devices creates and updates its own local model of the problem. For that, each of them is continuously acquiring new information from the data stream perceived through its sensors. This information is raw data, which must be locally pre-processed before using it in a learning stage: noise detection, data transformation, feature extraction, data normalization, instance selection, etc. The preprocessing is task-dependent and it is performed online as new information becomes available. The preprocessed data are stored locally until a significant amount is available to train a model. Each local model is sent to the cloud where a global aggregation stage is performed to join the local contributions, thus obtaining a global model. The global model is then shared with all the devices in the network. After that, each device can take advantage of that global model 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.

Note that the information available at the local level will increase progressively and may evolve in unpredictable ways. As it is unrealistic to assume that infinite storage is available, a compromise solution must be reached to retain previous knowledge that is still relevant at the same time as old information is replaced with new one. ECFL allows to learn under strong storage constraints, working only with a small amount of data within a sliding window. Our proposal is also robust in the lack of labeled instances by incorporating a semi-supervised labeling system. Finally, it is able to deal with concept drift, as it integrates local mechanisms for change detection and adaptation. In this work, we have focused on continual single-task scenarios.

Thus, we address classification problems where the input data space, X, and the set of possible classes, $\{c_1, \dots, c_C\}$, are the same for all clients, and do not change over time, although the input probabilities $P(x)$, $x \in X$, may vary, leading to virtual concept drift.

Ensemble techniques play an important role in our proposal. First, the global model is an ensemble composed of a selection of local models. This allows global aggregation to be performed regardless of the learning algorithm used locally, which does not have to be SGD-based. The server is in charge of choosing the most relevant local models to become part of the global ensemble through a distributed voting system. On the other hand, we also use ensemble techniques locally to allow for continual adaptation to 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. In the following sections we explain the details of our approach at both learning levels, local (Sect. 5), and global (Sect. 6).

5 Local learning

Figure 2 shows the continuous work flow for each device. Basically, devices gather raw data from the environment. These data, conveniently preprocessed, are used to build or update the local model. The preprocessing of the data refers to feature extraction, normalization, instance selection, etc. Note that it might be partially labeled.

As we mentioned in the previous section, in order to learn and update the local model, we have opted for the use of ensembles. In particular, every device builds its own local ensemble of base classifiers.[1] Any algorithm that provides posterior probabilities for its predictions can be used as base classifier. In our experimental results (Sect. 7), we tried different methods: Naïve Bayes, C5.0 Decision Trees, Support Vector Machines, 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 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 (Czyz et al., 2004; Kittler et al., 1998). 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 (Tumer & Ghosh, 1996). 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. Thus, we predict that an instance x belongs to class c_j if the following condition is fulfilled:

$$\text{median}\{y_{1_j}(x), \dots, y_{N_j}(x)\} = \max_{k=1}^{C} \text{median}\{y_{1_k}(x), \dots, y_{N_k}(x)\}, \tag{1}$$

---

[1] 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.
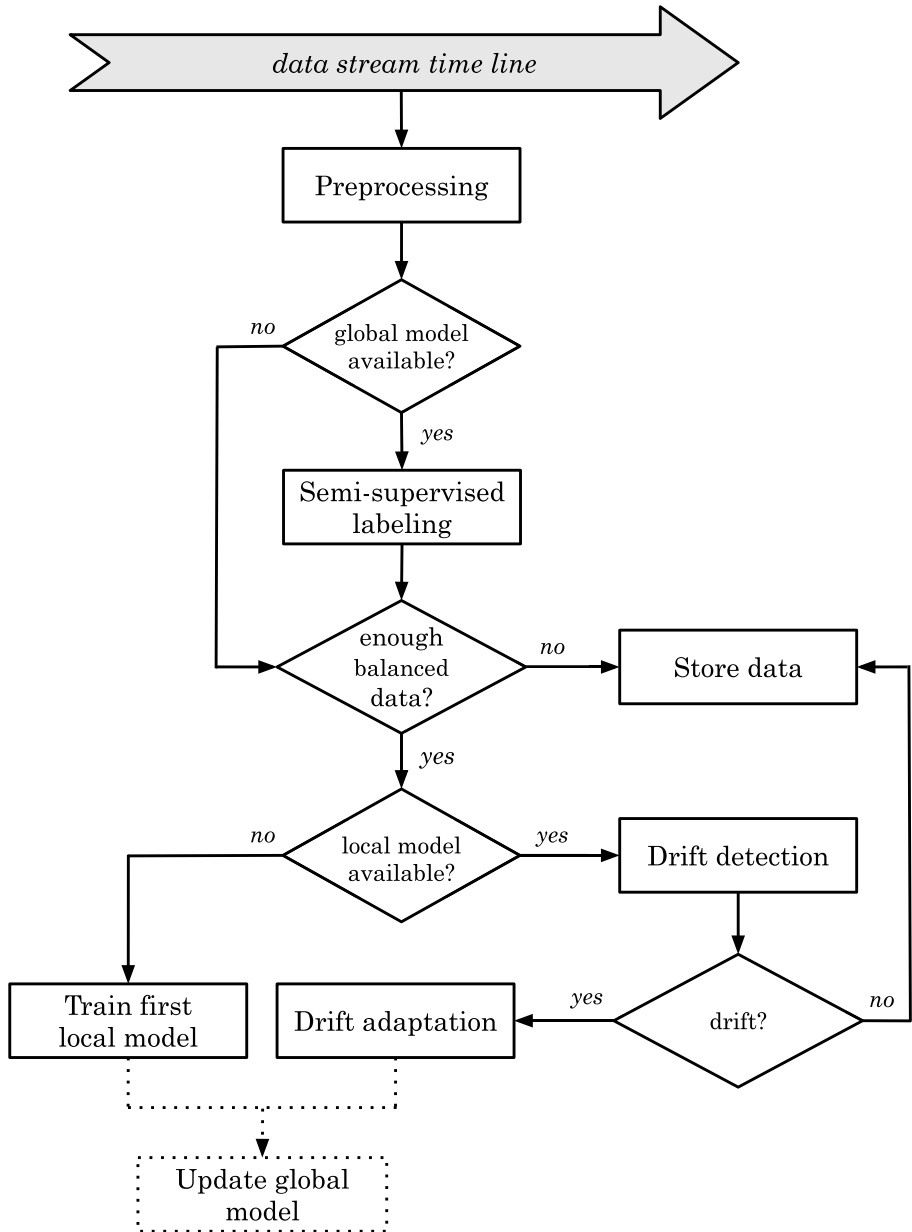
Fig. 2 Work flow on a local device

where C is the number of possible classes $(c_1, c_2, \ldots, c_C)$, N is the number of base classifiers, and $y_i = \{y_{i_1}(x), \ldots, y_{i_C}(x)\}$ is the output of the i-th classifier, for $i = 1, \ldots, N$.

As we can see in Fig. 2, the device gathers data and, if there is a global model available, it uses it to annotate the unlabeled samples. In particular, we perform what is called *semi-supervised transduction*. As soon as a new unlabeled instance is available, we take

advantage of the knowledge shared by the other devices and use the latest version of the global model to predict which is the most likely class label. Then, we filter the predictions based on their degree of confidence. We define the confidence of a prediction as the classifier conditional posterior probability, i.e., the probability $P(c_i|x) \in (0, 1)$ of a class c_i from one of the C possible classes $\{c_1, c_2, \dots, c_C\}$ to be the correct class for an instance x. We will explain how the confidence of the global model is obtained in Sect. 6. 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 ($\gamma \geq 0.95$) may allow to add very few examples to the labeled set. See the experimental results for more details (Sect. 7).

We keep collecting data until we have enough instances to perform the training of the first base classifier for the local model. The data is stored for a limited time in a sliding window W of maximum size N_{max}. This memory W follows the FIFO (First In, First Out) rule. A common question in batch learning is how much data is necessary for training. Unfortunately, the answer is not simple. In fact, it will depend on many factors, such as the complexity of the problem and the complexity of the learning algorithm (Jain & Chandrasekaran, 1982; Raudys & Jain, 1991). 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. To ensure a minimally balanced training set with representation of all classes, in this work we establish a single heuristic rule that must be fulfilled in order to allow the training of a base classifier. We define a minimum amount of labeled data, L, so that there must be at least $L/(2C)$ examples from each class in W to allow the training process, where C is the number of possible classes. Formally, each client keeps collecting data until the following condition is met:

$$\forall c_i \in \{c_1, c_2, \dots, c_C\} : |\{(x, y) \in (X \times Y) \cap W : y = c_i\}| \geq \frac{L}{2C}, \tag{2}$$

where (x, y) is a data sample, being x the input features and y the true class label. The first base learner of the local ensemble can be trained as soon as this rule is met. 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 $L = 2\Delta$, where Δ is a strongly related parameter used during drift detection algorithm that we will expose in Sect. 5.1. We also use a maximum size $N_{max} = 20\Delta$ for the sliding window W.

Once the device has a local model, the remaining question is when this local model should be updated using the new data that is being collected. It makes no sense to update it if it is performing well. However, as we said before, data is usually non-stationary and evolves in time causing concept drifts, which damage model performance. Thus, as we will show in Sect. 5.1, we will update the local model when a concept drift is detected.

Algorithm 1 details the complete local learning process. As soon as a new data sample (x, y) is available, we classify it using the global model (if any), obtaining the confidence ς that x corresponds to class label $\hat{y}$ (lines 8–19 in the pseudocode). We do this regardless of whether we know the actual label, y, because we will use the confidence record later for drift detection. Both instance and confidence are stored in W

(line 20). Depending on whether we know the true label or not, we will store y or its estimate $\hat{y}$. Note that $W = \{w_1, w_2, \ldots, w_N\}$ is a history of 3-tuples of the form "[features, label, confidence]". The number of 3-tuples that can be stored in W is given by N_{max}. Once this maximum size is reached, adding a new element to W implies deleting the oldest one (lines 5–7). When the data in W meet the condition from Eq. (2) (line 21), we can proceed to train the first base classifier, if our local ensemble is empty (line 32), or move into drift detection and adaptation, if we already have a local model (lines 23–30).

Algorithm 1: Ensemble and Continual Federated Learning (ECFL), client side.

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

    Output : None.

1 $W \leftarrow \{\varnothing\}$ // Initialize the sliding window

2 $E_l \leftarrow \{\varnothing\}$ // Initialize the local model (an ensemble, for now empty)

3 **while** *true** **do**

4   **if** *new data instance, (x, y), is observed* **then**

5   **if** $|W| = N_{max}$ **then**

6              $W \leftarrow W \setminus \{w_1\}$ // Remove the oldest element in W

7   **end**

8   **if** $E_G \neq \{\varnothing\}$ **then**

9              $[\hat{y}, \varsigma] \leftarrow \texttt{classify}(E_G, x)$ // Classify the pattern

10   **if** $y \neq \varnothing$ **then**

11                $w_{new} \leftarrow [x, y, \varsigma]$

12   **else if** $\varsigma \geq \gamma$ **then**

13                $w_{new} \leftarrow [x, \hat{y}, \varsigma]$ // Apply semi-supervised labeling

14   **else**

15                $w_{new} \leftarrow [x, \varnothing, \varsigma]$

16   **end**

17   **else if** $y \neq \varnothing$ **then**

18             $w_{new} \leftarrow [x, y, \varnothing]$

19   **end**

20          $W \leftarrow W \cup w_{new}$

21   **if** $\forall c_i \in \{c_1, \ldots, c_C\} : |\{(x, y) \in (X \times Y) \cap W : y = c_i\}| \geq \frac{L}{2C}$ **then**   // Condition from Eq. (2)

22   **if** $E_l \neq \{\varnothing\}$ **then**

23                $r \leftarrow \texttt{random}(0, 1)$ // Generate random number between 0 and 1

24   **if** $e^{-2\varsigma} \geq r$ **then**

25                    $d \leftarrow \texttt{driftDetection}(W, \lambda, N_{max}, \Delta)$ // Check for drift (Alg. 2)

26   **if** *d is true* **then**

27                      $E_l \leftarrow \texttt{localUpdate}(W, E_l, M_l)$ // Update local model (Alg. 3)

28                      $W \leftarrow \{\varnothing\}$ // Reinitialize the sliding window

29   **end**

30   **end**

31   **else**

32               $E_l \leftarrow \texttt{localUpdate}(W, E_l, M_l)$ // Learn first base classifier (Algorithm 3)

33   **end**

34   **end**

35   **end**

36 **end**

A drift detector (line 25) is in charge of analyzing possible changes in the data distribution and reporting them. In case a drift is detected, drift adaptation is applied (line 27) and the sliding window W is reinitialized (line 28). This concept drift management can be a bottleneck if we have to execute it after each new instance is processed. Therefore, we restrict the number of executions, so that we check for drift with a probability of $e^{-2\varsigma}$ (line 24). Hence, the higher the confidence, the lower the probability of executing the change analysis.

5.1 Drift detection

Drift detection refers to the techniques that characterize and quantify concept drift via identifying change points in data distribution. In our proposal, if a concept drift is identified it means that the global model is no longer a good abstraction of the knowledge of the devices, so it must be updated. Drift detection algorithms are typically classified into three categories (Lu et al., 2018): (1) error rate-based, (2) data distribution-based, and (3) multiple hypothesis test methods. The first class focuses on tracking changes in the online error of the classifier. The second uses a distance function or metric to quantify the dissimilarity between the distribution of historical and new data. The third group combines techniques from the two previous categories in different ways. Error rate-based methods operate only on true labeled data, because they need the labels to estimate the error. Therefore, in order to take advantage of both labeled and unlabeled data, in this work we decided to use a distribution-based algorithm, which do not present this restriction. Thus, we developed a detection mechanism inspired in the technique originally designed by Haque et al. (2016), which is a CUSUM-type method that works with *beta* distributions (Baron, 1999).

Algorithm 2 outlines our detection method. We propose to detect changes in the confidence of the predictions provided by the current global model for the instances in W. For that, we divide W into two sub-windows for every pattern k between Δ and $N - \Delta$ (lines 3–6), where N is the total number of examples in W. Let W_a and W_b be the two sub-windows, where W_a contains the most recent instances and their confidences. Each sub-window is required to contain at least Δ examples to preserve statistical properties of a distribution. When a drift occurs, confidence scores are expected to decrease. Thus, only changes in the negative direction are required to be detected. In other words, if m_a and m_b are the mean values of the confidences in W_a and W_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 use $\lambda = 0.05$ and $\Delta = 100$ in our experiments, since these values are widely used in the literature (Haque et al., 2016).

Algorithm 2: Change detection algorithm (`driftDetection`).

Input : Sliding window W, new data instance x, sensitivity to change λ, maximum size N_{max} for the sliding window, and padding Δ.

Output : Boolean indicating whether a drift is detected or not.

1   $s_f \leftarrow 0$
2   $T_h \leftarrow -\log(\lambda)$
3   $N \leftarrow |W|$
4   **for** $k \leftarrow \Delta$ **to** $(N - \Delta)$ **do**
5      $m_b \leftarrow \texttt{mean}(\varsigma_1 : \varsigma_k \in W)$
6      $m_a \leftarrow \texttt{mean}(\varsigma_{k+1} : \varsigma_N \in W)$
7   **if** $m_a \leq (1 - \lambda) \cdot m_b$ **then**
8         $s_k \leftarrow 0$
9         $[\hat{\alpha}_b, \hat{\beta}_b] \leftarrow \texttt{estimateParams}(\varsigma_1 : \varsigma_k \in W)$ // Get parameters of beta distribution
10        $[\hat{\alpha}_a, \hat{\beta}_a] \leftarrow \texttt{estimateParams}(\varsigma_{k+1} : \varsigma_N \in W)$
11   **for** $i \leftarrow k+1$ **to** N **do**
12            $s_k \leftarrow s_k + \log\left(\dfrac{f\left(\varsigma_i \in w_i \mid \hat{\alpha}_a, \hat{\beta}_a\right)}{f\left(\varsigma_i \in w_i \mid \hat{\alpha}_b, \hat{\beta}_b\right)} \right)$
13   **end**
14   **if** $s_k > s_f$ **then**
15           $s_f \leftarrow s_k$
16   **end**
17   **end**
18   **end**
19   **if** $s_f > T_h$ **then**
20   **return** *true*
21   **else**
22   **return** *false*
23   **end**

We can model the confidence values in each sub-window, W_a and W_b, as two different *beta* distributions. However, the actual parameters for each one are unknown. The proposed algorithm estimates these parameters at lines 9 and 10. Next, the sum of the log likelihood ratios s_k is calculated in the inner loop between lines 11 and 13, where $f\left(\varsigma_i \in w_i \mid \hat{\alpha}, \hat{\beta}\right)$ is the probability density function (PDF) of the *beta* distribution, having estimated parameters $\left(\hat{\alpha}, \hat{\beta}\right)$, applied on the confidence ς_i of $w_i = [x_i, y_i, \varsigma_i] \in W$. The variable s_k is a dissimilarity score for each iteration k of the outer loop between lines 4 and 18. The larger the difference between the PDFs in W_a and W_b, the higher the value of s_k (line 12). Let k_{max} is the value of k for which the algorithm calculated the maximum s_k value where $\Delta \leq k \leq N - \Delta$. Finally, a change is detected at point k_{max} if $s_{k_{max}} \equiv s_f$ is greater than a threshold T_h (line 19), being $T_h = -\log(\lambda)$.

5.2 Local update

Once a drift is detected, the local model should be updated according to that drift. There exist three main groups of drift adaptation methods (Lu et al., 2018): (1) simple retraining, (2) ensemble retraining, and (3) model adjusting. The first strategy is to simply train a new model combining in some way the latest data and the historical data to replace the obsolete model. The second one preserves old learners in an ensemble and when a new one is trained, it is added to the ensemble. The third approach consist of developing a model that adaptatively learns from the changing data by partially updating itself. This

last strategy is arguably the most efficient 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. Instead, we apply ensemble retraining. In particular, as we already mentioned before, we propose a rule-based ensemble using the median rule from Eq. (1). Therefore, we conceive each local model as an ensemble of base historical classifiers. Recall that, for simplicity, we refer to this ensemble simply as the local model of the device.

Algorithm 3 details our adaptation method. Each device is allowed to keep up to M_l base classifiers, which will make up its local model. The new base learner will be trained using only the labeled data stored in the sliding window W (lines 1 and 2 in the pseudocode). 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). Hence, with this strategy we also face the infinite length problem, as a constant and limited amount of memory will be enough to keep both training data and the model. In our experiments, we evaluated ECFL for different values of M_l (see Sect. 7).

Algorithm 3: Change adaptation algorithm (`localUpdate`).

Input : Sliding window W, local ensemble $E_l = \{l_1, \ldots, l_n\}$, maximum size of the local ensemble M_l.

Output : E_l.

1 $D \leftarrow$ `getLabeledPatterns`(W) // Get those patterns that already have a label (either real or obtained with semi-supervised transduction)
2 $l_{new} \leftarrow$ `train`(D) // Train a new base classifier
3 **if** $|E_l| = M_l$ **then**
4 $\quad$ | $\quad E_l \leftarrow E_l \setminus \{l_1\}$ // Remove the oldest base classifier in the ensemble
5 **end**
6 $E_l \leftarrow E_l \cup l_{new}$ // Add the new base classifier to the local ensemble
7 **return** E_l

---

6 Global learning

The global model is an ensemble that integrates a selection of local models. As in the local level, any state-of-the-art ensemble technique could be used to combine the predictions of the local models. There are some interesting choices, such as *stacking* (Wolpert, 1992). Stacking tries to induce which base classifiers are reliable and which are not by training a meta-learner. This meta-learner is a higher-level model which is trained on a meta-dataset composed from the outputs of all base classifiers on a given training set. However, it presents a great limitation, which is the need to have this meta-dataset available in advance. In our context of devices, this would involve gathering a certain amount of labeled data from all users in the cloud, which is not feasible. Therefore, a better solution is to use a simpler but equally effective rule-based ensemble, as we already do at the local level (Sect. 5). In this case, the optimal combination rule is the *product rule*, because each local classifier operates in a different measure space (Kittler et al., 1998): different environments and users. The product rule predicts that an instance x belongs to class c_j if the following condition is met:

$$\prod_{i=1}^{N} y_{i_j}(x) = \max_{k=1}^{C} \prod_{i=1}^{N} y_{i_k}(x), \tag{3}$$

where C is the number of possible classes $(c_1, c_2, \ldots, c_C)$, N is the number of base classifiers, and $y_i = \{y_{i_1}(x), \ldots, y_{i_C}(x)\}$ is the probabilistic output of the i-th classifier, for $i = 1, \ldots, N$. The global ensemble is shared with all clients so that everyone can use it. Note that, unlike in other federated learning solutions, we aggregate the outputs of the local models, and not the models themselves. Therefore, each time a client uses the global model, the product rule from Eq. (3) is applied. Also note that the outputs of this equation are the estimated posterior probabilities that we use on each device as confidence values to decide which unlabeled patterns can and cannot be labeled. Therefore, the elements y_{i_k} are always greater than 0, hence their product and maximum is also greater than 0.

Although the time complexity of using the global model 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. As the global model is sent back to local devices, it would also have a negative impact on the bandwidth and computational requirements of the clients. In addition, assuming that there is an optimal global abstraction of knowledge, not all the local models will bring the same wealth to the ensemble. On the contrary, there will be devices that, accidentally or intentionally, may be creating local models with poor performance, which should be detected in time so as not to participate 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 each new incoming update. The computation of the scores is based on the Effective Voting (EV) technique (Tsoumakas et al., 2004). EV propose to perform 10-fold cross validation for the evaluation of each model and then apply a paired t-test for each pair of models to evaluate the statistical significance of their relative performance. Finally, the most significant ones are selected.

In distributed contexts, involving multiple devices, skewness and bias are frequent, so cross-validation is not a fair way to evaluate each local model. Instead, 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 q performance measures for that model. Both p and q parameters depend on the maximum ensemble size, M_g, and the total number of devices available online, N, so that $M_g \leq q \leq p \leq N$. This process could be considered a distributed cross-validation. After gathering the q 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 models E_{l_i}, E_{l_j} so that:

$$t(E_{l_i}, E_{l_j}) = \begin{cases} 1 & \text{if } E_{l_i} \text{ is significantly better than } E_{l_j}, \\ -1 & \text{if } E_{l_j} \text{ is significantly better than } E_{l_i}, \\ 0 & \text{otherwise.} \end{cases} \tag{4}$$

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

$$S(E_{l_i}) = \sum_{j=1}^{M_g+1} t(E_{l_i}, E_{l_j}). \tag{5}$$

Finally, we select the new M_g models with the highest significance index or score, S. If there are ties, we break them by selecting the most accurate ones (we compute the mean accuracy from the q available evaluations). Figure 3 summarizes the whole selection process, that we call *Distributed Effective Voting* (DEV). As we will see in Sect. 7, 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 $p = q = M_g$. In any case, the global ensemble size will be dependent on the problem.

Algorithm 4 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).

Algorithm 4: Ensemble and Continual Federated Learning (`ECFL`), server side.

> **Input** : List of participant clients $K = \{k_1, k_2, \ldots, k_m\}$, maximum size of the global ensemble M_g.
> **Output** : Global model E_G.
> 1 $E_G \leftarrow \{\varnothing\}$ // Initialize the global model (an ensemble, for now empty)
> 2   **while** *true** **do**
> 3   Listen for client updates $\forall k_j \in K$
> 4   **if** $\exists k_j \in K$: *new local model E_j^u is received* **then**
> 5   **if** $\exists E_j^u : E_j^u \in E_G,\ u < t$ **then**
> 6 $E_G \leftarrow E_G \setminus \{E_j^u\}$ // Remove the previous model from user k_j
> 7 $E_G \leftarrow E_G \cup \{E_j^t\}$ // Add the new one
> 8   **else if** $|E_G| < M_g$ **then**
> 9 $E_G \leftarrow E_G \cup \{E_j^t\}$
> 10   **else**
> 11 $E_G \leftarrow$ `DistributedEffectiveVoting`(K, E_G, E_j^t)
> 12   **end**
> 13   `broadcast`(E_G, K) // Share the latest version of the global model with all the clients
> 14   **end**
> 15 **end**

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 q random clients.
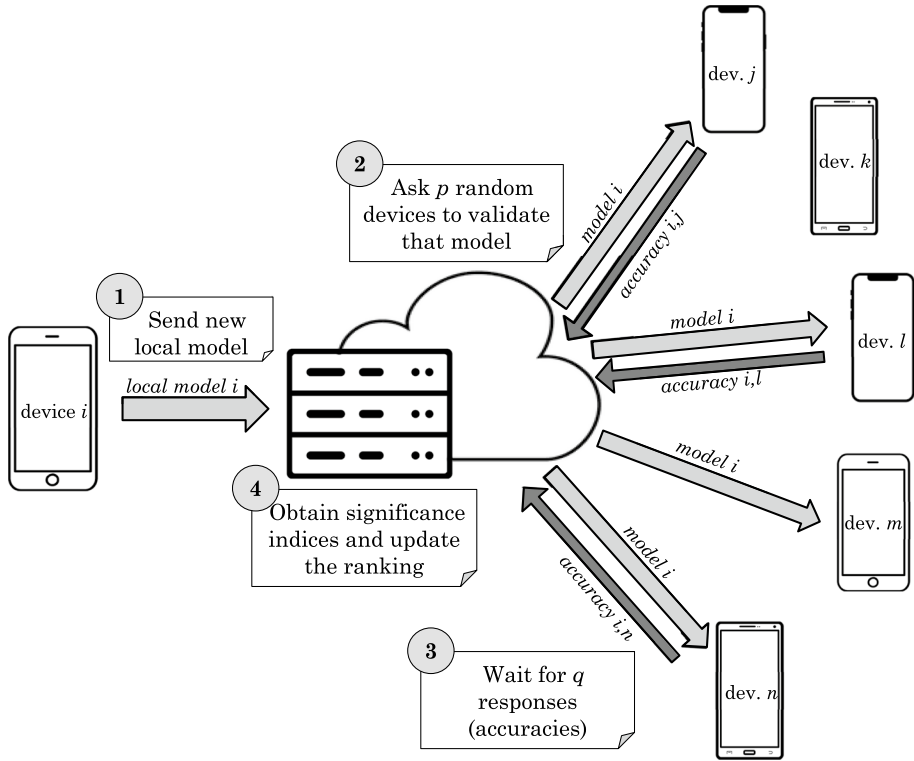
Fig. 3 Work flow of the distributed effective voting

Nevertheless, as we will see in Sect. 7, results indicate that the global ensemble always performs close to or better than the best local model. This leads us to think that perhaps the diversity on the global ensemble is more important than the accuracy of each of its members.

7 Experimental results

The aim of this section is to evaluate ECFL, while illustrating how it works and highlighting its main properties in different situations. In particular, we want to test the performance of the global model, obtained from the consensus of the local devices, in distributed, continual, heterogeneous, and semi-supervised classification scenarios. To this end, throughout the section, we will work on walking activity recognition on smartphones. For more in-depth results and additional evaluation on other datasets, see Appendices A and B.

We believe that walking recognition is a good benchmark to evaluate our proposal given its complexity and its natural fit in a collaborative and continual framework. It is relatively easy to detect the walking activity when a person walks carrying the smartphone in a fixed position. Nonetheless, 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. This has already been addressed applying standard (centralized) machine learning processes (Casado et al., 2020; Rodríguez et al., 2018). These approaches, however, 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 federated and continual learning approach can avoid such issues.

There are currently a number of public datasets for human activity recognition, but there are not many designed for experimentation in federated and continual learning. This means involving different participants, including metadata for each sample indicating its timestamp and the user and device it belongs to. Although we could synthetically modify some existing data, for this work we decided to create a new dataset to be more realistic. For that, we developed an Android application that samples and logs inertial data (accelerometer and gyroscope) on the phone continuously. The app allows the user to indicate whether he/she is walking or not through a switch button in the graphical interface. This is optional, so depending on the user's willingness to participate, there will be more or less labeled data. The app also labels autonomously some examples applying a series of heuristic rules when it comes to clearly identifiable positives or negatives (e.g., when the phone is at rest). With this app, we collected partially labeled data from 10 different people. Participants installed our application and recorded data continuously while they were performing their usual routine.

We have used the entire dataset obtained with our app for training. For testing, we have employed the Walking Recognition Dataset (WRD), taken from the literature (Casado et al., 2020). It is a fully labeled dataset that includes recordings from 77 different people. Several features were extracted from the raw time series in both cases, ending with almost 70,000 patterns for training and 8000 for testing. Further details on software, data distribution, and preprocessing can be found in Appendix A.

7.1 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 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 C.

Next, we addressed the problem in a federated manner, forcing the data and the learning to be distributed among the 10 clients. In this situation, none of the algorithms from Table 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 feed-forward neural networks using the two most popular FL methods: (a) FedAvg, and (b) FedProx. The performance of each

global model is shown in Table 2. If we compare these results with those in Table 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 in this experiment we still did not include temporal restrictions, so that each client had access to all its data at any time.

7.2 Continual federated setting

Both the results in Tables 1 and 2 are useful to get an idea of the performance of the classifiers under ideal conditions. Having this baseline, we finally 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 by the Android app. 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 1). We also executed FedAvg and FedProx in this setting. 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, ECFL uses a sliding window, W, of limited size during learning. 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 evaluated all models (local and global) over time, after each local update, always using the same test dataset (WRD). We repeated each execution 10 times, randomly varying the iteration in which each client joins the learning. Table 3 shows, for each of the configurations, the average performance obtained by the global model after the 10 executions. That is, Table 3 provides the mean accuracy obtained with the global model at the end of the data stream, once the latest update has been performed and there is no more data left to process in any of the devices. 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 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 6, 7, and 8 in Appendix A.

The comparison of the results in Tables 1 and 2 with those of Table 3 is not fair, since the 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 classifiers, providing similar performances. Especially noteworthy is the case where SVMs are used as base classifiers, since it outperforms both the single model from Table 1 and the ensemble from Table 2. We can also see that, while still performing quite well, both FedAvg and FedProx

Table 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

Table 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

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, and all the models in Tables 1 and 2 are trained using all labeled data from all clients. The results in Table 3 demonstrate the robustness of ECFL in demanding scenarios.

To better illustrate the learning process in ECFL, we will now show in detail one of the executions from Table 3 as an example. We will select the case where we use SVMs as base classifiers. Figure 4 details the evolution of the performance per client and over time. The upper graphs shows the accuracies of all the models—from each of the 10 users and also the global one—when evaluated on the test dataset. 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 (∘) on the line indicates when 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 as one of the 5 models of the global ensemble—the other 4 chosen are marked with a cross (✗)—. 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 Fig. 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

Table 3 Average performance of ECFL (using different base classifiers), FedAvg, and FedProx, 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)

The most relevant results are highlighted in bold

approach explained in Sect. 5. 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 Appendix B, where we use another dataset that contains annotated information on when a change occurs. Regarding the construction of the global ensemble, it should be reminded that local models are chosen using the Distributed Effective Voting (DEV) method, explained in Sect. 6. DEV is based on the voting carried out by a subset of q users randomly chosen (5 in this case, because $q = M_g = 5$). The results presented here, as well as the extended evaluation in Appendix A, demonstrate that the global ensemble provides excellent performance, always similar or greater than that of any local model.

Another important aspect to highlight in Fig. 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 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 cases, the global model takes approximately 10 rounds to converge, which is roughly the half of the data stream.

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 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 shows the average performance of ECFL, FedAvg and FedProx in this continual setting with outliers. As can be appreciated, in this case our proposal far outperforms the results provided by the other approaches.

Figure 6 shows an example of executing ECFL in this scenario, in the particular case where SVM is used as base classifier. The results are comparable to those shown in Fig. 4.
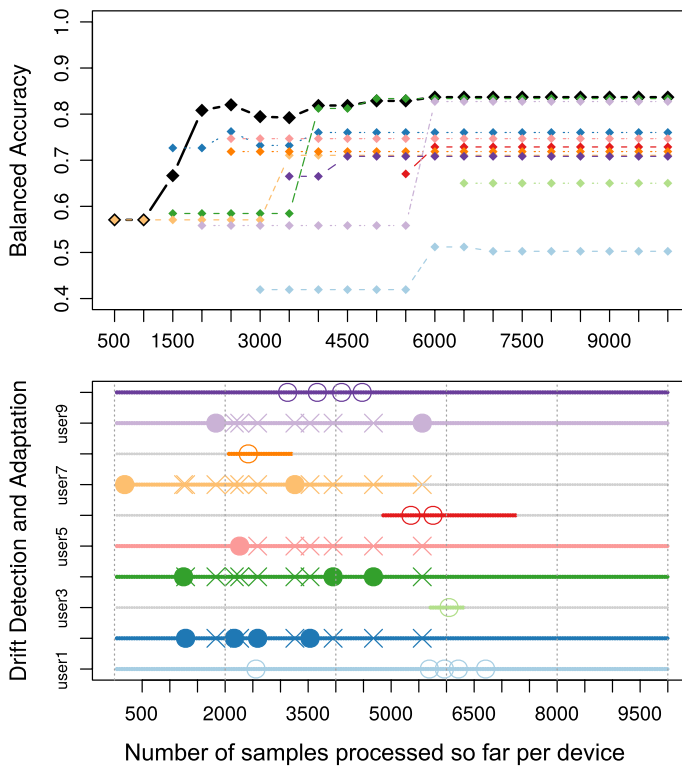
Fig. 4 Example of executing ECFL with SVM as base classifier. The upper graph shows the evolution of the accuracy over time. The black line corresponds to the global model, while the others are the clients. The bottom graph shows the local updates and global selection

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

8 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 upload to the cloud or explicit sharing of raw data. In addition, the entire database is naturally segmented into local storages, from different owners, which makes it more difficult to hack.

However, it should be noted that our approach, in its simplest implementation, 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
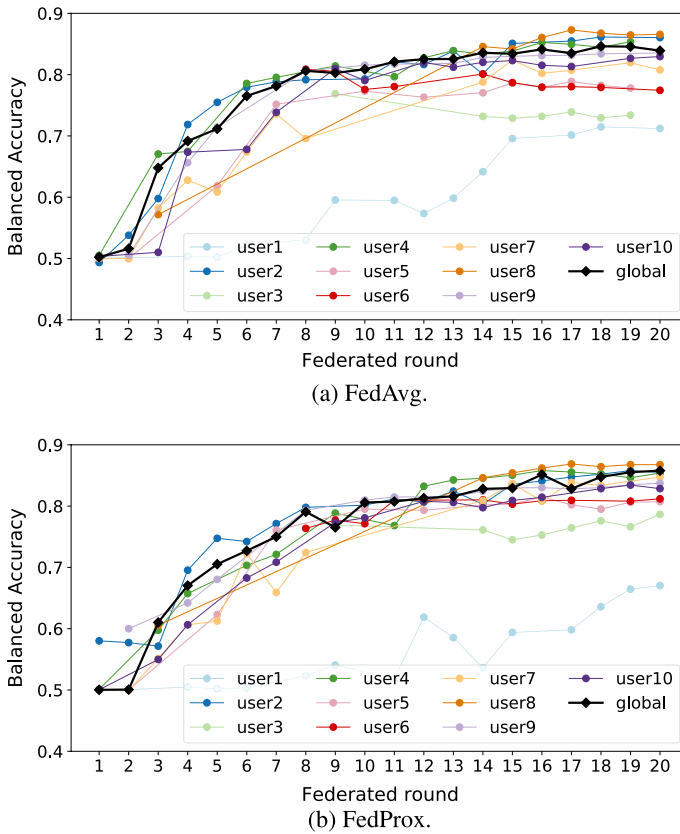
(a) FedAvg.

(b) FedProx.

Fig. 5 Example of execution of FedAvg and FedProx in the federated and continual setting

can lead to privacy leaks due to, for example, the reverse engineering that can be done on the models (Nasr et al., 2019; Wang et al., 2019b). To avoid this, there are several protection mechanisms that can be added as additional layers of security. Firstly, differential privacy (Dwork, 2008; Augenstein et al., 2019) can be introduced for client models, so that each participant could have a personalised privacy budget. In addition, there are strategies based on Secure Multi-party Computation (SMC) (Canetti et al., 1996), where communications are secured and protected with cryptographic methods. One example of this is homomorphic encryption (Aono et al., 2017), which allow operations to be performed directly on encrypted data and models without the need to decrypt them. Finally, in our particular case we must highlight the fact that the server acts as a central orchestrator, but it can carry out its tasks without the need 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

Table 4 Average performance of ECFL, FedAvg, and FedProx, 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)

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.

9 Conclusions

In this paper we have presented ECFL, a novel architecture for continual and federated classification using ensemble techniques. Our approach allows to build an ensemble composed of different local models, which are trained in a distributed way on different client devices. The ensemble is the global, shared model, which can be used by the devices and help them to label unlabeled patterns to improve performance. This architecture poses several advantages for federated learning, such as its simplicity, flexibility, or generalizability. Furthermore, it allows to work with problems that require continual adaptation, something that is still beginning to be addressed in research. Thus, our proposal is robust to situations where data is distributed among multiple clients and is also non-stationary, evolving over time and causing concept drifts. We have tested ECFL in different tasks and under diverse conditions, showing that it is able to provide results that are comparable to or even better than those obtained using other state-of-the-art methods trained in ideal conditions (i.e., data is centralized, static, and IID).

Our approach has a series of components that are clearly separable: local training, semi-supervised labeling, drift detection and adaptation, global selection of candidates, and global aggregation. We believe that new proposals can be made to improve some of these modules. For example, we would like to explore optimal ways to combine models in the ensemble, study effective ways to perform distributed feature selection, or analyze the use of the global model for instance selection in the local devices. It could also be of interest to take advantage of our architecture in the context of explainable artificial intelligence (XAI), given that the use of ensembles is compatible with the use of local learners that facilitate the explanation of their decisions. Another promising feature could be to have some personalization system, so that the global model would be able to adjust to local particularities. In addition, it remains open to explore the privacy enhancements mentioned in the previous section. There is undoubtedly a whole world of possibilities to be discovered in continual and federated learning, and we believe that this paper, in addition to providing a novel architecture in this field, opens up new paths of exploration in it.
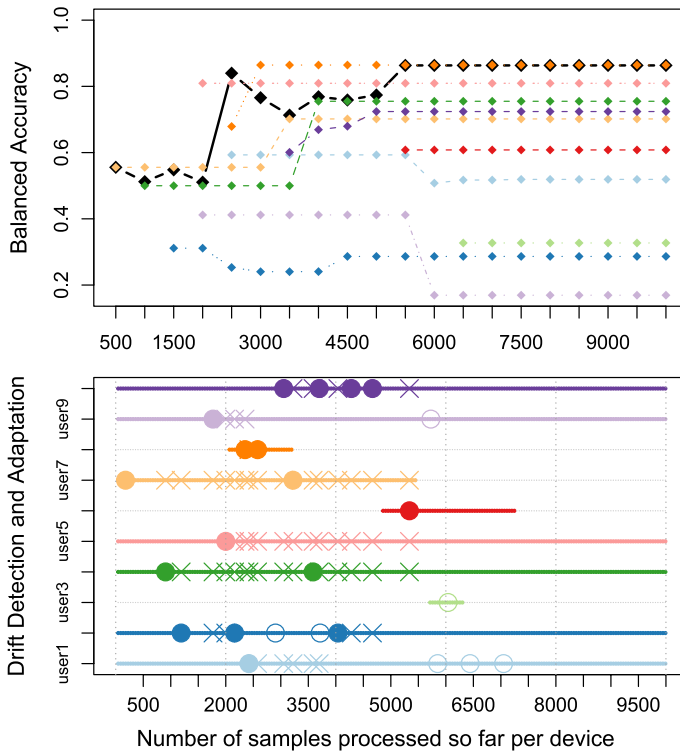
Fig. 6 Example of executing ECFL when 4 clients mislabel the data. The upper graph shows the evolution of the accuracy. The black line corresponds to the global model, while the others are the clients. The bottom graph shows the local updates and global selection

A: Extended results on walking recognition

In this appendix we provide further details on the experiments performed on the walking recognition problem, which were omitted in Sect. 7 for ease of reading.

A.1: Data preprocessing and distribution

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, although in the case of the training set this information is partial.

Before learning any classifier, we performed the signal preprocessing originally proposed in Casado et al. (2020), given the good performance demonstrated in the past. Basically, the signal was filtered and centered by applying a 10th order Butterworth filter with a 3 Hz cut-off frequency and then a DC-bias filter. The resulting signal was split into windows of 250 measurements, which is equivalent to 2.5 s. Then, each of these windows was transformed into a pattern. For that, we applied feature extraction, calculating 21
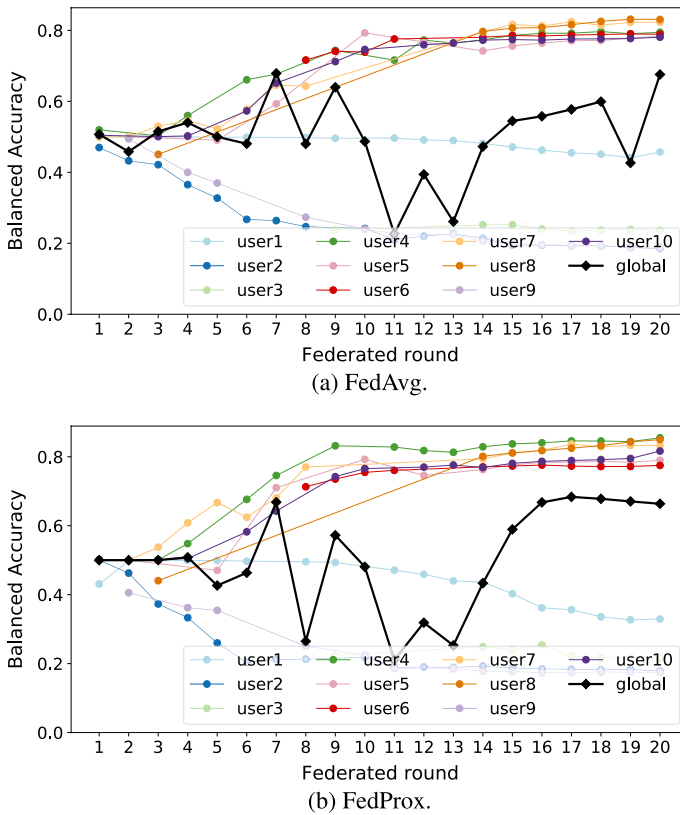
(a) FedAvg.

(b) FedProx.

Fig. 7 Example of execution of FedAvg and FedProx in the federated and continual setting when 4 clients mislabel the data

state-of-the-art features for each window. We selected diverse and representative features in both the time and frequency domains, including the standard deviation for each axis of the acceleration, the variance of the angular velocity, the peak count of the acceleration norm, the skewness of the angular velocity norm, and the modal frequency of the acceleration norm, among others. Readers interested in more details on preprocessing are referred to Casado et al. (2020). After this, we obtained the final sets used in our experiments, where each instance corresponds to 2.5 s of activity. Table 5 summarizes the data distribution in both datasets, attending to the class labels. For training data, a breakdown by user is provided.

A.2: Full results

In the following, we provide an exhaustive evaluation of our approach 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

Table 5 Summary of the train and test data distribution

		Walking	Not walking	Unlabeled	Total
Training set	User 1	3130	2250	4620	10,000
	User 2	2519	4359	3122	10,000
	User 3	186	325	125	636
	User 4	2432	2455	5113	10,000
	User 5	233	2785	6982	10,000
	User 6	554	1821	69	2444
	User 7	2582	2052	769	5403
	User 8	232	678	229	1139
	User 9	1151	2669	6180	10,000
	User 10	2329	2669	5002	10,000
	Total	15348	22063	32211	69,622
Test set		6331	1586	0	7917

client joins the learning. For each setting, we provide the mean and standard deviation of the balanced accuracy.

Table 6 is an extended version of Table 3 (Sect. 7), and shows the influence of the base classifier selection in the results. All other parameters were kept constant, being $M_l = M_g = 5$, $\gamma = 0.9$, and $L = 200$. We provide, for each configuration, the average accuracy obtained by the global model after 10 different executions. In this extended version we also include the average accuracy of each of the local models and an additional column with the mean accuracy of all local models. Table 6 empirically demonstrates that the global model always performs similar or better than the best of the local ones.

It is important to mention that, despite the good performance, the use of Random Forest 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 combining the final decisions of each of the forests. Proposals like this are out of the scope of this work, in which we have focused on a global learning method scalable to multiple settings.

Table 7 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.

Finally, in Table 8 we present the influence of the confidence threshold, γ, and the minimum amount of training data, L. In general, 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

Table 6 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	Local mean	Global model
GLM	0.736 (±0.008)	0.737 (±0.047)	0.625 (±0.066)	0.672 (±0.100)	0.684 (±0.042)	0.743 (±0.066)
Naïve Bayes	0.500 (±0.000)	0.838 (±0.019)	0.761 (±0.008)	0.679 (±0.010)	0.706 (±0.005)	0.789 (±0.014)
C5.0 Tree	0.533 (±0.111)	0.733 (±0.023)	0.583 (±0.006)	0.547 (±0.084)	0.617 (±0.036)	0.795 (±0.037)
SVM	0.708 (±0.002)	0.863 (±0.001)	0.684 (±0.149)	0.684 (±0.022)	0.692 (±0.020)	0.857 (±0.032)
RF	0.682 (±0.006)	0.815 (±0.004)	0.677 (±0.014)	0.769 (±0.010)	0.729 (±0.007)	0.845 (±0.011)
SGB	0.667 (±0.106)	0.775 (±0.044)	0.697 (±0.075)	0.726 (±0.124)	0.700 (±0.065)	0.833 (±0.033)
FNN	0.763 (±0.013)	0.703 (±0.094)	0.663 (±0.051)	0.688 (±0.042)	0.692 (±0.026)	0.803 (±0.027)

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.

B: Additional experiments on the HAR multi-class dataset

We further provide additional results in a different task and data. We selected the dataset from Shoaib et al. (2014), which is another popular benchmark for Human Activity Recognition (HAR) on smartphones. Unlike in the previous case, this one poses a multi-class classification problem. It includes data of 10 different people performing seven physical activities: walking, going upstairs, going downstairs, sitting, standing, jogging, and biking. The data is fully labeled for all the participants. It was collected at 50 Hz and includes readings from accelerometer, gyroscope, and magnetometer.

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 (Fig. 8): (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. We will take advantage of this to analyze in greater detail the performance of ECFL in response to drifts.

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 s). 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.1). After that, each client has a total of 5000 samples, 1000 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 5000 for testing.

Similarly to what we did in Sect. 7, 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 state-of-the-art federated methods, FedAvg and FedProx. Table 9 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 seen that, for this task, the performance of federated methods lags slightly behind that of centralized methods.

Next, in order to test ECFL, we configured a federated and continual setting. Once again, we generated an evolving data stream for each of the clients. All clients worked at the same frequency, starting at the same time, so the data streams lasted 5000 iterations. In real life, data is often non-IID and evolves over time. Thus, we decided to evaluate accuracy under concept drift. For that, we took advantage of having the information about the position of the smartphone and we sorted the data of all the users using this as criteria. Data are sorted 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 force a non-stationary distribution that changes a total of 4 times. Note that this is not a totally realistic situation because, in real life, each client would acquire data in a different manner. Nonetheless, it is helpful to bound the changes and evaluate their impact during training.

Table 7 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	Local mean	Global
1	1	0.728 (±0.032)	0.789 (±0.008)	0.680 (±0.085)	0.637 (±0.041)	0.660 (±0.050)	0.734 (±0.053)
1	3	0.732 (±0.031)	0.865 (±0.010)	0.666 (±0.115)	0.665 (±0.063)	0.689 (±0.054)	0.769 (±0.069)
1	5+	0.856 (±0.031)	0.865 (±0.010)	0.720 (±0.047)	0.674 (±0.051)	0.708 (±0.044)	0.769 (±0.069)
3	1	0.710 (±0.030)	0.759 (±0.012)	0.518 (±0.014)	0.691 (±0.008)	0.647 (±0.022)	0.791 (±0.053)
3	3	0.711 (±0.007)	0.864 (±0.000)	0.525 (±0.006)	0.671 (±0.043)	0.672 (±0.009)	0.800 (±0.051)
3	5+	0.761 (±0.007)	0.864 (±0.000)	0.589 (±0.079)	0.697 (±0.035)	0.692 (±0.016)	0.840 (±0.034)
5	1	0.708 (±0.007)	0.760 (±0.012)	0.513 (±0.005)	0.684 (±0.010)	0.645 (±0.023)	0.845 (±0.047)
5	3	0.752 (±0.000)	0.863 (±0.002)	0.612 (±0.001)	0.673 (±0.134)	0.683 (±0.019)	0.852 (±0.043)
5	5+	0.752 (±0.002)	0.863 (±0.001)	0.684 (±0.149)	0.689 (±0.022)	0.697 (±0.020)	0.857 (±0.032)
7	1	0.707 (±0.010)	0.777 (±0.014)	0.528 (±0.026)	0.689 (±0.013)	0.656 (±0.017)	0.856 (±0.005)

Table 7 (continued)

M_g	M_l	User 7	User 8	User 9	User 10	Local mean	Global
7	3	0.743 (±0.001)	0.863 (±0.002)	0.650 (±0.166)	0.661 (±0.052)	0.684 (±0.024)	0.855 (±0.018)
7	5+	0.743 (±0.001)	0.863 (±0.002)	0.714 (±0.124)	0.691 (±0.027)	0.699 (±0.017)	0.868 (±0.018)
9	1	0.704 (±0.006)	0.761 (±0.013)	0.512 (±0.005)	0.689 (±0.009)	0.648 (±0.014)	0.840 (±0.009)
9	3	0.706 (±0.005)	0.864 (±0.002)	0.576 (±0.059)	0.633 (±0.017)	0.669 (±0.010)	0.847 (±0.014)
9	5+	0.740 (±0.004)	0.863 (±0.001)	0.664 (±0.022)	0.663 (±0.009)	0.690 (±0.006)	0.849 (±0.012)

Table 8 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	Local mean	Global
0.85	100	0.609 (±0.015)	0.826 (±0.045)	0.587 (±0.089)	0.577 (±0.003)	0.637 (±0.017)	0.689 (±0.062)
0.85	200	0.707 (±0.005)	0.863 (±0.001)	0.615 (±0.085)	0.670 (±0.009)	0.682 (±0.012)	0.861 (±0.013)
0.85	300	0.668 (±0.005)	0.869 (±0.000)	0.808 (±0.001)	0.733 (±0.005)	0.727 (±0.007)	0.812 (±0.032)
0.85	400	0.777 (±0.013)	0.873 (±0.008)	0.649 (±0.006)	0.800 (±0.010)	0.725 (±0.008)	0.878 (±0.006)
0.85	500	0.781 (±0.006)	0.874 (±0.002)	0.795 (±0.003)	0.807 (±0.005)	0.743 (±0.008)	0.873 (±0.003)
0.90	100	0.607 (±0.015)	0.826 (±0.045)	0.581 (±0.094)	0.577 (±0.003)	0.636 (±0.017)	0.680 (±0.047)
0.90	200	0.708 (±0.002)	0.863 (±0.001)	0.684 (±0.149)	0.684 (±0.022)	0.692 (±0.020)	0.857 (±0.032)
0.90	300	0.666 (±0.009)	0.867 (±0.003)	0.809 (±0.002)	0.733 (±0.005)	0.726 (±0.004)	0.825 (±0.019)
0.90	400	0.770 (±0.002)	0.869 (±0.001)	0.650 (±0.006)	0.802 (±0.006)	0.723 (±0.004)	0.874 (±0.009)
0.90	500	0.776 (±0.004)	0.873 (±0.000)	0.795 (±0.004)	0.806 (±0.005)	0.740 (±0.005)	0.874 (±0.001)

Table 8 (continued)

γ	L	User 7	User 8	User 9	User 10	Local mean	Global
0.95	100	0.607 (±0.015)	0.826 (±0.045)	0.581 (±0.094)	0.577 (±0.003)	0.636 (±0.017)	0.680 (±0.047)
0.95	200	0.707 (±0.001)	0.863 (±0.002)	0.592 (±0.084)	0.677 (±0.006)	0.682 (±0.011)	0.854 (±0.026)
0.95	300	0.668 (±0.002)	0.869 (±0.000)	0.808 (±0.002)	0.733 (±0.006)	0.726 (±0.004)	0.819 (±0.029)
0.95	400	0.770 (±0.007)	0.871 (±0.003)	0.651 (±0.009)	0.800 (±0.008)	0.722 (±0.004)	0.878 (±0.004)
0.95	500	0.772 (±0.005)	0.874 (±0.002)	0.796 (±0.004)	0.807 (±0.000)	0.741 (±0.007)	0.874 (±0.002)

Fig. 8 Positions in which the smartphone was placed during data recording in HAR
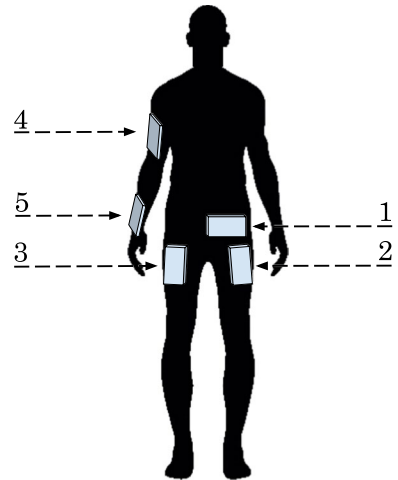
Table 9 Average accuracies of several classifiers, trained and tested in ideal conditions (static, IID)

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 NN	0.881 (±0.037)
FedAvg	0.806 (±0.043)
FedProx	0.800 (±0.047)

We ran ECFL using different base classifiers (the same as those evaluated under ideal conditions, Table 9). The hyperparameters were set to their default values: $M_l = M_g = 5$, $\gamma = 0.9$, and $L = 200$. We also executed FedAvg and FedProx in this scenario. Table 10 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.

To end this section, in Fig. 9 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.

Table 10 Average accuracies on HAR dataset for local and global models in ECFL using different base classifiers

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 (±0.630)	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 (±0.016)	0.737 (±0.059)	0.753 (±0.048)	0.746 (±0.040)	0.745 (±0.044)	0.713 (±0.055)

Method	User 7	User 8	User 9	User 10	Local mean	Global
ECFL (NB)	0.726 (±0.040)	0.691 (±0.033)	0.654 (±0.053)	0.720 (±0.059)	0.687 (±0.049)	0.736 (±0.039)
ECFL (C5.0)	0.649 (±0.073)	0.605 (±0.047)	0.709 (±0.046)	0.698 (±0.049)	0.689 (±0.056)	0.750 (±0.036)
ECFL (SVM)	0.754 (±0.062)	0.758 (±0.056)	0.738 (±0.057)	0.735 (±0.053)	0.761 (±0.050)	0.815 (±0.022)
ECFL (RF)	0.796 (±0.043)	0.800 (±0.091)	0.837 (±0.040)	0.863 (±0.032)	0.837 (±0.050)	**0.884** (±0.053)
ECFL (SGB)	0.703 (±0.056)	0.736 (±0.045)	0.745 (±0.043)	0.696 (±0.066)	0.718 (±0.056)	0.793 (±0.041)
ECFL (FNN)	0.736 (±0.035)	0.726 (±0.038)	0.713 (±0.057)	0.717 (±0.054)	0.748 (±0.046)	0.790 (±0.021)
FedAvg	0.771 (±0.059)	0.742 (±0.057)	0.744 (±0.045)	0.747 (±0.060)	0.747 (±0.051)	0.788 (±0.053)
FedProx	0.762 (±0.055)	0.738 (±0.060)	0.731 (±0.049)	0.751 (±0.063)	0.743 (±0.049)	0.790 (±0.009)

The most relevant results are highlighted in bold
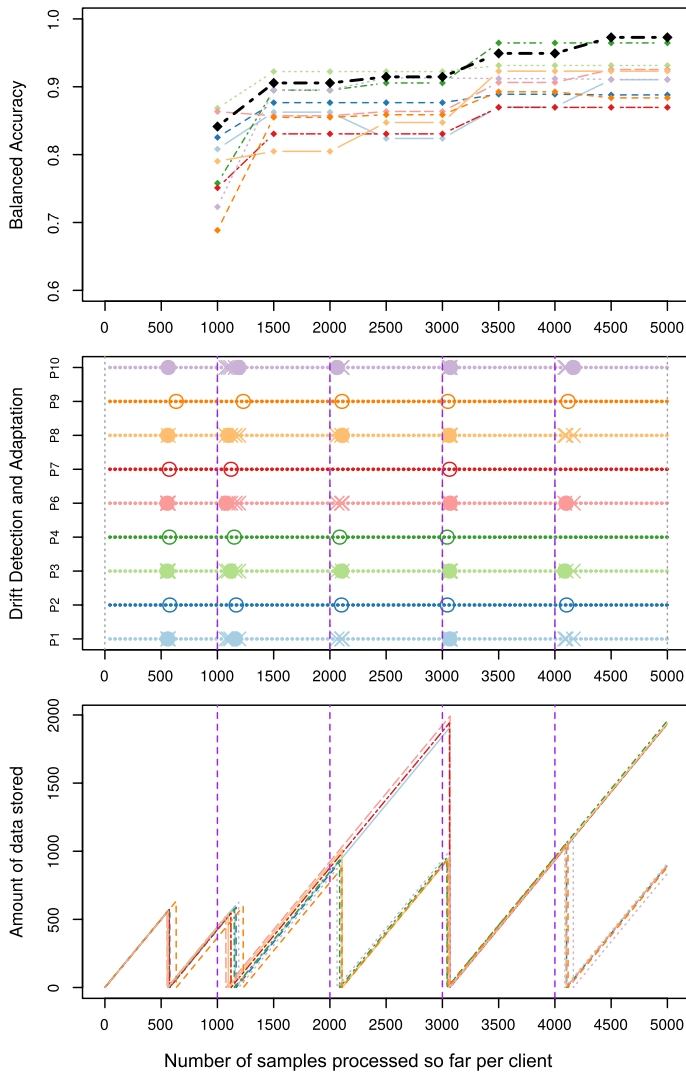
Fig. 9 Example of executing ECFL in the HAR task. The black line corresponds to the global model, while the rest of the colored lines are associated with one client

By looking at Fig. 9, 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 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 model 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 local model.

C: Implementation details

Here we provide additional details on the execution of the experiments.

C.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 walking recognition task was done using 10 mid-range smartphones from different manufacturers and running Android OS in different versions (between 7.0 and 9.0).

C.2: Software

The app used for data recording was implemented in Android SDK API 25. Data preprocessing, model training, and evaluation, were mainly performed using R programming 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 (Kuhn et al., 2020). The ECFL framework was also developed in R using `caret` to train the base classifiers. Instead, for FedAvg and FedProx, we used Python 3.6 together with TensorFlow 2.5 library.

C.3: Hyperparameters

To train the baseline models from Tables 1 and 9, a grid search for the optimal hyperparameters was performed using the methods already provided by `caret`. 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 feedforward neural networks had 3 hidden layers, with 32 neurons in each. When FedAvg and FedProx were tested under ideal conditions (Tables 2 and 9) they were allowed to train for 30 rounds, performing 3 local epochs per round and using all local models for global aggregation.

References

Ananthanarayanan, G., Bahl, P., Bodík, P., Chintalapudi, K., Philipose, M., Ravindranath, L., & Sinha, S. (2017). Real-time video analytics: The killer app for edge computing. *Computer, 50*(10), 58–67.

Androutsopoulos, I., Koutsias, J., Chandrinos, K.V., Paliouras, G., & Spyropoulos, C. D. (2000). An evaluation of naive bayesian anti-spam filtering. In *Proceedings of the workshop on machine learning in the new information age, 11th european conference on machine learning (ECML 2000)*.

Aono, Y., Hayashi, T., Wang, L., Moriai, S., et al. (2017). Privacy-preserving deep learning via additively homomorphic encryption. *IEEE Transactions on Information Forensics and Security, 13*(5), 1333–1345.

Arrieta, A. B., Díaz-Rodríguez, N., Del Ser, J., Bennetot, A., Tabik, S., Barbado, A., García, S., Gil-López, S., Molina, D., Benjamins, R., et al. (2020). Explainable artificial intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI. *Information Fusion, 58*, 82–115.

Augenstein, S., McMahan, H. B., Ramage, D., Ramaswamy, S., Kairouz, P., Chen, M., Mathews, R., & y Arcas, B.A. (2019). Generative models for effective ml on private, decentralized datasets. In *International conference on learning representations*.

Bagui, S., & Nguyen, L. T. (2015). Database sharding: To provide fault tolerance and scalability of big data on the cloud. *International Journal of Cloud Applications and Computing (IJCAC), 5*(2), 36–52.

Bakopoulou, E., Tillman, B., & Markopoulou, A. (2019). A federated learning approach for mobile packet classification. arXiv:1907.13113

Baron, M. (1999). Convergence rates of change-point estimators and tail probabilities of the first-passage-time process. *Canadian Journal of Statistics, 27*(1), 183–197.

Brisimi, T. S., Chen, R., Mela, T., Olshevsky, A., Paschalidis, I. C., & Shi, W. (2018). Federated learning of predictive models from federated electronic health records. *International Journal of Medical Informatics, 112*, 59–67.

Canetti, R., Feige, U., Goldreich, O., & Naor, M. (1996). Adaptively secure multi-party computation. In *Proceedings of the twenty-eighth annual ACM symposium on theory of computing* (pp. 639–648).

Casado, F. E., Rodríguez, G., Iglesias, R., Regueiro, C. V., Barro, S., & Canedo-Rodríguez, A. (2020). Walking recognition in mobile devices. *Sensors 20*(4).

Custers, B., Sears, A. M., Dechesne, F., Georgieva, I., Tani, T., & van der Hof, S. (2019). *EU personal data protection in policy and practice*. Springer.

Czyz, J., Kittler, J., & Vandendorpe, L. (2004). Multiple classifier combination for face-based identity verification. *Pattern Recognition, 37*(7), 1459–1469.

Dietterich, T. G., et al. (2002). Ensemble learning. *The Handbook of Brain Theory and Neural Networks, 2*, 110–125.

Ditzler, G., & Polikar, R. (2012). Incremental learning of concept drift from streaming imbalanced data. *IEEE Transactions on Knowledge and Data Engineering, 25*(10), 2283–2301.

Dwork, C. (2008). Differential privacy: A survey of results. In *International conference on theory and applications of models of computation* (pp. 1–19). Springer.

Fazi, M. B. (2021). Beyond human: Deep learning, explainability and representation. *Theory, Culture and Society, 38*(7–8), 55–77.

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

Guha, N., Talwalkar, A., & Smith, V. (2019). One-shot federated learning. arXiv:1902.11175

Hamer, J., Mohri, M., & Suresh, A. T. (2020). Fedboost: A communication-efficient algorithm for federated learning. In *International conference on machine learning, PMLR* (pp. 3973–3983).

Haque, A., Khan, L., & Baron, M. (2016). Sand: Semi-supervised adaptive novel class detection and classification over data stream. In *Thirtieth AAAI conference on artificial intelligence* (pp. 1652–1658).

Hard, A., Rao, K., Mathews, R., Ramaswamy, S., Beaufays, F., Augenstein, S., Eichner, H., Kiddon, C., & Ramage, D. (2018). Federated learning for mobile keyboard prediction. arXiv:1811.03604

Jain, A. K., & Chandrasekaran, B. (1982). 39 dimensionality and sample size considerations in pattern recognition practice. *Handbook of Statistics, 2*, 835–855.

Kittler, J., Hatef, M., Duin, R. P., & Matas, J. (1998). On combining classifiers. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 20*(3), 226–239.

Kuhn, M., Wing, J., Weston, S., Williams, A., Keefer, C., Engelhardt, A., Cooper, T., Mayer, Z., Kenkel, B., & Team, R. C., et al. (2020). Package 'caret'. *The R Journal*.

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

Li, Q., Wen, Z., Wu, Z., Hu, S., Wang, N., Li, Y., Liu, X., & He, B. (2019). A survey on federated learning systems: Vision, hype and reality for data privacy and protection. arXiv:1907.09693

Li, S., Da Xu, L., & Zhao, S. (2018). 5G internet of things: A survey. *Journal of Industrial Information Integration, 10*, 1–9.

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

Lim, W. Y. B., Luong, N. C., Hoang, D. T., Jiao, Y., Liang, Y. C., Yang, Q., Niyato, D., & Miao, C. (2019). Federated learning in mobile edge networks: A comprehensive survey. arXiv:1909.11875

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

Liu, B., Wang, L., & Liu, M. (2019). Lifelong federated reinforcement learning: A learning architecture for navigation in cloud robotic systems. *IEEE Robotics and Automation Letters, 4*(4), 4555–4562.

Lu, J., Liu, A., Dong, F., Gu, F., Gama, J., & Zhang, G. (2018). Learning under concept drift: A review. *IEEE Transactions on Knowledge and Data Engineering*.

Ludwig, H., Baracaldo, N., Thomas, G., Zhou, Y., Anwar, A., Rajamoni, S., Ong, Y., Radhakrishnan, J., Verma, A., & Sinn, M., et al. (2020). Ibm federated learning: an enterprise framework white paper v0.1. arXiv:2007.10987

McMahan, H. B., Moore, E., Ramage, D., & y Arcas, B. A. (2016). Federated learning of deep networks using model averaging. arXiv:1602.05629v1

Nasr, M., Shokri, R., & Houmansadr, A. (2019). 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)* (pp. 739–753). IEEE.

O'Mahony, N., Campbell, S., Carvalho, A., Harapanahalli, S., Hernandez, G. V., Krpalkova, L., Riordan, D., & Walsh, J. (2019). Deep learning vs. traditional computer vision. In *Science and information conference* (pp. 128–144). Springer.

Parisi, G. I., Kemker, R., Part, J. L., Kanan, C., & Wermter, S. (2019). Continual lifelong learning with neural networks: A review. *Neural Networks*.

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

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

Raudys, S. J., & Jain, A. K. (1991). Small sample size effects in statistical pattern recognition: Recommendations for practitioners. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 13*(3), 252–264.

Rey, V., Sánchez, P. M. S., Celdrán, A. H., & Bovet, G. (2022). Federated learning for malware detection in IoT devices. *Computer Networks,* 108693.

Rodríguez, G., Casado, F. E., Iglesias, R., Regueiro, C. V., & Nieto, A. (2018). Robust step counting for inertial navigation with mobile phones. *Sensors 18*(9).

Sagi, O., & Rokach, L. (2018). Ensemble learning: A survey. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery, 8*(4), e1249.

Saputra, Y. M., Hoang, D. T., Nguyen, D. N., Dutkiewicz, E., Mueck, M. D., & Srikanteswara, S. (2019). Energy demand prediction with federated learning for electric vehicle networks. In *2019 IEEE global communications conference (GLOBECOM)* (pp. 1–6), IEEE.

Sattler, F., Wiedemann, S., Müller, K. R., & Samek, W. (2019). Robust and communication-efficient federated learning from non-iid data. *IEEE Transactions on Neural Networks and Learning Systems, 31*(9), 3400–3413.

Schwartz, R., Dodge, J., Smith, N. A., & Etzioni, O. (2020). Green AI. *Communications of the ACM, 63*(12), 54–63.

Shoaib, M., Bosch, S., Incel, O. D., Scholten, H., & Havinga, P. J. (2014). Fusion of smartphone motion sensors for physical activity recognition. *Sensors, 14*(6), 10146–10176.

Soliman, A., Girdzijauskas, S., Bouguelia, M.R., Pashami, S., & Nowaczyk, S. (2020). Decentralized and adaptive k-means clustering for non-iid data using hyperloglog counters. In *Pacific-Asia conference on knowledge discovery and data mining* (pp. 343–355). Springer.

Strubell, E., Ganesh, A., & McCallum, A. (2019). Energy and policy considerations for deep learning in NLP. In *Proceedings of the 57th annual meeting of the association for computational linguistics* (pp. 3645–3650).

Tsoumakas, G., Angelis, L., & Vlahavas, I. (2004). Clustering classifiers for knowledge discovery from physically distributed databases. *Data and Knowledge Engineering, 49*(3), 223–242.

Tumer, K., & Ghosh, J. (1996). Error correlation and error reduction in ensemble classifiers. *Connection Science, 8*(3–4), 385–404.

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

Wang, Z., Song, M., Zhang, Z., Song, Y., Wang, Q., & Qi, H. (2019b). Beyond inferring class representatives: User-level privacy leakage from federated learning. In *IEEE INFOCOM 2019-IEEE conference on computer communications* (pp. 2512–2520). IEEE.

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

Widmer, G., & Kubat, M. (1996). Learning in the presence of concept drift and hidden contexts. *Machine Learning, 23*(1), 69–101.

Wolpert, D. H. (1992). Stacked generalization. *Neural Networks, 5*(2), 241–259.

Yoon, J., Jeong, W., Lee, G., Yang, E., & Hwang, S. J. (2021). Federated continual learning with weighted inter-client transfer. In *International conference on machine learning, PMLR* (pp. 12073–12086).

Zhou, Z. H. (2009). Ensemble learning. *Encyclopedia of Biometrics, 1*, 270–273.

Authors and Affiliations

Fernando E. Casado[1] ⓘ **· Dylan Lema[1] · Roberto Iglesias[1] · Carlos V. Regueiro[2] · Senén Barro[1]**

Dylan Lema
dylan.lema@usc.es

Roberto Iglesias
roberto.iglesias.rodriguez@usc.es

Carlos V. Regueiro
carlos.vazquez.regueiro@udc.es

Senén Barro
senen.barro@usc.es

[1] CiTIUS (Centro Singular de Investigación en Tecnoloxías Intelixentes), Universidade de Santiago de Compostela, 15782 Santiago de Compostela, Spain

[2] CITIC, Computer Architecture Group, Universidade da Coruña, 15071 A Coruña, Spain