Ensemble Methods for Collective Intelligence: Combining Ubiquitous ML Models in IoT

Bharath Sudharsan[§*], Piyush Yadav[*], Duc-Duy Nguyen[*], Jefkine Kafunah[*], and John G. Breslin[*]
[§]ARM Machine Learning Infrastructure, Galway, Ireland
[*]Data Science Institute, NUI Galway, Ireland
{bharath.sudharsan, piyush.yadav, ducduy.nguyen, jefkine.kafunah, john.breslin}@insight-centre.org

***Abstract*—The concept of *ML model aggregation rather than data aggregation* has gained much attention as it boosts prediction performance while maintaining stability and preserving privacy. In a non-ideal scenario, there are chances for a base model trained on a single device to make independent but complementary errors. To handle such cases, in this paper, we implement and release the code of 8 robust ML model combining methods that achieves reliable prediction results by combining numerous base models (trained on many devices) to form a central model that effectively limits errors, built-in randomness and uncertainties. We extensively test the model combining performance by performing 15 heterogeneous devices and 3 datasets based experiments that exemplifies how a complicated collective intelligence can be derived from numerous elementary intelligence learned by distributed, ubiquitous IoT devices.**

***Index Terms*—IoT Devices, Data Mining, Privacy Preservation, Distributed Training, Ensemble Learning.**

I. Introduction

In many IoT scenarios, the training dataset sources are decentralized over different devices such as sensors and mobile phones. This poses challenges to train robust ML models due to data accessibility and privacy concerns (such as GDPR). In such cases, learning algorithms are deployed across distributed IoT devices that have access to the same type of rich training data. Then, without voiding the privacy protection regulations, problem-solving base machine learning (ML) models are created at the IoT device level and trained without storing the live data. Later these multiple base ML models are transmitted to a central high-end device (such as cloud) for model combining. This concept of collecting intelligence (trained base models) from billions of deployed ubiquitous IoT devices, rather than raw data, is the *future of ML and IoT* [1].

ML Model combining, often regarded as a key sub-field of ensemble learning, started as early as 1977, has been widely used in both academic research and industry applications. Also, in many ML competitions, ensemble learning is the key to winning solutions because it trains multiple base ML models then combines them to solve a problem [2]. Since this approach is in contrast to the conventional method of constructing one model to solve the problem, it is *committee based learning system* [3]. The term ensemble means a group of base models generated by a dataset using any base model training algorithms such as k-NN, Decision Tree, SVM. When an ensemble contains the same type of base models, it is a *homogeneous ensemble*. When multiple types of base models are grouped, it is a *heterogeneous ensemble* [4]. This concept of base models combining is appealingly termed as *distributed ensemble learning* when base models are trained remotely across various systems (IoT devices in our scenario), then combined in a central location. The paper contributions can be summarized as follows:

- The studies from centralized learning, split learning, distributed ensemble learning have extensively investigated combining models trained on devices like smartphones, Raspberry Pis, Jetson Nanos, etc. Such devices have sufficient resources to train base models (or ensembles) using standard training algorithms from Python Scikit-learn or light version ML frameworks like TensorFlow Lite. In contrast, we aim to achieve collective intelligence using MCUs, since billions of deployed IoT devices like HVAC controllers, smart meters, video doorbells have resource-constrained MCU-based hardware with only a few MB of memory.
- From the available multitudinous number of studies, we choose, implement, and provide 8 robust ML model combining methods that are compatible with a wide range of datasets (varying feature dimensions and classes) and IoT devices (heterogeneous hardware specifications). We open-source the implementation[1], utilizing which researchers and engineers can start practicing distributed ensemble learning by combining ML base models trained on ubiquitous IoT devices.

II. Implementing ML Model Combining Methods

Usecase- Providing sensitive medical data for research. The data required for most research are sensitive in nature, as it revolves around a private individual. So, GDPR restricts sending such sensitive yet valuable medical data (from hospitals, imaging centers) to research institutes. As shown in Fig. 1, when the resource-constrained medical devices like insulin-delivery devices, BP apparatus are equipped with IoT hardware-friendly training algorithms [5, 6], they can perform onboard training of base models, even without depending on the hospital's local servers. After training, the base models from similar devices can be extracted, combined, and sent to research labs with improved data privacy preservation. For example, the 2 base models $\mathrm{M7}_1$, $\mathrm{M7}_2$ (see Fig. 1) trained on ECG monitors using the vital data of patients can be combined centrally, then shared for research.

[1]Code: https://github.com/bharathsudharsan/ML-Model-Combining

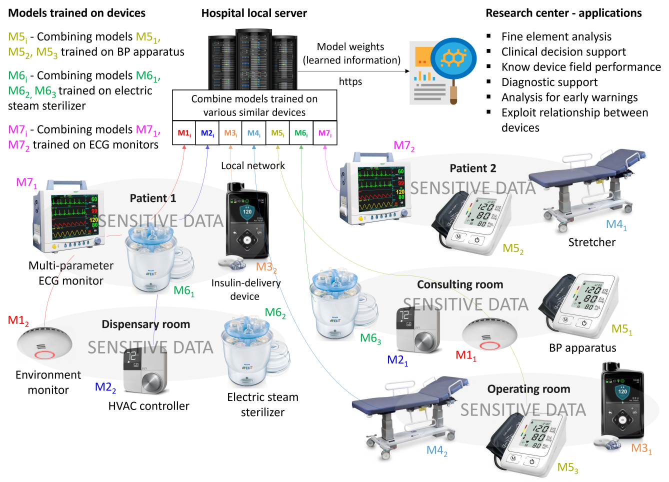
Models trained on devices
Hospital local server
Research center - applications
$M5_i$ - Combining models $M5_1$, $M5_2$, $M5_3$ trained on BP apparatus
$M6_i$ - Combining models $M6_1$, $M6_2$, $M6_3$ trained on electric steam sterilizer
$M7_i$ - Combining models $M7_1$, $M7_2$ trained on ECG monitors
Model weights (learned information)
https
Combine models trained on various similar devices
$M1_i$ $M2_i$ $M3_i$ $M4_i$ $M5_i$ $M6_i$ $M7_i$
Fine element analysis
Clinical decision support
Know device field performance
Diagnostic support
Analysis for early warnings
Exploit relationship between devices
Local network
Patient 1
SENSITIVE DATA
Multi-parameter ECG monitor
Insulin-delivery device
Patient 2
Stretcher
Consulting room
BP apparatus
Dispensary room
Environment monitor
HVAC controller
Electric steam sterilizer
Operating room
$M7_1$ $M7_2$ $M6_1$ $M6_2$ $M6_3$ $M3_1$ $M3_2$ $M1_1$ $M1_2$ $M2_1$ $M2_2$ $M4_1$ $M4_2$ $M5_1$ $M5_2$ $M5_3$

Fig. 1. Collective learning among medical IoT devices: A use case of privacy-preserving sensitive medical data collection.

A. Combine by Simple Averaging

Simple Averaging is the most fundamental method, where the combined output is obtained by directly averaging the outputs of individual base classifiers. When given a set of T individual base classifiers $\{h1, ..., h_T\}$ and the output of h_i for the input x is $h_i(x) \in \mathbb{R}$, the task becomes to combine all the h_i to attain the final prediction on the real-valued variable.

Due to its effectiveness and simplicity, this method is represented as the first choice for many real-world applications. However, the error reduction method of simple averaging is derived based on the assumption that the errors of the individual base classifiers are uncorrelated. While in our case of strategically combining classifiers, the errors are typically highly correlated since all the base classifiers are trained to solve the same problem. Therefore, the error reduction method of simple averaging is generally hard to achieve - because it suggests that *ensemble error is smaller by a factor of T (value of total base classifiers) than the averaged error of the individual classifiers*.

B. Combine by Weighted Averaging

Here, we obtain the combined output by averaging the outputs of individual learners with different weights implying different importance [3]. The above simple averaging, which can be regarded as taking equal weights for all individual learners, can be viewed as a special case of weighted averaging. Similarly, other combining methods such as voting are also special cases or variants of weighted averaging.

The data of real tasks are usually noisy and insufficient, resulting in producing estimated weights that are often unreliable. Particularly when combining a large number of classifiers, more weights need to be learned, which can easily lead to overfitting. Simple averaging does not have to learn any weights, so it suffers little from overfitting. In general, we recommend using the above simple averaging for combining classifiers with similar performances, whereas if the individual classifiers exhibit nonidentical strength, weighted averaging with unequal weights can show better performance.

C. Combine by Voting

The concept of voting is the fundamental and most popular combining method for nominal outputs. Since we are dealing with combining classifiers, we use a classification example to explain this concept. When we have T number of individual classifiers $\{h_1, ..., h_T\}$, here, our task is to combine all h_i to predict the class label from a set of l possible class labels. When classifiers are trained to solve binary problems, the class labels are $\{c_0, c_1\}$. Here, for an input x, the outputs of the classifier h_i are given an l dimensional label vector $(h_i^1(x), \ldots, h_i^l(x))^\top$, where $h_i^j(x)$ is the output of h_i, for the class label c_j. The $h_i^j(x)$ takes different types of values according to the information provided by the individual classifiers, e.g., for a *Crisp label* (our case): $h_i^j(x) \in \{0, 1\}$, which takes value 1 if h_i predicts c_j as the class label and 0 otherwise. Or can take *Class probability*: $h_i^j(x) \in [0, 1]$, which can be regarded as an estimate of the posterior probability $P(c_j|x)$.

1) Majority Voting: This is the most popular voting method. Here, each base classifiers votes for one class label, the final output class label is the one that receives more than half of the votes. If none of the class labels receives more than half of the votes, a rejection option will be given, and the combined classifier makes no prediction. Hence the output class label of the combined central classifier becomes,

$$H(\boldsymbol{x}) = \begin{cases} c_j & \text{if } \sum_{i=1}^{T} h_i^j(x) > \frac{1}{2} \sum_{k=1}^{l} \sum_{i=1}^{T} h_i^k(x), \\ \text{reject} & \text{otherwise} \end{cases} \tag{1}$$

This suits well when the individual classifiers are statistically independent, but since all the base classifiers are trained to solve the same problem (i.e., the same dataset is used), they are highly correlated. Therefore, it is impractical to expect the majority voting accuracy to converge to one when the base classifier count is high.

2) Weighted Majority Voting: In reality, each base classifier is reliable in different areas of the problem/dataset, and most times, the individual classifiers are with unequal performance. Hence, it is highly inefficient to treat the labels from different classifiers with the same weight. So in the *Weighted Majority Voting* (WMV) method, which is the natural extension of above majority voting, we give more power to the stronger classifiers in voting. Hence, as shown in the Eqn (2), the weighted majority voting rule is an extension of Eqn (1) with a weight $w_i \geq 0$ assigning to the classifier h_i.

$$H(x) = c_{\arg\max} \sum_{i=1}^{T} w_i h_i^j(x) \tag{2}$$

Here, the accuracy of the final aggregated label depends much on the accuracy of the estimated weights. During the distributed model training, some of the involved IoT devices might be from a vendor who might not have added proper hardware filters (cost-cutting). So there exist noises that impair the local training data quality. On such devices/data, the training algorithm produces base learners that show fluctuations in prediction results. In such cases, WMV is more suitable than other voting methods, as it can make the models from devices with a higher competence level count more for the final decision/prediction.

D. Combine by Maximization and Median

Other combining methods, in addition to thus far explained ones, are the *algebraic methods* termed maximization and median, that we implement here.

The class probability output from individual base classifiers trained on different IoT devices can be considered as an estimate of the posterior probabilities. Hence, it is straightforward to derive combining rules under the probabilistic framework. The $h_i^j(x)$ is the class probability of c_j output from h_i as h_i^j. According to Bayesian decision theory, given T classifiers, the input x from the dataset should be assigned to the class c_j, which maximizes the posterior probability $P(c_j|h_1^j,...,h_T^j)$. Since h_i^j is the probability output, we have $P(c_j|h_i^j) = h_i^j$. Thus, if all classes are with equal priority, we get the product rule for combining as

$$H^j(x) = \prod_{i=1}^{T} h_i^j(x)$$

Similarly to above, maximum/median rules were derived using which here, we choose the *maximum* and *median* of the individual outputs as the combined output. For example, the median rule generates the combined output according to $H^j(x) = \mathrm{med}(h_i^j(x))$, where $\mathrm{med}(.)$ denotes the median statistic. In short, maximization method is carried out by taking the maximum scores, and in *Combine by Median* method, the median value across all scores/prediction results are taken.

E. Combine by Dynamic Classifier Selection (DCS)

In contrast to classic learning methods, which select the *best* individual learner and discard other, DCS keeps all the individual learners (thus regarded as a *soft combining method*) [4]. In contrast to typical ensemble methods, which combine individual learners to make predictions, DCS makes predictions by using one individual learner.

In this work, a DCS method called DCS-LA is implemented and used on models trained by distributed IoT devices. Here, each base classifier's accuracy in *local regions* of feature space is estimated, i.e., the best classifier for each partition of the dataset is determined. For classification results (inference), we use the output produced by the most locally accurate classifier as the final decision. In DCS-LA, we estimate the local accuracy w.rt. some output class. In other words, when considering a classifier that assigns an input data sample to class c_i, we can determine the percentage of the local training samples assigned to class c_i by this classifier that has been correctly labeled.

F. Combine by Dynamic Ensemble Selection (DES)

In most use cases like handwritten pattern recognition, multiple classifiers are used to improve the recognition rates [2]. To optimize such a multiple classifier system, a group of classifiers, known as an Ensemble of Classifiers (EoC), are selected from a pool of classifiers. To achieve high performance, dynamic selection methods (select different classifiers for different data patterns) need to be used over static selection methods (select an EoC for all data patterns). To combine models from various IoT devices, as introduced, we implement a scheme that dynamically selects an ensemble or every test data point - instead of dynamic classifier selection, dynamic ensemble selection is performed here. This method is implemented as it can perform better than static ensemble selection methods, and also, using an ensemble of classifiers provides more stability than a single classifier.

TABLE I
CHOSEN DEVICES ON WHICH DISTRIBUTED TRAINING IS PERFORMED.

	Board#: Name	**Specs: Processor flash, SRAM, clock (MHz)**
	B1: nRF52840 Feather	Cortex-M4, 1MB, 256KB, 64
	B2: STM32f10 Blue Pill	Cortex-M0, 128kB, 20KB, 72
	B3: Adafruit HUZZAH32	Xtensa LX6, 4MB, 520KB, 240
	B4: Raspberry Pi Pico	Cortex-M0+, 16MB, 264KB, 133
MCUs	B5: ATSAMD21 Metro	Cortex-M0+, 256kB, 32KB, 48
	B6: Arduino Nano 33	Cortex-M4, 1MB, 256KB, 64
	B7: Teensy 4.0	Cortex-M7, 2MB, 1MB, 600
	B8: STM32 Nucleo H7	Cortex-M7, 2MB, 1MB, 480
	B9: Feather M4 Express	Cortex-M4, 2MB, 192KB, 120
	B10: Arduino Portenta	Cortex-M7+M4, 2MB, 1MB, 480
	CPU#: Name	**Basic specs**
	C1: W10 Laptop	Intel Core i7 @1.9GHz
	C2: NVIDIA Jetson Nano	128-core GPU @ 1.4GHz
CPUs	C3: W10 Laptop	Intel Core i5 @1.6GHz
	C4: Ubuntu Laptop	Intel Core i7 @2.4GHz
	C5: Raspberry Pi 4	Cortex-A72 @1.5GHz

G. Combine by Meta Ensembling (Stacking)

The implementation of this method combines information from multiple base classifier models to generate a new model. Thus generated 2^{nd} level model is known as a stacked model that often outperforms base models - this is due to its smoothing nature and its ability to identify each base classifier where it performs best and discredit it in the areas where it performs poorly [7]. The implementation performs stacking in two phases. First, the base classifiers are trained across various devices using a local dataset. Then in a central location, the output of each model (i.e., meta-features) is collected to build a new dataset. So, the stacked model can discern where each model performs well and where each model performs poorly. Second, this new dataset is used with any meta-learning algorithm to provide the final classification result. Choosing the right stacker and the features is *more of an art than science*.

III. EXPERIMENTS: DISTRIBUTED TRAIN THEN COMBINE

Distributed, ubiquitous IoT Devices in the real world have heterogeneous hardware specifications. To replicate this scenario, the devices chosen to carry out the distributed training, given in Table I, contains 10 resource-constrained MCU boards (B1-B10) and 5 CPU devices (C1-C5). The training process on all the 15 devices are carried out using the resource-friendly classifier training algorithm from ML-MCU [6], and 3 datasets that are Banknote Authentication [8] (4 features), Haberman's

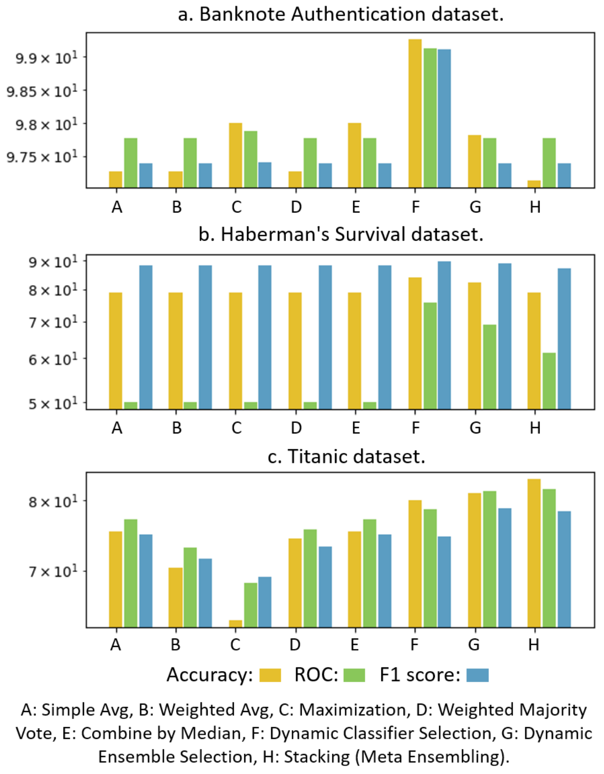

Fig. 2. Performance: Combining ML models trained on 15 devices.

Survival [9] (3 features), and the popular Titanic shipwreck [10] (6 features).

Initially, for the Banknote dataset, upon all devices completing the training, 15 base models are obtained (first set). Then, each of the 8 ML model combining methods from Section II are one by one applied on this first set of models, producing 8 central models (one central model as an output of each combining method). A similar procedure was followed for the remaining datasets, producing the second and third set of models, followed by model combining. At this stage, there are 8 central models for each dataset, whose performance was evaluated in terms of Accuracy, ROC, and F1 score (F1) metrics and reported in Fig. 2., and analyzed below.

A. *Performance Analysis of Combined Central Models*

For the Banknote dataset (see Fig. 2. a), the highest performance is shown by the Dynamic Classifier Selection (DCS-LA) method. Followed by Maximization, then the Median combination method, where both show the same accuracy and slightly different ROC and F1. The Simple Averaging, Weighted Averaging, and the Weighted Majority Vote (WMV) methods achieve similar performance. The combine by Stacking is the least performing, followed by Dynamic Ensemble Selection (DES) method. For Haberman's dataset (see Fig. 2. b) again, DCS-LA showed the top performance. The DES and Stacking methods that produced a low performance for the previous dataset are the second and third best-performing methods. The other algebraic, averaging, and voting methods perform almost the same, achieving good accuracy and F1 but low ROC. For the Titanic dataset (see Fig. 2. c), Stacking shows the highest accuracy, but DES achieved slightly higher ROC and F1 so, DES is the overall top-performing method. Unlike in previous datasets, here, the algebraic (combine by Maximization and Median), Averaging, and Voting methods show varying performance. From the algebraic methods, the combine by Median performed better. From averaging methods, Simple Averaging performed better.

The following observation was made during experimentation: (i) The computational cost for creating an ensemble is not much higher than training a single base model. It is because multiple versions of the base model need to be generated during parameter tuning. Also, the computational cost for combining multiple IoT devices trained base models was small due to the simplicity of the presented combination strategies. (ii) To construct a good ensemble, it is recommended to create base models *as accurate and as diverse as possible*. (iii) Creating a learning algorithm that is consistently better than others is a hopeless daydream. i.e., from Fig. 2, Stacking shows top performance for the Titanic dataset and least in the Banknote dataset.

IV. Conclusion

With strict privacy regulations, the historic datasets building process is rapidly transforming into historic intelligence building - achieved by distributed learning on the IoT devices, then central model combining. In this work, 8 robust ML model combining methods were implemented and extensively tested, whose open-sourced implementation, we believe to provide the basis for a broader spectrum of decentralized and collective learning applications.

Acknowledgement

This publication has emanated from research supported in part by a research grant from Science Foundation Ireland (SFI) under Grant Number SFI/16/RC/3918 (Confirm) and also by a research grant from SFI under Grant Number SFI/12/RC/2289_P2 (Insight), with both grants co-funded by the European Regional Development Fund.

References

[1] B. Sudharsan, P. Patel, J. Breslin, M. I. Ali, K. Mitra, S. Dustdar, O. Rana, P. P. Jayaraman, and R. Ranjan, "Toward distributed, global, deep learning using iot devices," *IEEE Internet Computing*, 2021.

[2] A. H. Ko, R. Sabourin, and A. S. Britto Jr, "From dynamic classifier selection to dynamic ensemble selection," *Pattern recognition*, 2008.

[3] Z.-H. Zhou, "Ensemble methods: Foundations and algorithms," *Chapman Hall/CRC*, 2012.

[4] K. Woods, W. P. Kegelmeyer, and K. Bowyer, "Combination of multiple classifiers using local accuracy estimates," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 19, no. 4, pp. 405–410, 1997.

[5] B. Sudharsan, J. G. Breslin, and M. I. Ali, "Edge2train: A framework to train machine learning models (svms) on resource-constrained iot edge devices," in *International Conference on the Internet of Things*, 2020.

[6] B. Sudharsan, J. G. Breslin, and M. I. Ali, "Ml-mcu: A framework to train ml classifiers on mcu-based iot edge devices," *IEEE IoTJ*, 2021.

[7] "Stacking via meta ensembling: https://www.gormanalysis.com/blog/guide-to-model-stacking-i-e-meta-ensembling/," 2016.

[8] "Uci machine learning repository: Data set," archive.ics.uci.edu. [Online]. Available: https://archive.ics.uci.edu/ml/datasets/banknote

[9] "Uci machine learning repository: Data set," archive.ics.uci.edu. [Online]. Available: https://archive.ics.uci.edu/ml/datasets/Haberman

[10] "Titanic: Machine learning from disaster," Kaggle.com, 2012. [Online]. Available: https://www.kaggle.com/c/titanic/data