Appliance Recognition on Internet-of-Things Devices

Gérôme Bovet*‡ and Antonio Ridi†‡ and Jean Hennebert‡
*LTCI
Institut Mines-Telecom, Telecom ParisTech, 46 Rue Barrault, 75013 Paris, France
Email: gerome.bovet@telecom-paristech.fr
†DIUF
University of Fribourg, Bd de Pérolles 90, 1700 Fribourg, Switzerland
Email: jean.hennebert@unifr.ch
‡iCoSys
University of Applied Sciences Western Switzerland, Bd de Pérolles 80, 1700 Fribourg, Switzerland

***Abstract*—Machine Learning (ML) approaches are increasingly used to model data coming from sensor networks. Typical ML implementations are cpu intensive and are often running server-side. However, IoT devices provide increasing cpu capabilities and some classes of ML algorithms are compatible with distribution and downward scalability. In this demonstration we explore the possibility of distributing ML tasks to IoT devices in the sensor network. We demonstrate a concrete scenario of appliance recognition where a smart plug provides electrical measures that are distributed to WiFi nodes running the ML algorithms. Each node estimates class-conditional probabilities that are then merged for recognizing the appliance category. Finally, our architectures relies on Web technologies for complying with Web-of-Things paradigms.**

I. Introduction

Nowadays, many objects are following the Internet-of-Things (IoT) paradigm by relying on well known IP technologies. In spite of increasing capabilities, those devices are still performing simple tasks such as sense and react. We also observe the emergence of data-driven ML algorithms that are able to leverage on the large amount of sensor data. ML algorithms are usually executed on powerful computers often residing server-side. In this demonstration, we explore the feasibility of deporting the execution of those algorithms directly on things. For this purpose, we use several OpenPicus FlyportPRO WiFi [1] modules which will act as distributed computing power. Our implementation is also relying on Web technologies as application layer according to Web-of-Things (WoT) paradigms [2].

II. Machine Learning Approach

The experimental setup demonstrates the feasibility and effectiveness of our approach by its deployment in the field of the electrical appliance recognition. The electrical consumption is measured with smart plugs placed between the appliance plug and the wall socket. Different features can be measured by the smart plug, as the active power, reactive power, RMS current, voltage, phase and frequency. The aim of the electrical appliance recognition task consists in analysing a sequence of electrical appliance measures, called *appliance signatures*, and provide information about the appliance generating the signal. Potentially, several information can be inferred, as the brand and model of the appliance, its category (i.e. coffee machine) or its actual state (i.e. stand-by). In this work, we demonstrate the recognition of the appliance category and state using an implementation inspired from [4].

For training the models we use the ACS-F2 database [3], containing 450 electrical signatures recorded from 225 appliances of different brands and / or models. A signature consists of one hour recording with a sampling frequency of 10^{-1} Hz. The appliances are uniformly spread into 15 categories.

The signatures in the ACS-F2 database are represented in a six dimensional space as described above. Given the small quantity of memory available on the OpenPicus, we decide to reduce as much as possible the feature space, removing the non relevant frequency and voltage features, as well as the phase angle which is redundant with the active and reactive power information [4].

We selected Hidden Markov Models (HMMs) to represent the state-based nature of the signals. In our setting, we assume that the HMMs are trained offline due to the large quantity of data requested by ML training procedures. The training of the HMMs is performed here using a classical Expectation Maximisation procedure that iteratively estimates the parameters of the HMMs, i.e. the transition probabilities between states and the emission probability estimators, in our case Gaussian Mixtures Models. We consider here two states models: on and off (or stand-by). After training, the models parameters are transferred to the OpenPicus that are then in charge of processing real-time data. The choice of generative models such as HMMs is also motivated by our need to spread the computation load in the network of OpenPicus. This is naturally done considering that we can train one model M_i per category and spread the computation of class conditional probabilities $p(x|M_i)$ (likelihoods) on each nodes. The decision about the category label is finally taken using the simple Bayes rule, i.e. electing the highest likelihood if all categories have equal priors. In a previous work the system reported increasing accuracy using larger number of Gaussian per state, typically up to 30 [3]. Given the memory restriction of the devices we are here limiting the number of Gaussians to 8, trading off slightly the accuracy rate.

III. General Architecture

As our architecture relies on the WoT paradigm, every device embeds a RESTful API accessible through CoAP (i.e a lightweight version of HTTP) [5]. In order to sense non-physical measures like an appliance category, we introduce the concept of *virtual sensors*. Those type of sensors are

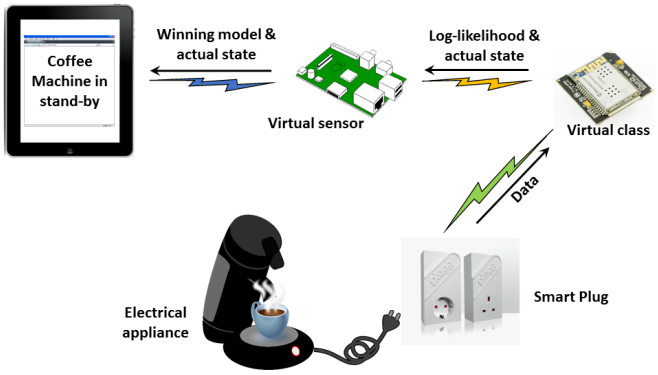

Fig. 1. Data exchange between the entities.

associated to complex data-driven process composed of several models called *virtual classes*. The Virtual sensor exploits the computational power of nearly located smart things and deploys models on them, forming a set of virtual classes. Virtual classes are agents able to compute class conditional probabilities (likelihoods) as explained above. They are periodically executing the algorithm using the data coming from the sensors. From a client's point of view, a virtual sensor acts like any other type of sensing device offering a Web resource representing the actual state. With a single GET request, a client can retrieve the actual value inferred from the Virtual classes.

Offline training algorithms will deploy the runtime algorithms by creating new virtual sensors resources. This step is achieved by providing a JSON document containing all the parameters of the machine learning models. The newly created virtual sensor then handles all the distribution of the models on the OpenPicus modules, creating virtual classes. Due to the memory limitation, parameters are exchanged in float binary format (IEEE 754) in order to avoid the JSON verbosity. Virtual classes will further register at physical sensors to be notified with new values.

Our implementation offers two access modes for clients (selectable when creating the virtual sensor). In the *end-to-end mode*, clients are requesting the virtual sensor for the actual appliance category. In order to determine the appliance category, each virtual class will be requested sequentially for its current likelihood. The virtual sensor then compares the likelihood values and, according to the Bayes rule, the model having the highest likelihood is designated as winner (assuming equal priors). The winning appliance category and its actual state are then returned to the client. The *sync-based mode* is fully event-driven and is especially tailored for real-time systems. Virtual classes will notify the Virtual sensor each time a new likelihood is computed. The decision making is performed as soon as a likelihood changes and is cached on the virtual client. This allows to avoid the sequential requesting of virtual classes and improves the scalability as well as the round-trip time. Clients can optionally observe a virtual sensor that will send notifications as soon as the appliance category changes.

IV. Demonstration

Our demonstration is composed of a smart plug *Plogg*, a set of OpenPicus FlyportPRO WiFi nodes, a Raspberry Pi and a client tablet device. The OpenPicus nodes execute a model (virtual class resource), corresponding to a specific appliance category. The Raspberry Pi acts as virtual sensor for making the decision about the category of the appliance plugged in the smart plug. The virtual classes will regularly be notified about new values coming from the smart plug. We use a mobile device as client to feedback the user about information on the identified appliance. A web page using Sync-based mode is used and notified by the Virtual sensor in real-time. Figure1 illustrates the different entities and their data exchange. Off-line experiments of the demonstration settings show correct identification rates of about 90%.

V. Conclusion

We have presented an architecture for executing the run-time part of a machine-learning process on smart things. An experimental setup demonstrates the feasibility of this architecture able to execute higher-level tasks on a network of IoT things. We successfully deployed complex trained models performing a classification task for appliance recognition. The extensive use of Web technologies among with the concepts of virtual sensor and virtual class allows a seamless integration in the world of IoT and WoT. Future works will include the recognition of user activities in houses, as well as the reconsidering the training part according to the WoT precepts.

VI. Acknowledgment

This work is supported by the grant Smart Living GreenMod from the Hasler Foundation in Switzerland and by the HES-SO.

References

[1] Openpicus flyportpro wifi. http://space.openpicus.com/u/ftp/datasheet/datasheet_flyportpro_wifi.pdf.

[2] D. Guinard. *A Web of Things Application Architecture - Integrating the Real World into the Web*. PhD thesis, ETHZ, 2011.

[3] A. Ridi, C. Gisler, and J. Hennebert. ACS-F2 - A New Database of Appliance Consumption Analysis. In *Proceedings of the International Conference on Soft Computing and Pattern Recognition (SocPar 2014), to appear*, 2014.

[4] A. Ridi, C. Gisler, and J. Hennebert. Appliance and State Recognition using Hidden Markov Models. In *Proceedings of the International Conference on Data Science and Advanced Analytics (DSAA 2014), to appear*, 2014.

[5] Z. Shelby, K. Hartke, and C. Bormann. Constrained application protocol (coap). draft-ietf-core-coap, 2014.