Virtual Things for Machine Learning Applications

Gérôme Bovet (1)(3)
(1) Laboratory for Communication and Processing of Information
Telecom ParisTech
Paris, France
gerome.bovet@telecom-paristech.fr

Antonio Ridi (2)(3)
(2) Department of Informatics
University of Fribourg
Fribourg, Switzerland
antonio.ridi@unifr.ch

Jean Hennebert (3)(2)
(3) Institute of Complex Systems
University of Applied Sciences Western Switzerland
Fribourg, Switzerland
jean.hennebert@hefr.ch

ABSTRACT
Internet-of-Things (IoT) devices, especially sensors are producing large quantities of data that can be used for gathering knowledge. In this field, machine learning technologies are increasingly used to build versatile data-driven models. In this paper, we present a novel architecture able to execute machine learning algorithms within the sensor network, presenting advantages in terms of privacy and data transfer efficiency. We first argument that some classes of machine learning algorithms are compatible with this approach, namely based on the use of generative models that allow a distribution of the computation on a set of nodes. We then detail our architecture proposal, leveraging on the use of Web-of-Things technologies to ease integration into networks. The convergence of machine learning generative models and Web-of-Things paradigms leads us to the concept of virtual things exposing higher level knowledge by exploiting sensor data in the network. Finally, we demonstrate with a real scenario the feasibility and performances of our proposal.

Categories and Subject Descriptors
I.2.1 [**Computing Methodologies**]: Applications and Expert Systems; H.1.0 [**Information Systems**]: General

General Terms
Design

Keywords
Machine learning, Sensor network, Web-of-Things

1. INTRODUCTION
Nowadays, machine learning (ML) techniques are increasingly used in the context of sensor networks. ML approaches are able to learn from data and infer knowledge for two distinct objectives: (1) *optimization of the network parameters* for problems such as energy aware communication, optimal node deployment and resources allocation [7], and (2) *information processing* for tasks such as event detection, human activity recognition, human tracking, electricity consumption monitoring and smart heating control [16][13][15]. This paper focuses on this second type of applications.

So far, typical implementations are pushing the data acquired by Internet-of-Things (IoT) devices to server-side computers or cloud services in charge of performing data storage and ML computation. While this is a satisfactory configuration for some scenarios, it is not conceivable for many applications where privacy is a concern. Some settings are also decoupled from the external network (intranet mode) or are targeting to reduce data transfer for energy efficiency. Another concern is the traffic generated by the transfer of information through a border router, which can be overloaded and also representing a single point of failure. Finally, having the sensor network connected to the Internet represents a point of entry for attacks. With respect to the aforementioned problematic, deporting the ML runtime into the sensor network appears as a meaningful strategy. We also observe that IoT devices are showing increasing capacities in terms of storage and cpu, underlining the interest of such strategy.

In this paper, we propose an architecture for distributing machine-learning algorithms in a sensor network. The computational power of so-called things is exploited, forming a cloud space for ML algorithms. Moreover, we rely on Web technologies to form an open and reusable framework according to the Web-of-Things paradigm [9].

2. RELATED WORKS
The idea of distributing machine learning algorithms for building automation purposes has already been explored in [5]. Fuzzy logic is distributed across several agents by furnishing a XML document containing the model parameters. With this document, agents are able to automatically generate Java objects for the JADE agent framework, which is used for synchronization between the entities. Inference agents collect the outputs of the fuzzy logic agents and make

a decision before executing some predefined rules. Although going in the direction of decentralised systems, this solution requires a rather heavy infrastructure of powerful nodes and is probably not compatible with current IoT devices.

Trying to avoid heavy technologies such as Java by reusing common well-adapted Web technologies, the Actinium container [11] allows to execute applications written in JavaScript. It forms a cloud of machines dedicated to the execution of mashups that use sensors as input parameters. RESTful APIs are provided for configuration purposes and deployment. The container itself takes care of retrieving sensor data and forwarding the output of the applications to other nodes. In a similar approach, T-Res [6] targets a distributed space of applications running on constrained devices. They opted for Python as scripting language that is deployed and managed through a RESTful API split according to functionalities. The main improvement beside Actinium relies in the lower memory requirement of the container. Requiring less memory opens the possibility to empower constrained IoT devices with distributed computational capabilities.

Following the trend of using machine learning, the Google Prediction API [2] and BigML [1] are offering Web services running in the cloud for data analytics, prediction and classification purposes. These commercial products works with RESTful APIs for training and executing models. From a machine learning point of view, these systems are rather opaque regarding the type of algorithms used to train the models with no or few choices allowed for the user. In addition, no possibility to retrieve the model parameters is offered, making the customer dependent to the provider.

3. MACHINE LEARNING

In our discussion here, we assume to handle classification problems that represent the main type of problems tackled in machine learning. A classification problem typically requires two steps: *inference* and *decision*. The *inference* process (or training process) allows to extract knowledge from an observed system through the training data. The *decision* process (or testing process) associates a model (M_k) to a new observation (x). Assuming a probabilistic bayesian framework, two approaches can be used for this task :

Generative models. The inference problem consists in building a mathematical model for the likelihood $p(x \mid M_k)$ for each model M_k. The decision part is performed using the Bayes theorem to compute the posterior model probabilities with:

$$p(M_k \mid x) = \frac{p(x \mid M_k)p(M_k)}{\sum_k p(x \mid M_k)p(M_k)} \quad (1)$$

where $p(M_k)$ are the class prior probabilities. The class membership for the new observation x is obtained by comparing the posterior probabilities and selecting the one with the highest value.
The term *generative* comes from the fact that new observations can potentially be generated by the models. Naive Bayes classifiers, Gaussian Mixture Models and Hidden Markov Models are examples of models belonging to this category.

Discriminative models (or conditional models). The inference consists here to compute a mathematical model of the posterior probability $P(M_k \mid x)$. As for the *generative* models, the class membership for the new observation x is obtained by comparing the posterior probabilities. Support Vector Machines (SVMs), Artificial Neural Networks (ANNs) and Multinomial logistic regression are examples of algorithms belonging to this category. The term discriminative is related to the fact that the models are generally computing a decision border between classes instead of the class conditional probability density functions.

Both approaches have different strengths from a classification point of view. Discriminant models are usually preferred, leading to good tradeoffs between performance and size of training data. In fact, discriminative modelling can be seen as an easier task as we try to directly solve a problem (find the posterior probabilities) without solving a more general problem as in the generative modelling (find the likelihoods). Evidences show that discriminative modelling leads to a lower asymptotic error, however generative modelling approaches his error faster than the discriminative [8]. Finally the generative approach are reported to deal better with missing input values, outliers and unlabeled data [12]. Both approaches offer different possibilities for distributed computation. In our case we are interested in the possibility to distribute the computation load of the inference and decision processes on several devices. With generative models this is naturally done considering that each model contribution M_k can be computed individually. In other words, the likelihood computation of a given model is independent to the others and the computation of class conditional probabilities $p(x \mid M_k)$ (likelihoods) can be distributed in different nodes. This distribution is usually not possible or more difficult for discriminative models such as ANNs or SVMs where class parameters are shared in the model and where class decisions are computed in one pass.

4. ARCHITECTURAL DESIGN

We propose an architecture for ML algorithms in which classification tasks are delegated to constrained things, accessing their capabilities using RESTful APIs either with HTTP or CoAP [17]. The REST architectural style is extended to non-physical devices like model parameters for offering a standard and flexible manipulation. Our architecture is based on a set of agents having the capability to execute pre-deployed specific ML algorithms by receiving as input the model parameters (pre-trained). We opted for such an approach better than dynamically deploying code due to the memory restrictions of constrained things where an application container can not be usually executed.

In order to comply with a ML process for classification tasks, we introduce two novel entities, namely the *Virtual Sensor* and the *Virtual Class*. The general architecture and the interaction between the different entities is represented in Figure 1.

4.1 Virtual Sensor

Information processing applications can be considered as high level sensors. They indeed merge other sensors mea-

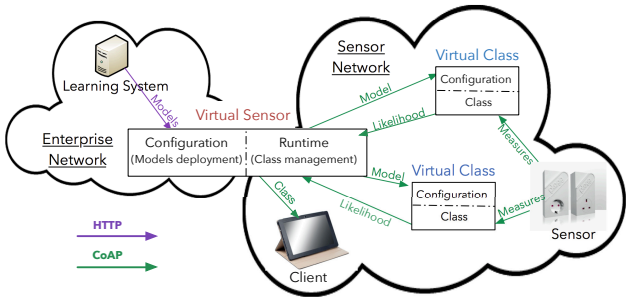

Figure 1: General architecture composed of Virtual Sensor and Virtual Classes with information exchanged during runtime.

sures in order to extract knowledge, as for example a class. In our architecture, a Virtual Sensor instance stands for a non-physical sensor exploiting a generative model. Within the sensor network, they appear as regular sensors performing measurements. This approach of hiding the complexity of the data fusion by acting as a conventional sensor increases reusability and ease of integration within already existing sensor networks. Moreover we decouple the configuration API from the runtime API in order to hide the ML part.

The configuration API is HTTP-based in order to be accessible by high-level applications like learning systems residing on the enterprise network. New Virtual Sensors can be created by PUTting to `http://virtual-sensor.example.com/virtualsensors/{id}` with the models parameters contained in the payload (the payload format is described in Section 4.4). The role of the Virtual Sensor is then to distribute the models on the constrained devices. It will therefore query for agents able to execute the provided models, becoming Virtual Classes. The selection is made according to the complexity of the models and the available memory on the agents. Once having successfully deployed the models, it will create a new resource on the runtime API.

For its part, the runtime API represents the output of the decision making task. Each Virtual Sensor instance owns its own runtime resource accessible over CoAP by issuing a GET request to an URI such as `coap://virtual-sensor.example.com/{virtual-sensor-name}`. We opted for CoAP as application protocol instead of HTTP as it is conceived for sensor networks. CoAP differs from HTTP with its small header and low memory requirement.

4.2 Virtual Class

As previously mentioned, a ML classification task relies on the computation of several class likelihoods. These classes holds a mathematical model composed for example of mean and covariance matrices for GMMs. For simplifying the interaction with those mathematical models representing a complex structure, we apply the same principle of high abstraction level by categorizing them in Virtual Classes. In our architecture, a Virtual Class is an entity running such a mathematical model on a smart thing. Similarly to the Virtual Sensor, we make them acting as regular sensors by decomposing their interface in two distinct APIs accessible with CoAP.

Virtual Class deployment is realized by a Virtual Sensor through several resources located on the configuration API. The API structure depends on the type of algorithm the agent is able to execute. There will be a resource for each model parameter type, allowing the replacement of specific parameters instead of the whole model. The first step is always the creation of the Virtual Class by sending a PUT request such as `http://virtual-class.example.com/virtualclasses/{id}` with the general model properties contained in the payload. The *run* subresource can be POSTed to launch the execution engine of the model. During this phase, the Virtual Class will register as observer [10] on the sensors used as input dimensions of the model. Additionally, it will schedule the execution of the algorithm according to the polling interval property of the model. HMM and GMM algorithms are requiring a polling strategy for executing the model at constant intervals even if no sensor value has changed.

The class API appears as the sensor resource used for retrieving the current likelihood of a class. The output of an algorithm will be copied by the execution engine as new value to its class resource. Virtual Sensors will send GET requests as in `http://virtual-class.example.com/{class-name}`. We decided to make this resource observable for notifying Virtual Sensors in a real-time manner.

4.3 Operating Modes

The decision making task is dedicated to the Virtual Sensor which will collect the current likelihoods of all its Virtual Classes. It then compares the likelihood values and, according to the Bayes rule, the model having the highest likelihood is designated as winner (assuming equal priors[1]). Depending on requirements of client applications, this decision making task can be executed according to two different modes, selectable during the creation of the Virtual Sensor.

In the *End-to-end* mode, the Virtual Sensor will sequentially request all its Virtual Classes for retrieving their current likelihood. Once having collected all the data, it will be able to perform the decision and returning the winning class to the client, as visible in Figure 2 (a). This mode, which can also be considered as on-demand approach, is particularly suitable for not time-critical applications where the round-trip time does not play a key role.

In contrary, the *Sync-based* mode is fully event-driven and is especially designed for real-time systems. Instead of retrieving the likelihoods only when requested, the Virtual Classes will notify their Virtual Sensor each time the likelihood has changed, as visible in Figure 2 (b). The decision making is performed as soon as a likelihood changes and is cached on the Virtual Sensor. 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 winning class changes.

4.4 Model Representation and Deployment

[1]Unequal priors can simply be taken into account by multiplying the likelihoods by the priors.

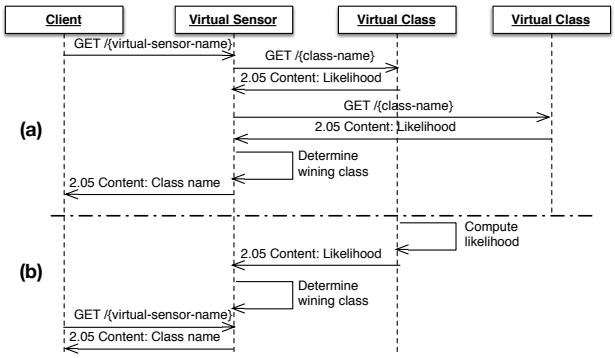

Figure 2: (a) End-to-end mode and (b) sync-based mode.

Exchanging information between different systems is always a tricky task. One has to agree about formalisms of data representation and structure. This is especially true for machine learning applications having to exchange mathematical models. The Data Mining Group attempted to solve this problematic by proposing the Predictive Model Markup Language (PMML) [4]. This XML schema semantically describes how to represent in XML several widespread predictive models. However, PMML do not describe widespread generative models such as GMMs or HMMs that we intend to use and matching our constraints as explained above. Furthermore, the PMML structure is rather complex and induces a high verbosity which is less suitable for smart things. We therefore developed our own data model, the *Machine Learning Exchange Format* (MaLeX) for describing generative models, including HMMs and GMMs. Our model relies on JSON commonly used in WoT applications and is defined as a JSON schema [3] describing general properties and mathematical models entities[2].

Numerical analysis systems performing the learning task will generate a JSON document provided during the creation of a Virtual Sensor. The Virtual Sensor first validates the document according to the MaLeX schema for ensuring semantic correctness. The general properties of the Virtual Sensors will be used to form an RDF document allowing its semantic discovery. Although MaLeX providing a strong formalism for representing models, it can not be used as such for creating Virtual Classes. Constrained devices have not enough memory for handling big JSON files. First, the representation of matrices made of float values is extremely verbose in JSON, as each character produces one byte of data. For this reason, we opted for a binary representation following the IEEE 754 notation of floats, already decreasing the required memory size by about 75%. Nevertheless, this does not allow a model deployment in one step on most devices, the matrices being too big. We overcome this restriction by deploying matrices stepwise. The whole process of model deployment is depicted in Figure 3.

5. IMPLEMENTATION

[2]Publicly available at `http://www.wattict.com/web/download/ml-model.json`

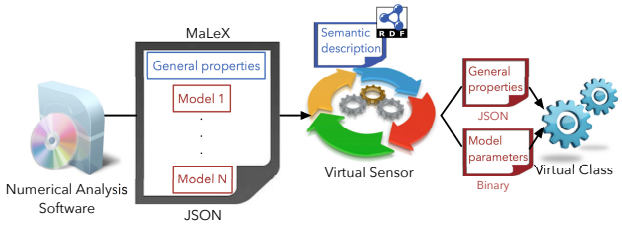

Figure 3: Model exchange between the different entities.

Regarding the technological aspects of our implementation, we developed the Virtual Sensor agent using Java, allowing to rapidly deploy it as a JAR application. For the HTTP interface used for deploying models, we opted for the embedded version of Jetty [3] requiring no dedicated server installation and being lightweight. For the runtime part using CoAP, our main criteria was the size of the library, reason why we chose JCoAP [4]. However, as this implementation is no longer maintained, we had to update it to comply with the version 18 of CoAP. The basis version of JCoAP working with a single-thread model, we improved it by parallelizing query processing with a thread pool. The validation of the MaLeX document is realized using the json-schema-validator for Java[5]. Because of the use of Java and the large amounts of data to be processed and validated (MaLeX documents), the Virtual Sensor can only run on hardware offering a certain amount of memory, such as a Raspberry Pi.

For the Virtual Class agent, our main concern was to use the lightest version possible of CoAP in order to leave the memory space for the model. We therefore opted for a C implementation of CoAP, namely microcoap[6]. This library being very minimalistic, we had to improve it by adding observation support.

6. EVALUATION

In this section, we report on the feasibility of deploying ML algorithms on IoT devices and we evaluate its performance with a real implementation.

6.1 Setup

We evaluate our proposal on a concrete scenario of appliance recognition using electricity consumption measures. Smart plugs IoT devices are placed between the appliance plug and the electrical wall socket. The aim of the recognition process is to analyse the electrical consumption signature to recognize the appliance category, e.g. coffee machine and its state of use, e.g. stand-by. Five OpenPicus FlyportPRO WiFi[7] modules are acting as agents able to execute HMM models able to compute class and state likelihoods through a standard Viterbi algorithm. A Raspberry Pi is used for running a Virtual Sensor agent.

[3]`http://www.eclipse.org/jetty/`
[4]`https://code.google.com/p/jcoap/`
[5]`https://github.com/fge/json-schema-validator`
[6]`https://github.com/1248/microcoap`
[7]`http://space.openpicus.com/u/ftp/datasheet/datasheet_flyportpro_wifi.pdf`

Training of the models is performed using data from the ACS-F2 signature database [13]. The database contains 2 hours of recording for 225 appliances of different brands and / or models using a sampling frequency of 10^{-1} Hz. For our evaluation, we selected 5 categories among the 15 available, in order to match the number of available OpenPicus. The selected categories are *Coffee machine*, *Laptop*, *Incandescent lamp*, *Compact fluorescent lamp* and *Mobile phone*. Six features are recorded by the smart plug: active power, reactive power, phase angle, RMS current, RMS voltage and frequency of the electrical network. We excluded the phase angle, RMS Voltage and frequency of the network, demonstrated to be not relevant or redundant [14].
We use Hidden Markov Models (HMMs) as machine learning algorithm for the appliance recognition. HMMs belong to the generative modelling category, permitting to easily distribute the computation resources on the different OpenPicus. Moreover they are particularly suitable for representing the state-based nature of the electrical signatures. The training is performed offline and the models parameters, as the transition and emission matrices are represented with MaLeX. This document is further transferred to the Raspberry Pi and models are then distributed on the OpenPicus in order to classify in real-time the upcoming data. We used a two states HMM topology to represent the on and standby states of the appliances. In a previous work the system provided good results when using about 30 Gaussians per state [13]. Given the memory restriction of the OpenPicus we scaled down the models to 8 Gaussians per state, trading off slightly the accuracy rate.

6.2 Performance

The performance of the architecture can be evaluated from two perspectives: machine learning and system. Regarding the first one, experiments showed a correct identification rates of appliances of about 90% which is comparable to server-side implementations and high enough for practical applications [13]. Regarding system evaluation, we evaluate the round-trip time and scalability for the two different operating modes.

In Figure 4 (a) we first compare the round-trip time for consecutive requests according to the end-to-end and sync-based modes. First, each request will have as consequence the sequential questioning of Virtual Classes. For this case, the average round-trip time for 3000 requests is 268 milliseconds. In the second case, the sequential questioning is bypassed and the current class is directly returned to the client. In this case, the average round-trip time was 32 milliseconds. The observation that the sync-based mode performs far better is not surprising. Indeed, each request is served from the cache of the Virtual Sensor without communicating with the Virtual Classes.

In the second evaluation, we evaluate the difference between the operation modes in terms of concurrent requests. As for the previous evaluation we performed requests using the end-to-end and sync-based modes. Figure 4 (b) shows the results when having up to 100 concurrent clients issuing 5 requests. Not surprisingly, the sync-based mode scales much better. The success rate of the end-to-end approach drops abruptly as soon as the number of concurrent clients is reaching 33. We can explain this limitation by the high amount of subrequests that are generated by the Virtual Sensor to the Virtual Classes. Indeed each client request will result in 5 requests to Virtual Classes. The bottleneck is in the Virtual Classes running on the OpenPicus which has no threading and handles requests one at a time. This creates a long queue of requests having as result some timeouts on the client side.

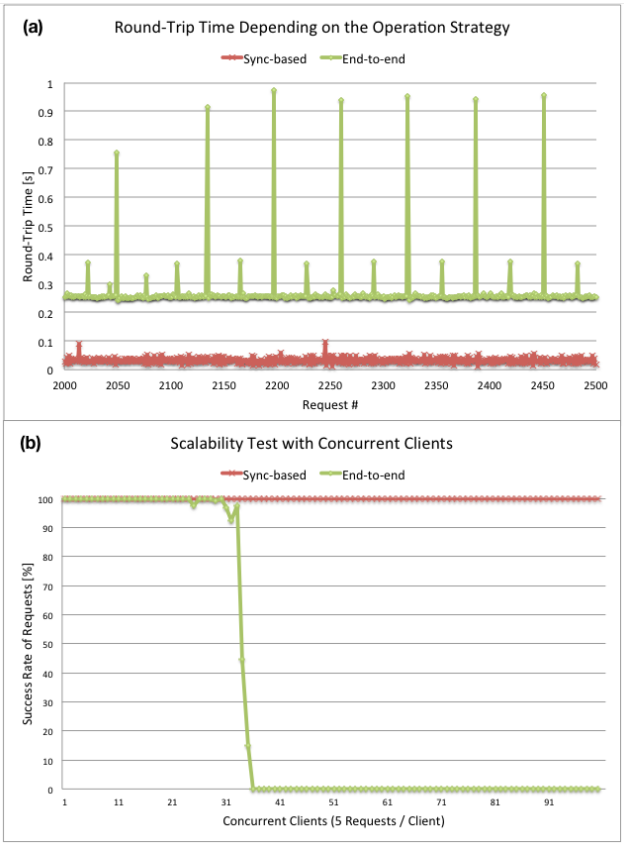

Figure 4: Round-trip time for 500 consecutive requests (a) and scalability in terms of concurrent requests (b).

6.3 Discussion

Distributed machine learning architectures open new dimensions for enhancing sensor networks with intelligent information processing. We show the possibility to use the computational power of constrained devices already present in the sensor network to run machine learning tasks, reducing the need of pushing data on dedicated computers located outside of the sensor network. However, we faced memory constraints that limited the deployment to 2-3 models on each node. As expected, not all information processing applications can be deployed in-network but this initial result is, according to us, convincing and encouraging. These limitations will be pushed back with future IoT devices that will embed more memory and more computing power.

RESTful APIs and Web technologies play a key role in our architecture. Beyond the fact that plenty of constrained devices are following the WoT paradigm, they bring together the machine learning community around one common archi-

tectural style. REST allows to simplify the manipulation of mathematical models by decomposing them in Web resources. Although MaLeX being today limited to HMM and GMM, it can be extended to other model representations and algorithms by defining new entities in the JSON schema, ensuring expandability towards new applications.

The performance evaluation tends to demonstrate that our system does not suffer from architectural mistakes limiting its usage. Moreover, it comes out that the sync-based mode should be preferred for real-time applications. However, the selection of the mode should also be done according to traffic efficiency. There is no clear answer to this as it depends on many parameters such as the number of customers, the frequency of queries, and the polling frequency of the models. The variability over time of these parameters also complicates the mode selection.

7. CONCLUSIONS

In this paper we have presented an architecture unifying the worlds of IoT and machine learning. The ever growing computational power of smart things is used for executing data-driven algorithms, usually located server-side or in clouds. Being able to run higher-level information processing tasks within the sensor network paves the way for new types of applications. The fact of keeping the sensors data inside the sensor network increases privacy while lessening the traffic. Web technologies are the foundations of our solution, allowing a full compatibility with today's sensor networks following the Web-of-Things paradigm. Using abstract entities like the Virtual Sensor and Virtual Class for representing ML processing tasks increases the ease of use of the system. For its part, the MaLeX format for exchanging ML models is the federating element for ensuring a low-coupling between numerical analysis software performing the training part and the runtime interface represented by Virtual Sensors.

Even if relying on constrained devices, the ML runtime shows very promising results. The round-trip time and scalability are satisfying for many IoT applications. Nonetheless, deploying large models composed of a large quantity of states and / or Gaussians is not yet feasible due to the memory restriction of smart things. In the future, we plan to apply our architecture to Smart buildings for executing models aiming at the recognition of user activities in houses. We also intend to reconsider the training part according to the Web-of-Things percepts for creating a common representation that can be distributed among multiple physical devices. Finally, we envision a cloud of machine learning entirely composed of smart things, eliminating the need of a dedicated architecture.

8. ACKNOWLEDGMENTS

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

9. REFERENCES

[1] Bigml. `https://bigml.com/`, 2014.

[2] Google prediction api. `https://developers.google.com/prediction/?hl=EN`, 2014.

[3] Json schema. `http://json-schema.org/`, 2014.

[4] Predictive model markup language (pmml). `http://www.dmg.org/v4-2-1/GeneralStructure.html`, 2014.

[5] G. Acampora and V. Loia. Fuzzy control interoperability and scalability for adaptive domotic framework. *IEEE Transactions on Industrial Informatics*, 1(2):97–111, 2005.

[6] D. Alessandrelli, M. Petraccay, and P. Pagano. T-res: Enabling reconfigurable in-network processing in iot-based wsns. In *Proc. of the 2013 IEEE International Conference on Distributed Computing in Sensor Systems*, pages 337–344, 2013.

[7] M. Di and E. M. Joo. A survey of machine learning in wireless sensor networks from networking and application perspectives. In *Proc. of the 6th International Conference on Information, Communications and Signal Processing*, 2007.

[8] T. G. Dietterich, S. Becker, and Z. Ghahramani. On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes. In *Proc. of the 2001 Conference on Advances in Neural Information Processing Systems*, 2001.

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

[10] K. Hartke. Observing resources in coap. draft-ietf-core-observe-14, 2014.

[11] M. Kovatsch, M. Lanter, and S. Duquennoy. Actinium: A restful runtime container for scriptable internet of things applications. In *Proc. of the 3rd IEEE International Conference on the Internet of Things*, pages 135–142, 2012.

[12] C. E. Rasmussen and C. K. I. Williams. *Gaussian Processes for Machine Learning*. the MIT Press, 2006.

[13] 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)*, 2014.

[14] 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.

[15] A. Ridi, C. Gisler, and J. Hennebert. A survey on intrusive load monitoring for appliance recognition. In *Proc. of the 2014 International Conference on Pattern Recognition*, 2014.

[16] A. Ridi, N. Zarkadis, G. Bovet, N. Morel, and J. Hennebert. Towards Reliable Stochastic Data-Driven Models Applied to the Energy Saving in Buildings. In *International Conference on Cleantech for Smart Cities & Buildings from Nano to Urban Scale (CISBAT 2013)*, pages 501–506, 2013.

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