CEML: Mixing and moving complex event processing and machine learning to the edge of the network for IoT applications

José Ángel Carvajal Soto
Marc Jentsch
Fraunhofer FIT
Sankt Augustin, Germany
angel.carvajal@fit.fraunhofer.de
marc.jentsch@fit.fraunhofer.de

Davy Preuveneers
iMinds-DistriNet
KU Leuven
Leuven, Belgium
davy.preuveneers@cs.kuleuven.be

Elisabeth Ilie-Zudor
Institute for Computer Science and Control
MTA SZTAKI
Budapest, Hungary
ilie@sztaki.mta.hu

ABSTRACT

The Internet of Things (IoT) is a growing field which is expected to generate and collect data everywhere at any time. Highly scalable cloud analytics systems are frequently being used to handle this data explosion. However, the ubiquitous nature of the IoT data imposes new technical and non-technical requirements which are difficult to address with a cloud deployment. To solve these problems, we need a new set of development technologies such as Distributed Data Mining and Ubiquitous Data Mining targeted and optimized towards IoT applications. In this paper, we present the Complex Event Machine Learning framework which proposes a set of tools for automatic distributed machine learning in (near-) real-time, automatic continuous evaluation tools, and automatic rules management for deployment of rules. These features are implemented for a deployment at the edge of the network instead of the cloud. We evaluate and validate our approach with a well-known classification problem.

ACM Classification Keywords

Computing methodologies Machine learning; Computing methodologies Machine learning: Supervised learning by classification; Information systems Data mining: Data stream mining; Computer systems organization Embedded and cyber-physical systems: Real-time systems

Author Keywords

Machine Learning; Complex Event Processing; Stream Mining; Internet of Things; Edge Computing.

INTRODUCTION

The Internet of Things is a blooming field where our world is being digitized by Connected Objects (CO). The data generation is growing rapidly in size, speed and nature. Analyzing data effectively in a cross-domain application field like the IoT where COs are producing data anytime and everywhere rising some open challenges [23]. The forecasts predict an exponential growth of COs, from more than 8.7 billion to around 20 to 50 billons of devices by 2020 [2, 15]. It is expected that each device will generate growing amounts of data, thanks to more efficient low-powered devices and networking protocols. With the current tendency, the transferred data through the internet will go from 4.4 Zettabytes (ZB) in 2013 to 44 ZB in 2020. With 2% being generated in the IoT today, we expect an increase up to 10% in 2020. Additionally, at 2013 we could store around 33% of the total data, but by 2020 our storing capacity will drop to 17% [40]. The dropping storage capacity is moving the Data Mining field into Stream Mining [17]. Stream Mining provides a solution for labelling data allowing Active Learning on streams [33]. However, Active Learning and On-line Learning are nonstationary and evolving environments [6], issues such as drifting [42] are more significant. On the one hand, much of the Data Streams Mining approaches concentrate on cloud deployments to handle the data growth. However, such deployments create bottlenecks, increasing the pains of the Internet [1] and the data processing latencies [27]. Centralizing the data increases hardware requirements which in turn increases the cost and reduces business opportunities [25]. Furthermore, data transfers in some deployments under real-time conditions is exorbitantly expensive and practically impossible [18].

A large number of sectors (e.g. security and investment services, media, banking, utilities, discrete and process manufacturing, healthcare) accumulate vast amounts of data [20] and along with the benefits from exploiting big data in a large spectrum of application domains. The legal, privacy and security aspects are still to be dealt with [8]. E.g., in Smart Cities where many devices and system interact and overlap, the distribution of responsibilities and collaborated actuation is a significant issue [10]. In the Smart Health domain, the data *controllership*, privacy and ethics represent a greater issue [39]. Therefore, we need to provide technologies that address data analytics in a holistic manner to handle the aforementioned concerns.

Ubiquitous Data Mining [16] contributes to avoiding the above listed issues by processing the data where it was produced or

IoT'16, November 07 - 09, 2016, Stuttgart, Germany

ISBN 978-1-4503-4814-0/16/11…$15.00

DOI: http://dx.doi.org/10.1145/2991561.2991575

close to the source at the edge of the network. However, recent work in this area left several questions unanswered [17]. How to manage the ubiquitous streams? How should we manage remotely the evolving learning models? How to continuously evaluate the model? These are some of the open questions we aim to address in this work. The contribution we present in this paper is our framework, Complex-Event Machine Learning (CEML), and a reference implementation of this framework, the IoT Learning agent. The framework was developed for real-time automatic non-persistent stream mining. The implementation was developed to be deployed in the cloud or in embedded devices at the edge the network.

This paper is divided as follows. In section 2, we discuss the evolution of learning and stream mining in the last years. In section 3, we present the CEML framework followed by 5 subsections highlighting the different phases of the framework. In section 4, we discuss the implementation of the framework, and section 5 contains the results of the evaluation of this implementation. Finally, we conclude in section 6 summarizing our contributions and highlighting opportunities for further research.

RELATED WORK

Early works on combing stream processing and machine learning can be found on the Diamond Eye system [9]. Diamond Eye was a pioneer image recognition distributed system used to extract spatial objects from real-time images. The system was one of the first of its time. However, it was not horizontally scalable.

The *MobiMine* was the first Ubiquitous Data Mining application developed for the PDA. However, the mining components reside at the server side [22]. The need for on-board mining was first discussed in [37], without presenting any implementation or results.

In [28] the necessity of distributed data mining and possible applications are studied. The architecture may scale horizontally, but the system is storage-based and an implementation is not presented. The data explosion problem is pushing the limits of the Data Mining technologies, as confirmed in [21].

The SAMURAI architecture [29] presents a horizontal scalable system combining data and stream mining. However, the system is not optimized for mixed deployments operating at the edge of the network or embedded devices.

The Ubiquitous Data Mining first presented here [18] discusses technologies for resource constrained devices with results presented in [17]. In this work the problem of mining in ubiquitous systems is explored. Additionally, some algorithms to handle learning in constrained devices are discussed. However, the work is heavily oriented towards algorithmic aspects, and leaves fundamental issues open, as summarized in the conclusion.

Formalization and a taxonomy of streams systems can be found in this [12] meticulous study. Albeit, without covering any Machine Learning interaction. The Machine Learning techniques applied on Data Streams such as Active Learning or Stream Mining have bean extensively compiled in [19] and [4]. However, both works stay on the algorithmic and formal aspect of it. Moreover, they present few studies on how the models can be evaluated continuously. While [19] presents two formal models, [4] leave it as an open issue to be researched.

The current paper proposes a framework for the management of the learning process in IoT environments, answering some of the questions left open by [17] and provides an implementation for continuous evaluation on stream addressing open issues pointed by [4] and [18].

COMPLEX-EVENT MACHINE LEARNING

The Complex-Event Machine Learning (CEML) is a framework that combines Complex-Event Processing (CEP) [12] and Machine Learning (ML) [5] applied in the IoT. Applied in the IoT means that the framework was developed to be deployed everywhere, from the edge of the network to the cloud. This also means that the framework can manage itself and work autonomously manner.

The following section briefly describes the different aspects that CEML covers. To be applicable in the IoT, the framework must automate the learning process and the deployment management. This process can be broken down in different phases: (1) The data must be collected from different sensors, either from the same devices or in a local network. (2) The data must be pre-processed for attribute extraction. (3) The learning process takes place. (4) The learning must be evaluated. (5) When the evaluation shows that the model is ready, the deployment must take place. Finally, all these phases happen continuously and repetitively, while the environment constantly changes. Therefore, the model and the deployment must adapt too.

Data Propagation Phase

Data in the IoT is produced in several places, protocols, formats, and devices. This work does not address in depth the problem of data heterogeneity. Nevertheless, the learning agents require a mechanism to acquire and manage the heterogeneity of the data. The mechanism must be scalable and, at the same time, the protocol should handle the asynchronous nature of IoT. Finally, the protocol must provide tools to handle the *pub/sub* characteristics of the CEP engines. Therefore, we have chosen MQTT [3], a *well-established* IoT client server publish/subscribe messaging transport protocol. The topic base message protocol provides a mechanism to manage the data heterogeneity by making a relation between topics and payloads. It allows deployments in several architectures, OS, and hardware platforms; and addresses the constraints at the edge of the network. The protocol is payload agnostic, and as such allows for maximum flexibility to support several types of payloads.

Data Pre-Processing (Munging) Phase

Typically ML is tied to already stored datasets, which produces several drawbacks. Firstly, the learning can take place just with persistent data. Secondly, usually the models generated are based on historical data, not current data. Both constrains, in the IoT has dire consequences. It is neither feasible nor profitable to store all data. Also, embedded devices do not

have much storage capacity which makes it impossible to use ML on them. Furthermore, IoT deployments are commonly exposed to ever-changing environments. Using historical data for off-line learning could cause outdated models learning old patterns rather than current ones, producing drifted models. In some IoT deployments, there is no historical data available, e.g. in the case of waste fill levels per bin in cities, it would take several years to create enough historical data. Therefore, there is a need for non-persistence manipulation tools. This is precisely what the CEP engine provides in the CEML framework. This means, the CEP engine decides which and how the data is manipulated using predefined CEP statements deployed in the engine. Each statement can be seen as a *processing node*, to which each learning model is subscribed. Each update of the subscribers provides a sample to be learnt in the learning phase.

Learning Phase

There is no fixed algorithm selection in our framework, due to restrictions imposed by the problem domain. For example, in extreme constrained devices, algorithms such as Algorithm Output Granularity (AOG) [16] may be the right choice. In other cases where the model changes quickly, one-shot algorithms may be the best fit. Artificial Neural Networks are good for complex problems but only with stable phenomena. This means that the algorithm selection should be made case-by-case. Our framework provides mechanisms for the management and deployment of the learning models, and the process of how the model is fed with samples. In general, the process is based on incremental learning [36], albeit with online and non-persistent data. The process can be summarized as follows. The samples, without the target provided in the last phase, are used to generate a prediction. The prediction will be sent to the next phase. Thereafter, the sample is applied to train the model. Thus, all samples are used for the learning process.

Continuous Validation Phase

This section describes how the validation of the learning models is done inside the CEML. This phase does not influence the learning process. Furthermore, this section does not validate the CEML framework itself.

The evaluation for distributed environments or embedded devices is not extensive addressed in the literature. E.g. in the extensive work [19], validation is just 3.3% of the work. We think learning ML models validation is a challenging topic in real-time environments, and needs further research. I.e. the typical Data Mining cross-validations are not practical for Stream Mining, especially in embedded devices. There are two addressed strategies. Either we holdout an evaluation dataset, this means taking a control subset for given time-frame (time window). Or we use *Predictive Sequential*, also known as *Prequential* [13], in which we asses each sequential prediction against the observation. The holdout approach produces several problems. Firstly, sampling measurements in a continuous streaming environment raises some questions [18]. How big must the sample set be? When should the evaluation take place? Or, how often? Secondly, continuous cross-validation will trigger regular batch processes, which increase the memory and CPU needs. Therefore, to solve the continuous validation problem, we need to scope the validation somehow. We decided to follow the Prequential approach by creating *data-windows*. In this manner, the validation is carried out when a *data-window* is full. This solves the problem sampling size of an *undetermined-size* data stream. Still, data windows do not solve the computational complexity of the cross-validation neither the extra memory needs. Thus, we needed to adapt the current validation mechanisms. The following section describes the continuous validation for a **classification** problem, but it can be applied for other cases.

Instead of accumulating a sample for validation, we analyze the predictions made before the learning took place. Each prediction is assessed each time an update arrives. The assessment is an entry for the confusion matrix [35] which is accumulated in an *accumulated confusion matrix*. The matrix contains the accumulation of all assessed predictions done before. In other words, the matrix does not describe the current validation state of the model, instead the trajectory of it. Using this matrix, the accumulated validation metrics (e.g. Accuracy, Precision, Sensitivity, etc.) are being calculated.

This methodology does have some drawbacks. First, the accumulation nature of the validation may show different results than the model currently has. Second, the validation will have meaningful results only when the model converges towards something. The first problem is especially difficult when the model takes lots of measurements to learn. This could end up by not reaching the expected validation metrics results. Therefore, we bounded the evaluation in a *tumble-data-window*. The confusion matrix is accumulated and the evaluation metrics are calculated within a *data-window* of size defined by the number of data-points. Before the window had reached its size, the evaluation metrics are considered **not valid**. This window is named *Learning Window*. At the moment the *Learning Window* grows to the predefined size, then it becomes the *Learnt Window*. The *Learning Window* then tumbles to a *Learnt Window* and a new empty *Learning Window* is created. The process for the new *Learning Window* starts over, while the *Learnt Window* keeps evaluating the model as before with the difference that the metrics are now **valid** for evaluation. When the new *Learning Window* reaches the learnt status, it replaces the current *Learnt Window*. This process is repeated indefinitely. We named this approach the *Double Tumble Window Evaluation*.

The approach reduces the complexity and the weight of historical evaluation data which do not represent the model. But it must be balanced. A too narrow window will provide local validation, a too big window will still have the problem of accumulated data. Nonetheless, the approach provides advantages. First, once the first *learnt windows* is reached, the system can provide an instant evaluation. Second, the evaluation tumbles with the model, this means, if the phenomenon which is being learnt drifts, the evaluation changes accordingly as the model adapts.

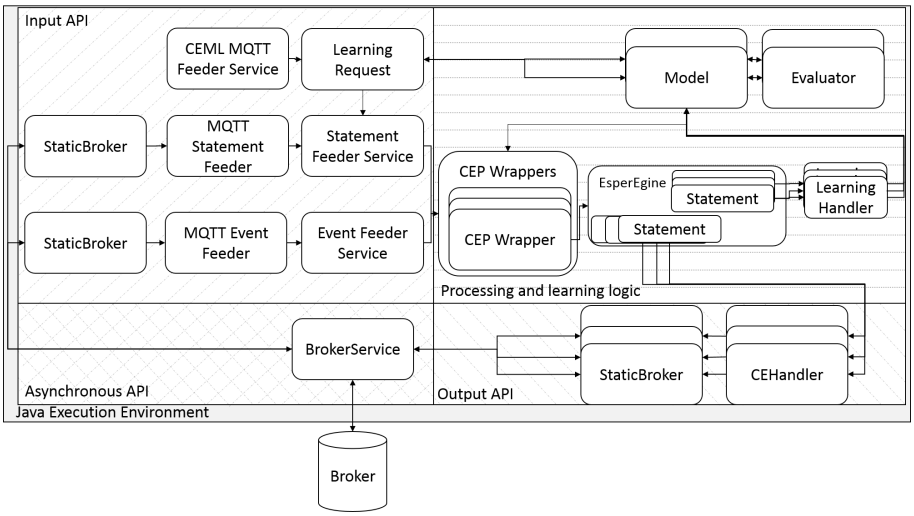

Figure 1. Architecture Sketch of the IoT Learning agent

Deployment Phase

The continuous validation opens the possibility for making an assessment of the status of the model each time a new update arrives, e.g. if it is accrued or not. Using this information, the CEML framework has the capability to decide if the model should or should not be deployed into the system at any time. If the model is *behaving well*, e.g. if the prediction of someone is present or not, then it should be deployed, otherwise it should be removed from the deployment. The decision is taken by user provided thresholds for the evaluation metrics. If the threshold is reached, the CEML inserts the model into the CEP engine and starts processing the streams using the trained model. If the evaluation metrics drops and the thresholds are not reached, then the opposite happens.

Real-time Phases

Finally, all phases happen one after another continuously and autonomously every *time* new input arrives to the CEML process. This means, the framework provides an assessment of all phases at *any time*. Additionally, the framework provides such an assessment *each time* an update is available. Altogether, it creates a moving snapshot of a learning model, which has a perception of the environment in *real-time*, or *near-real-time* considering the sensory, transporting, and processing lag.

IMPLEMENTATION

The current implementation has been developed out of the Data-Fusion Manager (DFM) [7], a Java-based LinkSmart® micro-service for the Smart Cities in the ALMANAC project [32]. The introduction of the CEML and the adaptation of the DFM took place in the IMPReSS project [31]. The current service, named IoT Learning Agent (LA) can be found in [11] and is part of our contribution.

The data collection phase was implemented using the Mosquitto broker and the Paho Java client. While the LA can adapt to several payloads, they must be defined. Therefore, we selected as base payload the OGC SensorThing standard [11]. The data pre-processing and deployment phase implementations are leveraging the Esper [14] engine. The learning phase was implemented using Weka [24].

Finally, the continuous evaluation phase was implemented from scratch. The selection criteria for this implementation were: (1) be able to run them efficiently in embedded devices, (2) provide a multi-platform solution, (3) based on a well-known and tested implementation, and (4) proven long-term development commitment.

Architecture

An architecture sketch is presented in figure 1. The learning system presents a publish/subscription architecture pattern, on its external communication with its clients (devices, applications or services) as well as in its internal logic. On the one hand, the clients communicate by *publishing* asynchronous messages to which the IoT Learning Agent (**LA**) or **LA**s are *subscribed*. On the other hand, the **Model**s *subscribe* to the data, provided by the clients, through *Statements*/queries; and sending back by *publishing* the selected information by its clients. The internal and external *pub/sub* architecture allows the system to distribute the data processing into external or internal *processing nodes*, depending on the needs. E.g. having the processes p_1 and p_2, where $p_1 = c * (b + c)$ and $p_2 = d * (b + c)$; the process can be split such that $p_i = a + b$ then $p_1 = c * p_i$ and $p_2 = d * p_i$. Each process is transformed into statements, in which can be deployed into one **LA** or several. In case of being deployed into one **LA**, the *processing nodes subscribe* to each other internally. While in the case of several **LA**, the *subscriptions* are between **LA**s distributing the processing in different services. Furthermore, the **LA** uses the **Broker** as data source, data recipient, control input and output. All those roles of the **Broker** can be realized by one **Broker** (as is shown in 1) or several, one per role. Regarding the data heterogeneity in the IoT, the system managed in twofold. Firstly, the different types of data payloads and data sources are separate in different topics. Secondly, the fusion or transformations needed on the data are made by **Statement**s and

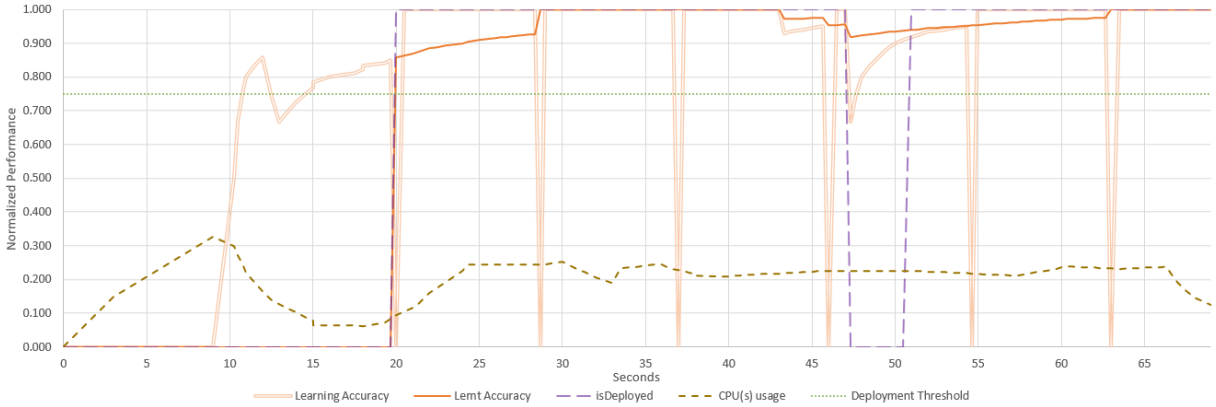

Figure 2. The Accuracy and CPU usage performance in a Raspberry PI 2 Model B measured in second and normalized values

put them together in internal streams or published externally in their corresponding topics. The internal streams are managed by **Handler**s depending on the type of statement produced the stream. The system orchestrate all the streams involved on the learning process of the **Model** through the **Learning Request**. While, each **Model** is responsible of *evaluation* themselves, and this process is done by the **Evaluator**.

The current implementation realized the architecture as follows. The **Broker** or **Broker**s role is taken by the MQTT mosquitto **Broker**. The *real* connection between the **LA** and one **Broker** is established and managed by the **BrokerService**. This means that there is one **BrokerService** per **Broker** in the deployment, and the connections are established using Paho clients. While the *logical* connections between the components and the **Broker**/s is managed by **StaticBroker**. This means, there are as many **StaticBroker** instances as there are components using a connection to a **Broker**. These logical connections are managed by one thread per topic, and each topic has a queue. I.e. in the Figure 1 there are one **Broker** deployed and five components that use a connection to that broker, therefore there is one **BrokerService** and five **StaticBroker**. The **Feeder**s are the components that managed the types of payloads, which are: data/events, statements, and learning request, managed by **EventFeederService**, **StatementFeederService**, and **CEMLFeederService**, respectively. The event payload enveloped is based on the *OGC SensorThing* standard [26]. The **Statement**s are deployed in the CEP engines by the **StatementFeeder**, which are access by the **CEP Wrappers**. The **CEP Wrappers** handles all the **CEP Wrapper**s as a single entity, while the **CEP Wrapper** hides the complexity of each CEP engine implementation. In the current **LA** implementation, there are two **CEP Wrapper**s one for Esper CEP [14] and one for WSO2 CEP [41]. Each **Statement**s inside the CEP engine produce streams which in a **LearningRequest** fall in three categories: *learning streams*, *deployment streams*, and *auxiliary streams*. The *learning streams* are the input of the **Model**s and are used by the **Evaluator** for *evaluation* as well. The contentious learning process is known as *learning process*, and is managed by **LearningHandler**. The *deployment streams* are the streams to be used when the **Model** has been evaluated as deployable by the **Evaluator**; and the streams are publish by the **CEHandler** on the output **Broker** accordantly to the **Statement**. The *auxiliary streams* are used as internal *processing nodes* by the *learning streams* or *deployment streams*, and their visibility is restricted to the CEP engine itself. Therefore, they don't have any handler. Finally, the **Model**s hides the complexity of learning algorithms to the system and provide a common interface. In the current implementation, there are two **Model** implementations, one for classification problems using Weka [24] `UpdateableClassifier` several implementations; and one for regression using *Auto-Regressive* which uses *Artificial Neural Networks* implemented by deep learning 4j [34].

Scalability and IoT deployments

In the IoT there is a huge variety of deployment scenarios ranging from hierarchical/vertical client-server centralized deployments, to highly distributed peer-to-peer without any central control deployments. This applies to the hardware platforms too, which vary from simple edge devices to highly versatile virtual hardware in cloud clusters. The **LA** implementation allows to be deployed on a large amount of devices, from edge devices to cloud deployments. The architecture allows horizontal scalability, in a distributed non-centralized or in single centralized deployment. I.e. a processing analysis system can be realized by a single **LA** with internal *processing nodes*; it can be a mesh of **LA**s interconnected to each other as external *processing nodes*. Both deployments types could be achieved in a cloud or edge system. The cloud deployment can be a single powerful analytic service, while by interconnecting several gateways, each with an **LA**; produces a mesh analytic ecosystems of services. Finally, a deployment could be a combination of both. The versatility, adaptability, and autonomy of the **LA** service is what makes it a scalable IoT agent.

PRELIMINARY RESULTS

In this section, we briefly evaluate the learning and computational performance of the implemented framework, monitoring the performance of an embedded deployment on a classification problem in real-time. The IoT Learning Agent is being used to improve the presence detection in the Universidade Federal de Pernambuco (UFPE) in Recife, Brazil in scope of Smart Building project IMPReSS; and for waste "production"

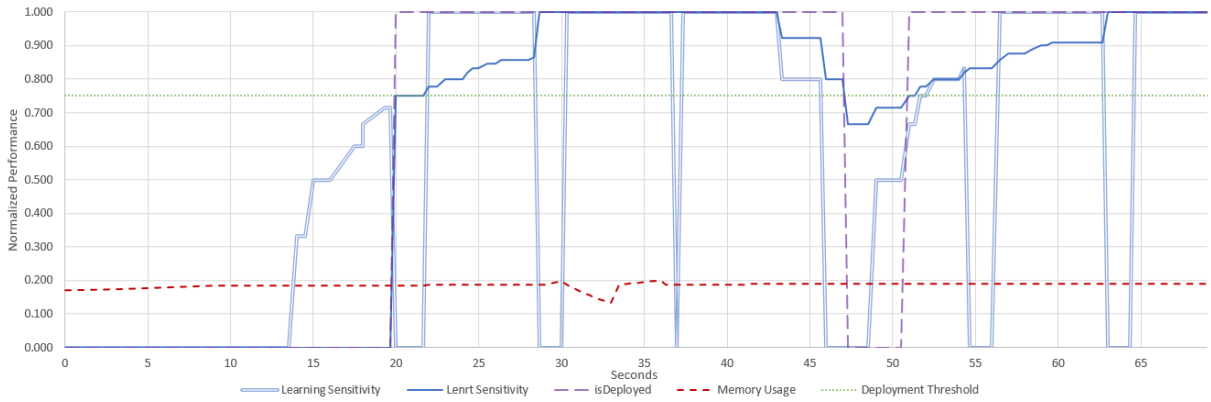

Figure 3. The Sensitivity and Memory usage performance in a Raspberry PI 2 Model B measured in second and normalized values

forecasting in Turin, Italy in the ALMANAC project. Nevertheless, instead of introducing a new case study, we considered it necessary to show the results in a *well-known* problem that is easier to replicate and reproduce. Therefore to validate our approach, we created a reproducible experiment which we consider can show similar level of computational complexity and it can show the learning and computational performance of the current implementation. Hence, we took a classic dataset [38] in which each feature of a sample in the dataset is packed in a measurement. Each measurement is sent through the network as if it was generated by a sensor. Then the broker and the **LA** are deployed on a Raspberry PI 2 Model B that receives the data and processes it using a learning request. The reason for selecting a Raspberry PI, it is to select a well-known hardware accessible to mostly anyone. With an open source service such as the **LA**, a widely-known dataset such as iris dataset, and common hardware such as Raspberry PI; we believe that we present a reproducible benchmark by mostly anyone. It is noteworthy that the experiment was not intended to test the limits of the system, just to validate the feasibility of the approach and to show the performance. In the experiment, the measurements were sent at an average of 19.17 measurements per second, one measurement every 52.17 ms. The selected algorithm implementation was the *NaiveBayesUpdateable*, because the algorithm provided good results in preliminary tests. Two validation metrics are given, the Accuracy and the Specificity, with both a deployment threshold of 0.75. The whole process takes 78 seconds. From idle to the deployment of the learning request, processing and learning, and back to idle again. Out of this time, just 67 seconds were used for processing and learning.

In figure 2 and 3, we present the most significant results. In 2 the Learning Accuracy (light-orange double-lined), Learnt Accuracy (continuous dark-orange line), the CPU usage (discontinuous brown-yellow line) is presented. In 3, the Learning Sensitivity (light-blue double-lined), Learnt Sensitivity (dark-blue line) and Memory usage (discontinuous light-red line) is shown. In both figures, the deployment threshold (doted green line) is shown, and the instant that the deployment takes place (represent as 1) or is removed (represented as 0) is shown (discontinuous dark-purple line). All values are presented as normalized values for the Y-axis, and the X-axis shows the *real-time* elapsed in seconds.

The results show that the learning performance did *well*, regarding merged quickly, with only two periods of time where the model was removed from the active deployment. It took around eleven seconds to reach the threshold, and twenty seconds for the sensitivity. We can see that the deployment readiness is reached at the same time as the first window tumbles. There is a small "instability" period between the second 47 and 50 in which the deployment is removed temporarily. But the charts show how the windows "protect" from local instability. In the performance perspective, the system reaches a peek at the beginning caused by the combination of the deployment of the request, and the first rush of measurements. In general, the CPU and memory percentage do not change much with a peek CPU of 32.6% and a 5.2% of memory around the second 8 and 30, respectively.

CONCLUSION

In [17] 13 issues for learning in the IoT where left open. The CEML framework addresses 10 out of the 13 challenges as follows:

1. Handling the continuous flow of data streams: This is done by the stream statements inside the CEP engine using continuous streams for learning an evaluating.
2. Unbounded memory requirements: The use of CEP engines in stream windows allows the intelligent usage of the memory as is needed, dropping it otherwise.
3. Transferring data mining results over a wireless network with limited bandwidth: This is partially handled. MQTT is a reliable low-bandwidth lightweight protocol developed for satellite monitoring of pipelines. Nevertheless, this paper does not address the physical layer.
4. Modelling changes of mining results over time: The CEML is a continuous automatic learning mechanism. The learning models will adjust as they learn.
5. Interactive mining environment to satisfy user requirements: The IoT Learning agent provide an REST API. Thus, update the learning request is possible, as well as, obtaining live or on-demand updates.

6. Integration between data-stream management systems and ubiquitous data-stream mining approaches: The CEML provides a REST API for managing each kind of request independently. Thus, the learning request can be manage as a whole, including the involved streams. Besides, the the streams can be managed individually as single stream statement. Additionally, the MQTT API provides a multicast API so the that in distributed multi-agent deployment, the agents can be managed as one, groups, or as one entity.
7. The relationship between the proposed techniques and the needs of real-world applications: Legal, ethical and technical reasons are part of the motivation. E.g. the storage constrain, the legal constraints in health domain.
8. Data pre-processing in the stream-mining process: This is handled in the pre-processing phase of the CEML.
9. The technological issue of mining data streams: The implementation presented here shows how the system behaves in a real-time environment.
10. The formalization of real-time accuracy evaluation: This is addressed by the *Double-Tumble-Window* Evaluation.

In summary, we presented the constraints of the IoT, focusing on analytics. We provide a theoretical framework and an implementation to manage learning in the IoT. Last but not least, we have shown how the framework behaves for a well-known classification problem, and discuss results for a deployment of a real-time scenario on top of an embedded device.

For future work we foresee the following mid-term work: (1) We will present our results in a real-world problem, i.e. we are working on the waste domain. (2) An extensive study in the scalability on data processing perspective in a edge of the network perspective. (3) Addressing the security aspect of the framework. (4) Generalize the evaluation for regression problems. (5) Present evaluate different kind of IoT deployments. (6) We will investigate the applicability of our framework for different vertical domains and applications in order to ascertain to what extent the current results can be generalized to other case studies.

Acknowledgment

The work presented in this paper was supported by the EU research projects ALMANAC (Project no. 609081), IMPReSS (Project no. 614100), EXCELL[30] (Project no. 691829) and by the Research Fund KU Leuven.

REFERENCES

1. 2010. *Growing Pains: Bandwidth on the Internet*. Technical Report. Internet Society. 20 pages. http://www.internetsociety.org/sites/default/files/Growing%20Pains-%20Bandwidth%20on%20the%20Internet.pdf
2. 2011. *More than 50 billion connected devices*. Technical Report 284 23-3149 Uen. Telefonaktiebolaget L. M. Ericsson. 12 pages. http://www.ericsson.com/openarticle/mwc-connected-devices_1686565587_c
3. 2014. MQTT Version 3.1.1. (2014). http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html Accessed: March, 2016.
4. Charu C Aggarwal. 2007. *Data streams: models and algorithms*. Vol. 31.
5. Christophe Andrieu, Nando De Freitas, Arnaud Doucet, and Michael I Jordan. 2003. An introduction to MCMC for machine learning. *Machine learning* 50, 1-2 (2003), 5–43.
6. Plamen Angelov, Dimitar Filev, and Nikola Kasabov. 2008. Guest Editorial: Evolving Fuzzy Systems: preface to the special section. *IEEE Transactions on Fuzzy Systems* 16, 6 (2008), 1390–1392.
7. Dario Bonino, Maria Teresa Delgado Alizo, Alexandr Alapetite, Thomas Gilbert, Mathias Axling, Helene Udsen, Jose Angel Carvajal Soto, and Maurizio Spirito. 2015. ALMANAC: Internet of Things for Smart Cities. In *The 3rd International Conference on Future Internet of Things and Cloud (FiCloud2015)*. IEEE.
8. Danah Boyd and Kate Craford. 2011. Six Provocations for Big Data. *A Decade in Internet Time: Symposium on the Dynamics of the Internet and Society* (2011).
9. Michael C Burl, Charless Fowlkes, Joe Roden, Andre Stechert, and Saleem Mukhtar. 1999. Diamond Eye: A distributed architecture for image data mining. In *AeroSense'99*. International Society for Optics and Photonics, 197–206.
10. José Ángel Carvajal Soto, Otilia Werner-Kytölä, Marco Jahn, Jaroslav Pullmann, Dario Bonino, Claudio Pastrone, and Maurizio Spirito. 2016. Towards a Federation of Smart City Services. *Proceeding of International Conference on Recent Advances in Computer Systems* (2016).
11. Carvajal Soto, José Ángel. 2015. LinkSmart IoT Learning Agent. (2015). https://linksmart.eu/redmine/projects/iot-data-processing-agent
12. Gianpaolo Cugola and Alessandro Margara. 2012. Processing flows of information: From data stream to complex event processing. *ACM Computing Surveys (CSUR)* 44, 3 (2012), 15.
13. A Philip Dawid. 1984. Present position and potential developments: Some personal views: Statistical theory: The prequential approach. *Journal of the Royal Statistical Society. Series A (General)* (1984), 278–292.
14. Espertech Inc. 2006. Esper. (2006). http://www.espertech.com/ Accessed: March, 2016.
15. Dave Evans. 2011. *The Internet of Things: How the Next Evolution of the Internet Is Changing Everything*. Technical Report. Cisco Systems, Inc. 10 pages. http://www.cisco.com/c/dam/en_us/about/ac79/docs/innov/IoT_IBSG_0411FINAL.pdf
16. Mohamed Gaber, Shonali Krishnaswamy, and Arkady Zaslavsky. 2004. Ubiquitous data stream mining. In *Proceedings of Current Research and Future Directions Workshop Proceedings, held in conjunction with the Eighth Pacific-Asia Conference on Knowledge Discovery and Data Mining*.

17. Mohamed Medhat Gaber, Shonali Krishnaswamy, and Arkady Zaslavsky. 2005a. *Advanced Methods for Knowledge Discovery from Complex Data*. London, Chapter On-board Mining of Data Streams in Sensor Networks, 307–335.
18. Mohamed Medhat Gaber, Arkady Zaslavsky, and Shonali Krishnaswamy. 2005b. Mining Data Streams: A Review. *SIGMOD Rec.* 34, 2 (June 2005), 18–26.
19. Joao Gama. 2010. *Knowledge discovery from data streams*. CRC Press.
20. Manyika James, Michael Chui, Brad Brown, Jacques Bughin, Richard Dobbs, Charles Roxburgh, and Angela Hung Byers. 2011. *Report: Big data: The next frontier for innovation, competition, and productivity*. Technical Report. McKinsey Global Institute. 156 pages. http://www.mckinsey.com/~/media/McKinsey/Business%20Functions/Business%20Technology/Our%20Insights/Big%20data%20The%20next%20frontier%20for%20innovation/MGI_big_data_full_report.ashx
21. Stephen Kaisler, Frank Armour, Juan Antonio Espinosa, and William Money. 2013. Big data: Issues and challenges moving forward. In *System Sciences (HICSS), 2013 46th Hawaii International Conference on*. IEEE, 995–1004.
22. Hillol Kargupta, Byung-Hoon Park, Sweta Pittie, Lei Liu, Deepali Kushraj, and Kakali Sarkar. 2002. MobiMine: Monitoring the stock market from a PDA. *ACM SIGKDD Explorations Newsletter* 3, 2 (2002), 37–46.
23. Stamatis Karnouskos. 2011. Cyber-Physical Systems in the SmartGrid. In *2011 9th IEEE International Conference on Industrial Informatics (INDIN)*. IEEE, 20–23.
24. Machine Learning Group at the University of Waikato. 1993. Weka. (1993). http://www.cs.waikato.ac.nz/ml/weka/ Accessed: March, 2016.
25. Aapo Markkanen. 2015. *IoT Analytics Today and in 2020*. Technical Report. Allied Business Intelligence, Inc. 10 pages. http://www.sigma-software.ca/wp-content/uploads/2015/07/ABI_Research_IoT-Analytics-Today-and-in-2020.pdf
26. Open Geospatial Consortium. 2015. OGC SensorThings API. (2015). https://ogc-iot.github.io/ogc-iot-api/
27. Byung-Hoon Park and Hillol Kargupta. 2002a. Distributed Data Mining: Algorithms, Systems, and Applications. Oxford Internet Institute's, 341–358.
28. Byung-Hoon Park and Hillol Kargupta. 2002b. Distributed data mining: Algorithms, systems, and applications. (2002).
29. Davy Preuveneers and Yolande Berbers. 2014. SAMURAI: A Streaming Multi-tenant Context-Management Architecture for Intelligent and Scalable Internet of Things Applications. In *Intelligent Environments (IE), 2014 International Conference on*. IEEE, 226–233.
30. Project Coordinator: Elisabeth Ilie-Zudor. 2016. EXCELL Project. (2016). http://excell-project.eu/
31. Project Coordinator: Markus Eisenhauer. 2013. IMPReSS Project. (2013). http://impress-project.eu/
32. Project Coordinator: Maurizio Spirito. 2013. ALMANAC Project. (2013). http://almanac-project.eu/
33. Burr Settles. 2009. *Active Learning Literature Survey*. Computer Sciences Technical Report 1648. University of Wisconsin–Madison.
34. skymind. 2016. Deep Learning 4j. (2016). http://deeplearning4j.org/
35. Stephen V. Stehman. 1997. Selecting and interpreting measures of thematic classification accuracy. *Remote Sensing of Environment* 62, 1 (1997), 77 – 89. DOI: http://dx.doi.org/10.1016/S0034-4257(97)00083-7
36. Nadeem Ahmed Syed, Syed Huan, Liu Kah, and Kay Sung. 1999. Incremental learning with support vector machines. (1999).
37. Steve Tanner, Mohammad Alshayeb, E Criswell, M Iyer, A McDowell, M McEniry, and K Regner. 2002. EVE: On-board process planning and execution. In *Earth Science Technology Conference, Pasadena, CA*, Vol. 39.
38. UC Irvine Machine Learning Repository. 1936. Iris Data Set . (1936). https://archive.ics.uci.edu/ml/datasets/Iris Accessed: December, 2015.
39. Ina Wagner. 1999. Ethical issues of healthcare in the information society. Opinion of the European Group on Ethics in Science and New Technologies No. 13, 30 July 1999. (1999). http://aei.pitt.edu/43228/
40. EMC Digital Universe with Research & Analysis by IDC. 2014. *The Digital Universe of Opportunities: Rich Data and the Increasing Value of the Internet of Thingss*. Technical Report. http://www.emc.com/leadership/digital-universe/2014iview/digital-universe-of-opportunities-vernon-turner.htm
41. WSO2. 2015. WSO2 Complex Event Processor. (2015). http://wso2.com/products/complex-event-processor/
42. Indre Zliobaite, Albert Bifet, Bernhard Pfahringer, and Graham Holmes. 2014. Active learning with drifting streaming data. *Neural Networks and Learning Systems, IEEE Transactions on* 25, 1 (2014), 27–39.