MLaaS: Machine Learning as a Service

Mauro Ribeiro, Katarina Grolinger, Miriam A.M. Capretz
Department of Electrical and Computer Engineering
Western University, London, Ontario, Canada N6A 5B9
{mribeir5, kgroling, mcapretz}@uwo.ca

***Abstract*—The demand for knowledge extraction has been increasing. With the growing amount of data being generated by global data sources (e.g., social media and mobile apps) and the popularization of context-specific data (e.g., the Internet of Things), companies and researchers need to connect all these data and extract valuable information. Machine learning has been gaining much attention in data mining, leveraging the birth of new solutions. This paper proposes an architecture to create a flexible and scalable machine learning as a service. An open source solution was implemented and presented. As a case study, a forecast of electricity demand was generated using real-world sensor and weather data by running different algorithms at the same time.**

***Keywords*—Machine Learning as a Service, Supervised Learning, Regression, Prediction, Service Oriented Architecture, Service Component Architecture, Platform as a Service**

I. Introduction

The amount of data generated has been continuously growing from global data sources like Web sites, social media, mobile applications, news networks, weather, political institutes, society and the economy. No matter how big the data are, they may be useless without proper preparation and processing. Many different machine learning algorithms have been used to extract valuable knowledge from data, e.g., for scientific modeling, consumer behavior, energy consumption forecasting, related article recommendation and user trends.

At the same time, with the popularization of sensors and mobile devices able to connect to a network (e.g., the Internet of Things), it is becoming viable to collect more data from specific contexts at higher levels of detail. By connecting global and context specific data, it is possible to extract even more detailed information and build richer knowledge using machine learning algorithms.

Large companies have enough resources to invest in their own machine learning solutions. However, small companies, developers and researchers in general have difficulties when facing the steep learning curve of how machine learning works and when building their own solutions or integrating with third-party ones. In addition, machine learning can require computational resources with impracticable costs. How could these users have access to affordable machine learning services?

One way to meet this demand is by creating a functional and ready-to-use Machine Learning as a Service (MLaaS) platform. Because multiple users will be using the same platform, computational resources can be shared or allocated on demand, reducing overall costs. By specifying a well defined interface, users can have access to machine learning process efficiently from anywhere, at any time. Users must not be concerned with implementation and computing resources, focusing mainly on the data itself.

This paper proposes a novel approach for machine learning, providing a scalable, flexible, and non-blocking platform as a service based on the service component architecture. This platform facilitates the creation, validation and execution of machine learning models. By taking advantage from service oriented architecture, the proposed approach becomes easily scalable and easy to adapt by adding, removing, changing and linking any component. This also makes the system more flexible for handling multiple data sources and different machine learning algorithms at the same time. In addition, a graphical user interface is presented to facilitate the comparison between different models.

The proposed framework source code is available[1] as an open-source project to facilitate its use for various prediction modeling tasks and to enable it to be adapted for other purposes.

The following sections of this paper are organized as follows: Section II gives an overview of machine learning, service component architecture and the main related works on machine learning as a service; Section III describes the proposed architecture for MLaaS; Section IV explains the MLaaS process; Section V presents the case study; and finally, Section VI concludes the paper.

II. Related Works

A. Machine Learning

Machine Learning is one of the fastest growing fields in computer science [1]. It is a collection of statistical techniques for building mathematical models that can make inferences from data samples (known as a training set). Machine learning is a part of artificial intelligence: it must adapt itself to a changing environment.

Figure 1 roughly illustrates how to choose between the main categories of machine learning. There are three main types of learning [1]: (a) *Supervised Learning*, when the training set is labeled (i.e., it contains the attribute that the model is trying to estimate); (b) *Unsupervised Learning*, when the training set is not labeled, and (c) *Reinforced Learning*, when the learned results lead to actions that change the environment.

[1] https://github.com/mauro0x52/mlaas

M. Ribeiro, K. Grolinger, M.A.M Capretz, MLaaS: Machine Learning as a Service, International Conference on Machine Learning and Applications, 2015.

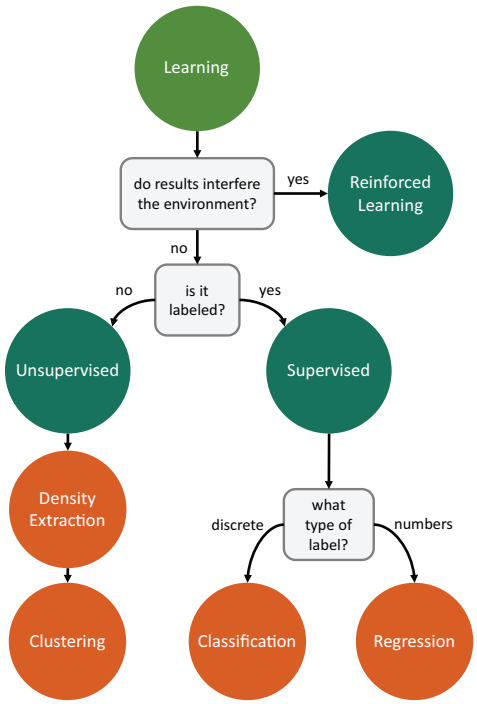

Fig. 1: Machine learning methods categorization

The labels in supervised learning can be discrete or continuous, which are handled by *classification* and *regression* algorithms respectively. Classification is used mostly for prediction, pattern recognition and outlier detection, whereas regression is used for prediction and ranking. Unsupervised learning is known as *density estimation* in statistics and is represented mainly by *clustering* algorithms. Classification, regression and clustering are widely used in data mining (applications of machine learning to large databases), whereas reinforced learning is mostly used in decision-making problems (e.g., a computer playing chess).

Independently of the applications just described, machine learning techniques work in a similar way: the model learns from a training set and then becomes able to make inferences for a new data set. This abstraction inspires the creation of a generic architecture to support any machine learning algorithm. This paper will focus on regression predictive modeling, although the approach can be adapted for other algorithms.

In predictive modeling, once rules have been extracted from past data (the training set), the model can make accurate prediction for new instances of data (the predictor set) if the future is similar to the past. Spam filtering, investment risk and energy consumption forecasting are some examples of predictive modeling. Predictive modeling approaches include: Artificial Neural Networks for energy consumption [2], Support Vector Machines for energy consumption [2] and K-Nearest Neighbors for wind power [3].

Validation for predictive models has a twofold importance: (a) choosing the most accurate algorithm and parameters; and (b) estimating the expected error for new predictions [1]. Accuracy can be related with errors, which can be calculated by comparing the estimated results from the model with the real measured results. A popular and reliable validation technique for predictive models is the K-Fold Cross-Validation. The data set is split randomly into K parts of the same size. One of the K folds is used to calculate the errors using the other K-1 folds to train the algorithm. The same process is repeated K times each time using different fold for validation. This method guarantees that the entire data set is validated with statistical significance.

Different models can perform better or worse, depending on the used algorithms, parameters and data set. However, there is no such a thing as the best learning algorithm [1]. For any algorithm, there are data sets that perform very accurately and others that perform very poorly. For the same data set, different algorithms can perform differently because of their own nature. MLaaS helps the user to run multiple algorithms and compare their performances, so the most suitable algorithm can be chosen.

B. Service Component Architecture

A service component architecture (SCA) [4] is a modeling specification for composing systems according to the principles of Service-Oriented Architecture (SOA).

SCA separates implementation concerns into three artifacts: (a) **components** implement its business function; (b) **composites** assemble various components together to create business solutions, and (c) **services** create an interface for remote access to component and composite functions. In a system, composites, services, and their relations with components are defined in a dynamic XML descriptor file.

Because SCA is built on top of SOA, it inherits all SOA's advantages — for example, intrinsic interoperability, inherent reuse, simplified architecture and solutions, and organizational agility [5]. In addition, whereas SOA focuses on building an architecture to design individual components, SCA focuses on assembling multiple components into a composite and facilitating design, implementation, and deployment. SCA systems have been successfully used, for example, in geographic information systems [6] and smart home systems [7] [8].

This research aims to build a platform which is capable of providing various machine learning algorithms to build different predictive models which will run at the same time. Adding a new algorithm must be simple. The system must provide well-defined APIs which can be remotely accessed over the Web by any external system. SCA provides enough artifacts to meet these requirements.

C. Machine Learning as a Service (MLaaS)

The increasing demand for machine learning is leveraging the emergence of new solutions. In this section, various machine learning platforms are reviewed.

PredictionIO [9] was launched in 2013. It is an open-source platform with an architecture that integrates multiple machine learning processes into a distributed and horizontally scalable

system based on Hadoop. In addition, PredictionIO provides access through web APIs and graphical user interface (GUI).

Baldominos et al. [10] also proposed a platform built on top of Hadoop. Its implementation was capable of handling up to 30 requests at one time while maintaining a response time of less than one second.

OpenCPU [11] is another open-source platform, launched in 2014, that creates a Web API for R [12], a popular statistical analysis software environment. However, because it is practically a middleware for accessing R functions, it does not take into account many non-functional requirements like scalability and performance.

In the industry context, Google, Microsoft, and Amazon have been releasing their own proprietary platforms. Google released its Prediction API[2] in 2014. Also in 2014, Microsoft launched Azure Machine Learning[3]. Most recently in 2015, Amazon released AWS Machine Learning[4]. Their sales can prove that the demand exists. Unfortunately, the designs and implementation specifications of these products are not publicly available.

PredictionIO, OpenCPU, and Baldominos' platforms are built on top of a specific analytical tools and suffer from its restrictions. This means less flexibility for adding new machine learning algorithms, for data storage, and for deployment. Although Hadoop and R are open-source projects, it is not a trivial challenge to adapt them to a new approach. The same happens with the industry players and their proprietary solutions when external developers cannot have access to the code to add new algorithms.

The MLaaS proposed in this paper focuses on predictive modeling. As an architecture based on SCA specifications, the architecture facilitates the addition of new algorithms, its improvement, and its adaptation to other machine learning applications. Even the revised platforms mentioned above can be attached to proposed architecture to build prediction models.

III. Architectural Design

This section describes the proposed MLaaS architecture, which is designed to support machine learning by gathering data from multiple sources and building multiple models using different algorithms. The approach focuses on predictive modeling, but it is adaptable to other applications.

The scope of this architecture deals with the machine learning itself, ignoring the front-end aspects such as the user interface. In a Model-View-Controller (MVC) perspective, this architecture focus on the model layer while the controller and view layers are only implemented as part of the case study.

The SCA diagram in Figure 2 depicts a high level overview of the architecture.

The *Modeler* composite is responsible for building new predictive models. A predictive model is an instance of *Model-μ* composite, running a specific algorithm. The cardinality 0..N shows that MLaaS can run multiples instances of *Model-μ* composite at the same time, through the *Build*, *Train*, *Test* and *Predict* services. The *model* property shows that each instance can run with different settings.

The architecture works as follows: the *Machine Learning as a Service* composite receives raw data from data sources through its *Send Training Set* service. First, data are received and prepared by the *Data Gatherer* composite. The *Modeler* composite then receives the prepared data to train a *Model-μ* instance. When receiving a predictor set from the *Send Predictor Set* service, the *Model-μ* instance calculates the prediction and serves it to external modules through the *Get Prediction* service.

The specified services provide well defined interfaces that increase the architecture's flexibility to new inputs and outputs: the *Send Training Set* and *Send Predictor Set* services enable the inclusion of various data sources that will be merged by *Data Gatherer*; the *Build*, *Train*, *Test* and *Predict* consumers enable the architecture to be pluggable with different *Model-μ* instances; and the *Get Report*, *Get Test* and *Get Prediction* services enable different user interfaces and external systems to consume the data.

The following subsections describe each of the composites shown in Figures 2 and 3.

A. Data Gatherer Composite

The *Data Gatherer* composite is responsible for receiving data, pre-processing it, and feeding it to the model. One instance is created for each *Send Training Set*, *Send Test Set* or *Send Predictor Set* services, so that they can run in parallel and independently. The *Data Gatherer* composite is made up of three components arranged in a pipeline as illustrated in Figure 3; they can be described as follows:

- The *Merger* component merges all received data (single data points or batches) from different data sources (e.g., sensors or databases). Data sets with different schema are joined into a single multicolumn schema by related attributes (e.g., time-stamp for time-series data, categories, identifiers, etc). When finished, it forwards the data to the *Outliers Remover* component.
- The *Outliers Remover* component removes outliers (e.g., missing values, zeros, extremely high values, etc.). Once finished, it forwards the cleaned data to the *Pre-Processor* component.
- The *Pre-Processor* component modifies the data set by re-sampling, creating columns, getting the maximum, minimum, or average values, etc. When it is finished, it sends the pre-processed data to the destination component in the *Modeler* composite.

B. Modeler Composite

This is the core composite in the architecture, because it is responsible for building, training, testing, and running the *Model-μ* instances. It is made up of five components as illustrated in Figure 2, which can be described as follows:

[2]https://cloud.google.com/predictio

[3]http://azure.microsoft.com/en-us/services/machine-learning

[4]http://aws.amazon.com/pt/machine-learning

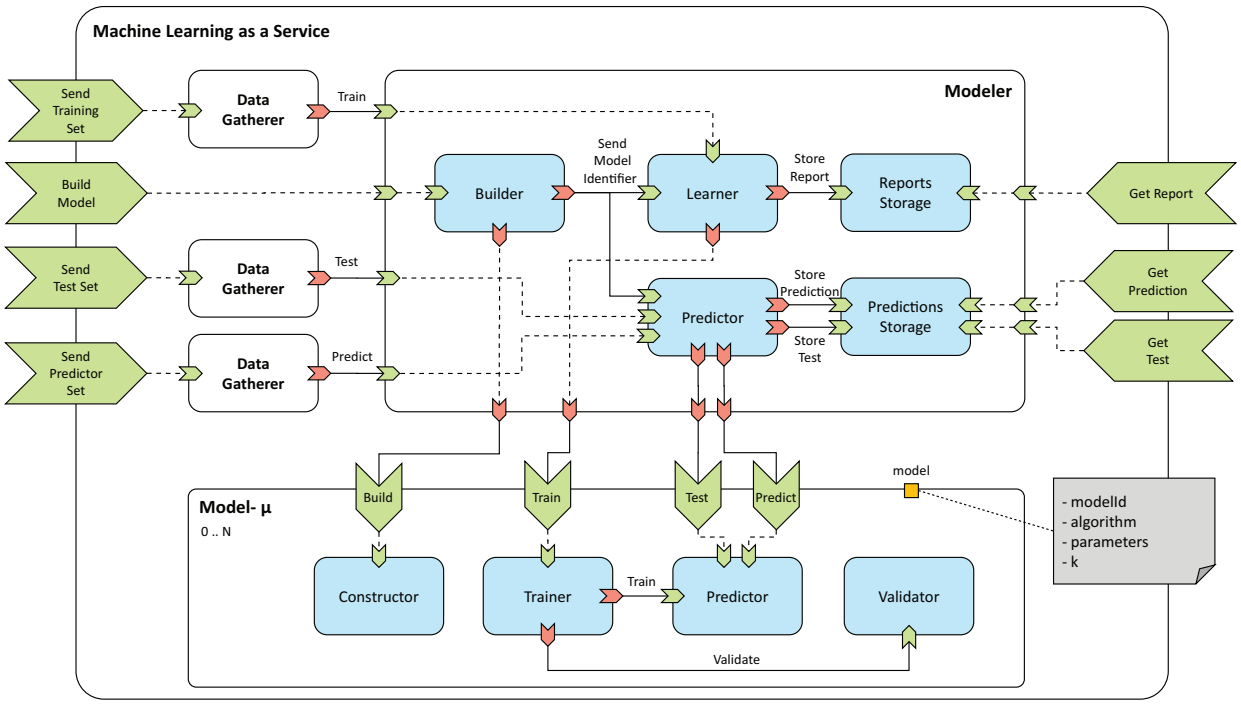

Fig. 2: MLaaS architecture using SCA notation

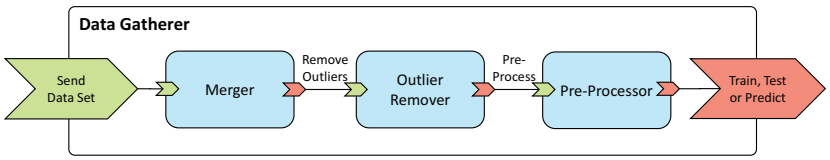

Fig. 3: Data Gatherer composite

- The *Builder* component receives from *Build Model* the parameters (e.g., algorithm and property values) to build and deploy a new model (a *Model-*μ instance) for the *Build* consumer. When the instance is created, *Builder* sends the model identifier back to the consumer and forwards it to the *Learner* and *Predictor* components.
- The *Learner* component receives the pre-processed data from the *Train* service and forwards them to the destined *Model-*μ instance. When it receives the training report from the *Model-*μ instance through the *Train* consumer callback, it forwards it to *Reports Storage*.
- The *Reports Storage* component receives the report from the *Learner* component through the *Store Report* service and serves it to external consumers through the *Get Report* service.
- The *Predictor* component receives the predictor set from the *Predict* service and forwards it to the *Model* through the *Predict* consumer, which will return the prediction through a callback. The prediction will be returned to the *Predict* requester and also forwarded to *Predictions Storage*. *Predictor* is also responsible for forwarding the testing set.
- The *Predictions Storage* component receives and stores the predictions and tests from the *Store Prediction* and *Store Test* services and provides them to external consumers through the *Get Prediction* and *Get Test* services.

C. Model-μ Composite

The *Model-*μ composite is an architecture for building different models. It holds all the implemented algorithms source codes (e.g., Multilayer Perceptron), but only one must be loaded. The algorithm to be loaded and its parameters should be specified when calling the *Build* service. In other words, for each *Build Model* service request, a new instance of a *Model-*μ composite is created.

The *model* property describes how the model needs to be built and executed. It is composed of four sub-properties: *modelId*: is the model unique identifier, *algorithm*: specifies which algorithm is going to be used by the model, *parameters*: adjust the algorithm behavior, and *k*: the number of folds to use in the K-Fold Cross-Validation.

The *Train*, *Test*, and *Predict* service specifications enable the *Modeler* composite to interact with any *Model-*μ instance.

The *Model-*μ composite is made up of four components, which can be described as follows:

- The *Constructor* component is responsible for loading the right algorithm and setting the properties of the model in-

stance using the *Build* service request parameters. When the instance is set up and running, it is ready to provide *Train*, *Test* and *Predict* services.

- The *Trainer* component receives the training set from the *Train* service and forwards it to *Validator* and *Predictor* components through the *Validate* and *Train* services respectively. When validation is finished, the *Trainer* component receives the validation report from the *Validate* service callback and returns it to the consumer through *Train* service callback.
- The *Validator* component receives the training set from the *Validate* service, feeds it to the model and validates the model (e.g., K-Fold Cross-Validation), returning a report.
- The *Predictor* component receives the training set from the *Train* service to feed the model for future prediction requests. When receiving predictor sets through the *Predict* service, it calculates and returns the predictions.

The implemented algorithms source code must be responsible only for training and predicting. Testing and validating do not depend on the algorithm itself, but on the results, which can be found by using the algorithm's training and predicting functions. Therefore, testing and validating functions are responsibilities of *Validator* and *Predictor* components, increasing standardization and reducing the effort when adding a new algorithm.

IV. MLaaS Process

The diagram in Figure 4 illustrates the main interaction flow between the *Consumer*, the *Modeler* and the *Model-μ* composites. To simplify, the earlier stage related to the *Data Gatherer* composite is ignored by assuming that data have already been pre-processed. The term *Consumer* in the following discussion refers to a generic consumer using the *Modeler* component.

The main flow is divided into three stages:

- *Building*: it starts with the *Consumer* requesting the *Builder* component to build a new model through the *Build Model* service. The *Builder* component will then create and configure a new *Model-μ* instance. When the building operation is complete, the *Builder* component sends the new model identifier to the *Learner* and *Predictor* components and to the *Consumer*.
- *Training*: the *Consumer* is now able to train the instantiated model. It sends the already pre-processed training set to the *Learner* component through the *Train* service, which will forward the training set to the *Trainer* component of the *Model-μ* instance. The *Trainer* component will make two requests at the same time: one to the *Validator* component to validate the model (e.g., K-Fold Cross-Validation) and another to the *Predictor* component to be trained for future prediction requests. When validation is complete, the *Validator* component responds to *Trainer* component with the validation report, which contains information such as error measurements. The report will be stored into *Reports Storage* component for future retrievals.
- *Predicting*: the model is ready to predict. The *Consumer* sends the predictor set to the *Modeler* composite's *Predictor* component, which will forward to the *Model-μ* instance's *Predictor* component, where the prediction is calculated and returned to the *Modeler*. The predictions are sent to the *Predictions Storage* to be stored and served.

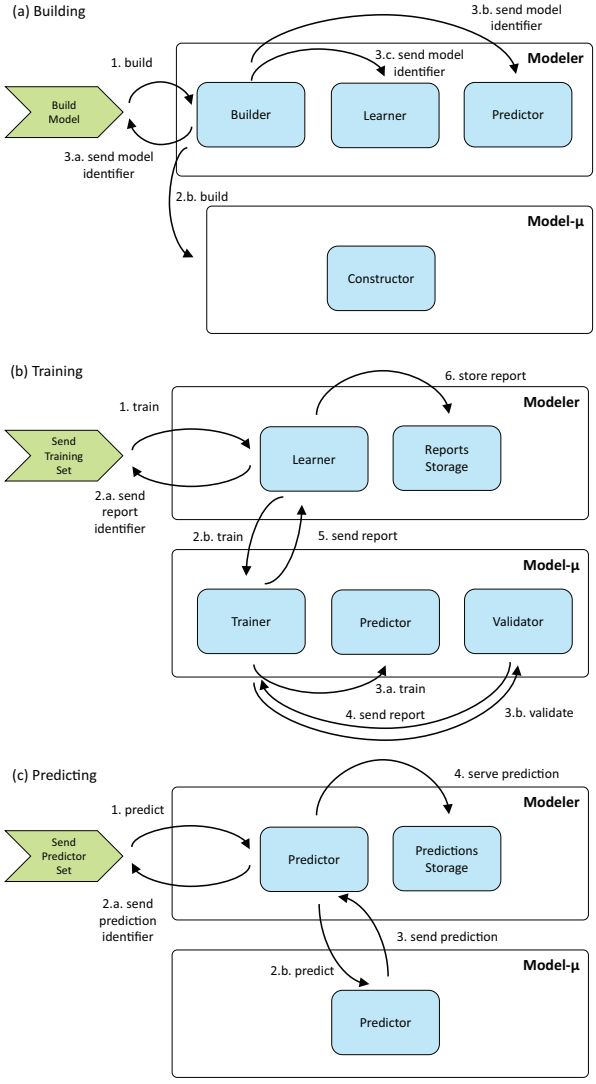

Fig. 4: Communication flow for the three machine learning stages

In *Training* and *Predicting* stages, the *Consumer* receives the report and prediction identifiers as soon as the *Learner* and *Predictor* components receive the request, so it is not necessary to keep the connection while the entire request is be processed. When the report or prediction is ready, it can be accessed from *Reports Storage* and *Predictions Storage* components, using the specific identifier.

A *Training Stage* can also be considered and works similarly to the *Predicting Stage*. The main difference is the final result, which contains testing information such as errors.

V. Case Study

The goal of this case study is to forecast energy demand based on past electricity demand data data for an office building, using different machine learning algorithms and finding the best-performing one. This experiment focuses mainly on the *Modeler* and *Model-μ* composites.

The proposed architecture was implemented using electricity demand data from Powersmiths' office building, in Brampton, ON, Canada. The data set were pre-processed before feeding them to the system. This data set was made up of 13 daily attributes: the energy demand peak, six weather attributes and six time attributes. The six weather attributes were: maximum temperature, minimum temperature, average temperature, maximum humidity, minimum humidity and average humidity. The six time attributes were: year, month (from 1 to 12), day of the month (from 1 to 31), day of the year (from 0 to 365), weekDay (from 0 for Sunday to 6 for Saturday) and dayType (0 for a business day, 1 for a weekend and 2 for a holiday).

The system was built using Node.js because of its ease and agility for coding and deploying Web services and handling JSON. Because there are currently no SCA frameworks for Node.js, one had to be implemented. JSON was used for Web service communication, data storage and the SCA artifact descriptor file. A simple user interface was developed to generate effective illustrations of the results obtained.

The source code is available in a public repository [5].

A. Algorithms

To evaluate the architectural flexibility of running different machine learning models at the same time, *Model-μ* composite was implemented to support the following algorithms:

- *Multi-Layer Perceptron (MLP)*: one of the most used techniques when evaluating machine learning models, and one of the most used for electrical consumption problems [2]. It was implemented using the Synaptic package[6].
- *Support Vector Regression (SVR)*: also one of the most used techniques for electrical consumption problems [2]. It was implemented using the Node-SVM package[7].
- *K-Nearest Neighbors (KNN)*: easy to understand, to code, and to debug. This algorithm was coded for this experiment.

A generic *Algorithm* class was coded under object-oriented programming structure, defining the standard interface for train and predict function calls. A new algorithm can be implemented simply by inheriting the *Algorithm* class and making minor adaptations. In this case study, the *KNN Algorithm* class was implemented first to test and validate the *Model-μ* composite. Later, using the same code structure, *MLP Algorithm* and *SVR Algorithm* classes were coded and imported into *Model-μ* composite.

[5]https://github.com/mauro0x52/mlaas

[6]http://synaptic.juancazala.com

[7]https://github.com/nicolaspanel/node-svm

When a *Model-μ* instance is built, the algorithm with the parameters (both specified in the *model* property) is loaded.

The test and validate functions are performed by *Predictor* and *Validation* components respectively, and not by the *Algorithm* class. Both functions use the results from *Algorithm*'s train and predict calls.

The *Validator* component implements de K-Fold Cross-Validation method to validate the model, calculating the mean absolute errors and the mean square errors. The number of folds K can be defined to the *model* property when building a new model.

The architectural design and the dynamic artifacts descriptor file make it possible to create new *Model-μ* instances dynamically. After the new *Model-μ* instance is deployed and the artifacts descriptor file is updated, the new *Model-μ* instance will be available without the need to recompile or restart the system.

B. Results

Three different models were created by instantiating the *Model-μ* composite. Table I shows the parameters used for each model. The models were requested to predict using a test set, which contains all the 13 attributes including the real measured daily electricity demand peaks. For the K-Fold Cross-Validation, K = 10 was fixed for all the models. The models were also requested to run a prediction using a different predictor set.

Figure 5 shows a screenshot of the MLaaS graphical user interface (GUI). Through the navigation bar, the user can access models (list, create and remove), train, test and predict models and consult a graphical summary of the results. The first row of charts shows the validation performance, with three graphics showing the mean absolute errors, mean square errors, and the execution time for each of the three models. The second row shows the test performance, comparing the mean absolute errors, mean square errors, and execution time for the three models. The third row is a chart comparing the three models' test results with the real measured data from the test set. Finally, the last row shows the results of a prediction.

TABLE I: Model Parameters

Algorithm	Parameter	Value
KNN	k	10
	max distance	2
MLP	nodes per layer	12, 14, 1
	learning rate	0.1
	max iterations	1000
	min error	0.0001
SVR	gamma	0.125, 0.5, 1
	c	8, 16, 32
	epsilon	0.001, 0.125, 0.5
	retained variance	0.995

Fig. 5: MLaaS screenshot comparing KNN, MLP, and SVR.

The SVR model showed better accuracy – it had the lowest mean absolute errors and mean square errors – both in validation and in testing. Although the KNN model had better accuracy in validation than the MLP model, it had the worse mean square error in testing.

The KNN model performed much faster during validation and could finish executing even while the SVR and MLP models were still running. The SVR model finished the validation last. On the other hand, during testing, MLP model finished first and KNN model was the last. In other words, one model's processing did not block the CPU as it would have on a single-threaded server.

VI. Conclusions

With the growing amount of data available, companies and researchers are demanding feasible and affordable ways to extract knowledge from all this data. This paper has presented a novel architecture for a scalable, flexible, and non-blocking machine learning as a service based on SCA and focusing on predictive modeling. The proposed architecture can support multiple data sources and create various models with different algorithms, parameters, and training sets.

To prove the concept, the system was built to predict electricity demand using real-world data. Once the main architecture is working and at least one algorithm coded, it is simple to implement other algorithms. It is possible to execute multiple models concurrently.

For future research, MLaaS can be adapted to machine learning applications other than predictive modeling, for example, pattern recognition, outlier detection, ranking and clustering.

Acknowledge

This research was supported in part by an NSERC CRD at Western University (CRDPJ 453294-13). Additionally, the authors would like to acknowledge the support provided by Powersmiths.

References

[1] E. Alpaydin, *Introduction to machine learning*. MIT press, 2014.

[2] a. S. Ahmad, M. Y. Hassan, M. P. Abdullah, H. a. Rahman, F. Hussin, H. Abdullah, and R. Saidur, "A review on applications of ANN and SVM for building electrical energy consumption forecasting," *Renewable and Sustainable Energy Reviews*, vol. 33, pp. 102–109, 2014. [Online]. Available: http://dx.doi.org/10.1016/j.rser.2014.01.069

[3] M. Yesilbudak, S. Sagiroglu, and I. Colak, "A new approach to very short term wind speed prediction using k-nearest neighbor classification," *Energy Conversion and Management*, vol. 69, pp. 77–86, 2013.

[4] Service Component Architecture Assembly Model Specification Version 1.1. Accessed: 30-04-2015. [Online]. Available: http://docs.oasis-open.org/opencsa/sca-assembly/sca-assembly-1.1-spec-cd03.html

[5] T. Erl, *Service-Oriented Architecture: Concepts, Technology, and Design*. Pearson Education India, 2005.

[6] F.-C. Lin, L.-K. Chung, W.-Y. Ku, L.-R. Chu, and T.-Y. Chou, "Service component architecture for geographic information system in cloud computing infrastructure," in *Advanced Information Networking and Applications (AINA), 2013 IEEE 27th International Conference on*. IEEE, 2013, pp. 368–373.

[7] T. Calmant, J. C. Américo, D. Donsez, and O. Gattaz, "A dynamic sca-based system for smart homes and offices," in *Service-Oriented Computing-ICSOC 2012 Workshops*. Springer, 2013, pp. 435–438.

[8] C.-C. Lo, D.-Y. Chen, and K.-M. Chao, "Dynamic data driven smart home system based on a service component architecture," in *Computer Supported Cooperative Work in Design (CSCWD), 2010 14th International Conference on*. IEEE, 2010, pp. 473–478.

[9] S. Chan, T. Stone, K. P. Szeto, and K. H. Chan, "PredictionIO: a distributed machine learning server for practical software development," in *Proceedings of the 22nd ACM international conference on Conference on information & knowledge management*. ACM, 2013, pp. 2493–2496.

[10] A. Baldominos, E. Albacete, Y. Saez, and P. Isasi, "A scalable machine learning online service for big data real-time analysis," in *Computational Intelligence in Big Data (CIBD), 2014 IEEE Symposium on*. IEEE, 2014, pp. 1–8.

[11] J. Ooms, "The OpenCPU System: Towards a Universal Interface for Scientific Computing through Separation of Concerns," *arXiv:1406.4806*, no. 2000, pp. 1–23, 2014. [Online]. Available: http://arxiv.org/abs/1406.4806

[12] R Core Team, *R: A Language and Environment for Statistical Computing*, R Foundation for Statistical Computing, Vienna, Austria, 2013, ISBN 3-900051-07-0. [Online]. Available: http://www.R-project.org/