2022 EC³
2022 European Conference on Computing in Construction
Ixia, Rhodes, Greece
July 24-26, 2022

PRENERGET: A FRAMEWORK FOR THE INCLUSION AND ADAPTATION STRATEGIES OF MACHINE LEARNING MODELS FOR ENERGY DEMAND FORECASTING IN BUILDINGS

Iker Esnaola-Gonzalez[1], Meritxell Gomez-Omella[1], Susana Ferreiro[1], Alvaro García[1], Francisco Javier Díez[1]
[1]TEKNIKER, Basque Research and Technology Alliance (BRTA), Spain

Abstract

Machine Learning (ML) models are key enablers for the implementation of different energy-efficiency strategies in buildings. There are a variety of frameworks that facilitate the development of ML models, but it is necessary to move into a different environment for their deployment and exploitation. Furthermore, their performance tends to degrade over time. Consequently, they need to be regularly evaluated and upgraded to ensure the robustness of the overall solution. The seamless exploitation, adaptation and evolution of ML models is still an open issue nowadays, and in this article, a software framework called PRENERGET aimed at addressing this issue is presented. The main contributions of PRENERGET are, on the one hand, the facilitation of the exploitation of ML models to make forecasts related to energy efficiency, and on the other, the maintenance and, if possible, the improvement of the forecasting performance over time.

Introduction

Nowadays, the building sector's energy consumption accounts for almost a third of the total energy consumption and its share of emissions has risen to almost 30% (International Energy Agency 2021). As a matter of fact, the energy consumed in the building sector is responsible for nearly 3 Gt of direct CO_2 emissions. Space heating, cooking and other daily activities account for the majority of global CO_2 emissions today in the buildings sector (International Energy Agency 2021), and the demand side management (DSM) and demand response (DR) programs have emerged in an effort to minimise these figures (Warren 2014, Albadi & El-Saadany 2007). However, the implementation of DR programs is not straightforward (Esnaola-Gonzalez et al. 2021), and being able to accurately forecast the energy demand plays a crucial role.

Artificial intelligence (AI) systems, and more precisely, Machine Learning (ML) based models have showcased their prominence in creating accurate predictions (Gómez-Omella et al. 2021, Tascikaraoglu & Sanandaji 2016, Lusis et al. 2017). There are a variety of environments and frameworks that facilitate the development of ML models, but when it comes to their deployment and exploitation, it is necessary to move into a different environment. Certainly, the value offered by the models is more often than not limited by the use of inappropriate application logic. Furthermore, under normal conditions, ML-based forecasting models performance degrade over time due to a change in the environment that violates the models assumptions (Widmer & Kubat 1996). Consequently, the deployed models need to be regularly evaluated and upgraded to ensure the robustness of the solution they are part of (Žliobaitė et al. 2016).

The seamless exploitation, adaptation and evolution of ML-based models is still an open issue nowadays, and in this article, a software framework called PRENERGET aimed at addressing this issue is presented. Namely, the main contributions of the developed framework are:

- To facilitate the exploitation of ML models to make forecasts related to energy efficiency.
- To maintain and, if possible, improve the forecasting performance over time.

The rest of the article is structured as follows. First, a summary of previous works found in the literature is presented in the Related Work Section. The development and deployment of PRENERGET is detailed in The Framework Section. Later, in the PRENERGET On the Loop Section, PRENERGET is validated in a real use case and the results obtained are shown. Finally, the conclusions obtained are summarised and future work is presented.

Related work

The AI is currently experiencing an upsurge that can be attributed to advances in computing and the increasing availability of data, and it has been useful for solving problems of different nature, including forecasting problems. In the field of energy efficiency in buildings, being able to accurately forecast future situations is key to ensure optimal decision-making. For example, forecasting the energy to be consumed and the energy to be produced by renewable systems installed in a building can contribute to maximising their energetic efficiency by implementing load curtailment (i.e., a reduction of electricity usage) or reallocation (i.e., a shift of energy usage to other off-peak periods) (Esnaola-Gonzalez et al. 2021).

Regarding the forecasting of energy coming from renewable sources, different approaches and mechanisms can be found in the literature, including ML algorithms. As a matter of fact, they have been proven to perform well when forecasting energy to be produced by photovoltaic panels (Ahmed et al. 2020, VanDeventer et al. 2019), solar thermal collectors (Unterberger et al. 2021) or even wind farms (Juban et al. 2007). Likewise, for the forecasting of buildings' energy consumption, different ML algorithms have been demonstrated to be valid. In (Zhang et al. 2018), a support vector regression modelling approach has been used for forecasting households electricity consumption, both to daily and hourly data granularity. But other algorithms such as regression trees and neural networks have

also been used in day-ahead load forecasting for residential customers problems (Lusis et al. 2017).

There are numerous software applications, tools and IDEs (integrated development environment) such as R Studio, Weka or Rapidminer that offer many possibilities for creating and training ML models. However, they do not offer functionalities that go beyond their own environments, and in order to deploy the developed models, the integration with other software is necessary. This concept has been extended to the cloud, which offers many advantages when it comes to the deployment of applications, both in terms of ease and scalability. In this regard, products such as Google Vertex AI and Amazon SageMaker offer sets of tools that enable deploying models in their own cloud infrastructure. The problem arises when users want to deploy these models on their own infrastructure or integrate them with applications that use their infrastructure.

In addition, the rapidly changing environment where we live in leads to the degradation of forecasting models' performance. This fact can be evident when abrupt changes occur such as during the COVID-19 pandemic-related confinement, curfew and mobility restriction measures (Gomez-Omella et al. 2020), but there are also other subtle changes that may equally affect the performance. This is known as the concept drift problem, which means that the statistical properties of the target variable the model is trying to forecast, change over time in unforeseen ways (Lu et al. 2018). In the face of such changes, forecasting models need to be adapted (Gama et al. 2014). Understanding the effect of the drift on the ML performance and defining the most appropriate adaptation strategies to make them more robust is one of the first steps to be considered. As a matter of fact, depending on the type of change, different adaptation mechanisms may be implemented. The work presented by (Celik & Vanschoren 2021) proposes different adaptation strategies that start from an initial model trained at least once with an initial batch of data. The strategy to follow will not always be the same and it will depend on many factors (e.g., the nature of the data, the application domain, unexpected events, etc.). This strategy will be determined by the data analysis task who is in charge of understanding the behaviour of the derivative of the model, detecting it, and even anticipating it.

The monitoring of the forecasting models' performance on the one hand, and the implementation mechanisms of the model adaptation on the other, are repetitive tasks that have a cost in terms of personnel dedication that, in many cases, may not offer improvements and therefore added value. Automating the models' performance evaluation and adaptation as well as the deployment of the adapted models alleviates workers from this tedious task and achieves better results by potential possible manual errors. Furthermore, this automation will also facilitate the selection of strategies to be followed when adapting the models.

Google also offers the open-source product TensorFlow which allows integration into on premise infrastructure. This suite includes the Pusher model deployment tool as well as other tools for model evaluation and validation such as Evaluator and InfraValidator. In this case, the developer is limited to use these libraries to generate the models and will not be able to use other languages such as R. On the other hand, the automation of the updates will have to be implemented in programming code using the previous tools.

Clipper (D. Crankshaw 2017) has been developed at a higher level of abstraction so as not to depend on certain frameworks to build the model. This framework is modular and allows invoking models developed in Apache Spark, Scikit-Learn, Caffe or TensorFlow. Clipper facilitates the integration of models through a unified REST interface but lacks model updating capabilities. Data and Learning Hub for science (DLHub) (Zhuozhao Li 2021) is another framework for the development of ML models. One of its pillars is the use of a standard model invocation via the previously developed "funcX" (R. Chard 2019, 2020), function and other one the use of Docker based containers as the current work. It is an excellent environment with good model characterisation capabilities and good performance, but it is oriented to research environments and not to production environments where resources have different restrictions. Muthusamy et al. (2018) shows a layer that encapsulates the ML models to provide a microservice interface that can be exploited in business applications. In this case, the model developer must be able to implement the defined interface. This interface offers functionalities to detect data drift and KPIs on accuracy and therefore assesses the need for retraining, but the update of the models is left out of the scope of the work.

Data scientists are experts in creating the ML models that solve the aforementioned prediction problems but the deployment of these models into production for their exploitation is not as straightforward as it might be thought. As a matter of fact, when it comes to exploiting the models, they are confronted with different execution environments than those used for analysis. These environments may work with technologies that are beyond the scope of knowledge of data analysts. And here is where programmer analysts come into play. They are specialists in creating software to be exploited in a given environment, but their knowledge of ML is often insufficient. Therefore, collaboration between the two types of profiles is not straightforward due to the different nature of the environments in which each works.

A framework to unify the interfaces between both of them can significantly improve the interaction necessary for the joint development of the software needed to not only exploit the forecasting models but to also ensure their performance over time. To the extent of knowledge of authors, existing approaches do not cover these requirements (Simmhan et al. 2013, Choi et al. 2016). Therefore, the definition and implementation of the necessary infrastructure, mechanisms, channels, interfaces, and workflows to facilitate the automation, deployment, and execution

of ML models under the same software architecture to streamline the process is a necessity.

The Framework

In this section, the design and the development of PRENERGET is presented. PRENERGET is a software framework for the deployment of ML models and its automatic adaptation to potential changes. The software framework offers a pipeline and a set of interfaces to incorporate data from different sources; the invocation and execution of multiple ML models; the mechanisms for monitoring and estimating the error generated by the ML models; and the required elements for the evaluation of the model and its future adaptability. PRENERGET provides a modular architecture with all the advantages that this involves and includes consistency in development, reduced development time, and flexibility.

Development

PRENERGET provides the data flows and workflows that maps out the flow of information, the definition of the pipelines for the interconnection of the different blocks or functionalities and the workflow engines that orchestrate the execution of tasks and exchanges of data between them. It is based on the interoperability offered by a set of REST API services, used to design, and integrate application software on different platforms, and a set of standard interfaces with an execution environment based on the R programming language that allows the development of advanced ML functionalities. This makes it easily integrated with other software either locally or remotely.
PRENERGET includes the following functionalities:

- ***R Script execution***: The set of programs and functions implemented in R programming language. They are in charge of training, executing and evaluating the ML models. They also include the functions to implement the automatic adaptation strategy or model adjustment. These functions can change, vary or even be replaced by others at any time without affecting the rest of the infrastructure because PRENERGET provides a modular architecture.

- ***Task Scheduling***: The set of Web services implemented in Java that encompass a set of tasks that can be executed both periodically and on demand. Most of these calls use a client to communicate with an R server, where the scripts are executed.

- ***Data management***: The PRENERGET architecture uses two types of databases. The first type, called 'external databases', is used to acquire the data and to develop the initial ML model its future re-adaptation. Being a modular and flexible architecture, it can work for any type of database (e.g., relational, time series...). All it takes is to change the connector to the database and the architecture will continue maintaining the rest of the functionalities. The second type, called 'internal database', is a database that includes the information to manage the tasks, and metadata about the models. This database allows the traceability monitoring and the correct understanding and interpretation of the models.

Figure 1 shows an example of the data flow between the R Scripts and the Web service that handles task scheduling and data management. In the flow, the components that are executed in the Web service are coloured blue and those that are carried out in the R scripts are coloured yellow. The components with grey bands represent the main scripts or classes while the solid components indicate actions that are carried out within them.

Deployment

The functionalities described in the previous section are provided by 3 software components. These have been encapsulated in Docker (Merkel 2014) containers to ease deployment, as shown in Figure 2.

- ***R server***: Will contain a running instance of R engine, along with the scripts and models to be called. As R does not need compiling to be executed, scripts can be updated or added during run-time, without the need of stopping the software.

- ***Apache Tomcat***: The task scheduling and data management functionalities will be provided by a Web service developed in Java, hosted in a Apache Tomcat server. By publishing the functions in a Web based API interface testing and integration with other software becomes easier.

- ***Internal Database***: This component stores the data used by the Web service: models, variables, data sources and scheduled tasks. This data is for internal usage only; data used to feed the model such as historical knowledge is stored in databases external. They are used by the software but are not considered part of the software solution.

The specific actions of the scheduled tasks change between different types but they follow a similar approach. After retrieving information of the involved variables from the database, the task will prepare the call to the script, retrieving data from the associated data sources and setting up the parameters. The R engine will be called to execute a specific script with the given parameters. After the execution has finished, it will read the results and use them, storing them, notifying other software or using them as parameters of other scripts.

PRENERGET on the loop

For the automation of PRENERGET, the period of execution of the previously explained flow (Figure 1) is decided. In each of these executions, a different training data set will be chosen from an external database to develop a model.

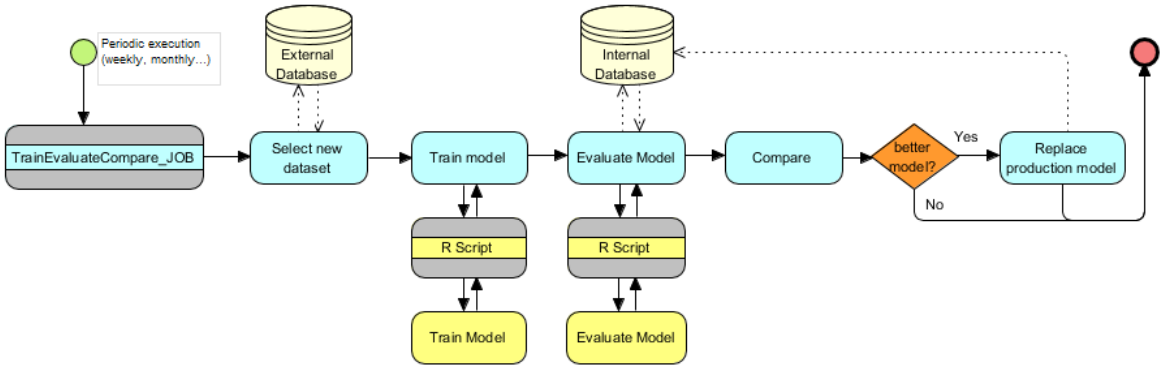

Figure 1: Data flow for model adaptation and updated

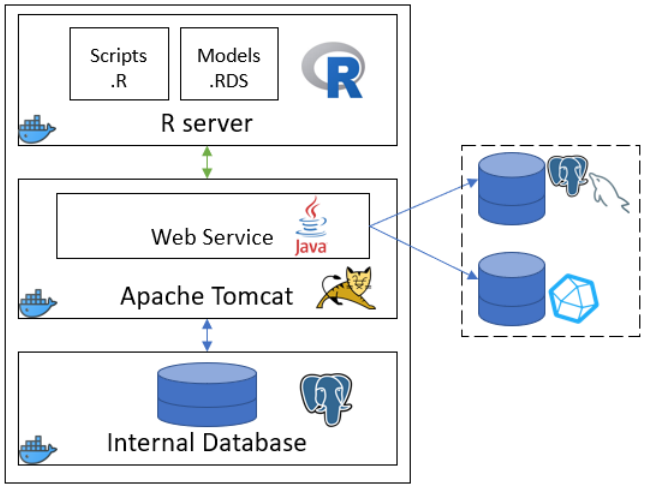

Figure 2: Software components architecture (with data sources)

Since the data is stored as time series over time, it is decided to use moving windows to update the values of the training phase. In this way, in each model fit, the oldest historical data is eliminated and the most recent is added to the training set. Afterwards, the training phase is started by calling an R script in charge of adjusting the model parameters using different techniques that allow generalisation, such as the commonly used k-fold cross validation. In the next phase, the evaluation of the model is done by calling another R script that obtained the errors or deviations of the training phase. These errors are calculated comparing the forecast values with the actual values registered available from the historical data. The update of the model in production will be based on the improvement of these error values that can be calculated using different metrics such as the Root Mean Squared Error (RMSE) or the Mean Absolute Error (MAE) among others. That is, in the last phase, the errors from the new and the old models are compared and if the new model improves the performance of the deployed model, it will be used to replace such a deployed model.

Being modular and configurable, PRENERGET allows programming different strategies, both for the testing and evaluation of the models, and for their update and adaptation to changes, in an agile and simple way.

Use cases

The feasibility of the PRENERGET framework has been tested in two real-world energy efficiency scenarios. On the one hand, in a neighbourhood in Madrid (Spain), and on the other, in a research centre in Eibar (Spain). For the sake of simplicity, this section will focus only on the latter. However, it is worth mentioning that the PRENERGET framework is designed to be applicable to other energy-related use cases.

The main objective of the forecast task was to provide accurate electric demand forecasts for the next day, so that adequate energy-saving strategies could be implemented in advance. Such forecast task has been performed using ML strategies based on the historical records containing hourly electric consumption data measured in kWh. So, before running the data flow of Figure 1, the forecasting algorithm, input variables, training method, and error metrics to be used should be set.

A previous comparative study of the results obtained from different ML algorithms concluded that the K-Nearest Neighbours (KNN) was the most efficient algorithm to provide further values in the problem at hand. The KNN was compared with an ARIMA model, a Linear Regression and a Support Vector Regression, obtaining better results both in forecast errors and in computational time. As a matter of fact, this method gives an accurate forecast due to the seasonality and the repetitions in the daily electric demand (Gómez-Omella et al. 2020).

After analysing different date and time related variables, the hour, the day, the month, the season of the year, the day of the week and a binary variable indicating whether the day is a working day or not, were the selected variables as they provided the highest correlations with the output variable. These support variables are needed to be used as inputs for the KNN algorithm, as the available data was a univariate time series containing the electricity consumed and a time index. As the KNN is an algorithm

that calculates distances between instances in order to decide the most similar neighbours, trigonometric transformations were made in cyclical variables. In other words, the hours and the month are modified to the sine and the cosine of their values to force their values to be equidistant, as explained in detail in Gomez-Omella et al. (2020).

Then, a 5-fold cross-validation technique was set to decide the optimal number of 'k' neighbours and the model was then fitted using a subset containing the 70% of the entire data available. This is a classic ML model training strategy that reserves 30% of the data to validate the decisions made in training task.

Finally, regarding the evaluation of the model accuracy, the RMSE was the metric chosen to compare the performance of the models. This value quantifies the mean error made in a forecast by averaging the squared errors made in all the estimated future points once their real values are known. The RMSE unit of measure is the same as the original data, making it intuitive to interpret.

Once the characteristics of the process have been configured for the specific problem, a first version of the KNN model was developed and deployed, and an automated task was programmed to execute the data flow from Figure 1 every day at 00:00h, that is, once every 24 hours.

In each iteration, the first phase consists in retrieving set of data to train the model. This set of data consists of historical energy consumption registries of the last 365 days stored in an external database. It is worth mentioning that, in this phase, a 24-hours rolling window has been set, so that in each iteration, the previous iteration's data set's first day is removed, and the last day is added to conform the new data set to train the model. The second phase consists in training the ML model. To do so, an R script is called which contains the functions to carry out the cross validation process and choose the optimal number of neighbours k to be used. Once the model is trained, in the next phase, it is evaluated by obtaining its RMSE. By calling an R script, the RMSE of the training phase is obtained and the score is stored in an internal database so that it can be retrieved at any given time. Furthermore, in this phase, the currently deployed model's RMSE is obtained in order to, in the next phase, compare it with the newest model's RMSE. As it can be seen in Figure 1, the model version update occurs in case the new model's RMSE is lower than the deployed model's RMSE. Every time a new model is created in each iteration, it is tagged with a different version code.

It starts with version 0.0 and then, version 0.1 is created. In case that version 0.1 was better than 0.0 and the current model needs to be replaced, version 0.1 will be automatically renamed to 1.0. Therefore, the model version coding convention is as follows. The models created in each iteration are labelled as minor versions x.1, x.2, x.3,... and the deployed ones as major versions 1.0, 2.0, 3.0,... If a given created model performs better than the deployed one, it is renamed as a major version. This approach allows to control the evolution of the model performance and to alert in case the values deviate from the acceptable boundaries.

Results

The process was initialised developing a model with a training set containing hourly electric consumption from 2020-03-30 to 2021-03-29, and an RMSE of 22.16 kWh. This model corresponds to the first version being deployed, so it is labelled as version 0.0. Once deployed, the first 24 hourly values forecast were corresponding to the next day, that is, 2021-03-30.

In the next iteration, the model 0.1 was trained with data from 2020-03-31 to 2021-03-30. Then, the RMSE obtained from the training of the deployed model 0.0 and the training of the new model 0.1 was compared. However, the new model 0.1's RMSE was higher, so the model 0.0 was not replaced and it continued to be active.

In the next iteration, a new model 0.2 was trained using data from 2020-04-01 to 2021-03-31 and the RMSE obtained was 21.58 kWh, less than that provided by the then-deployed model 0.0. For that reason, the model 0.2 is renamed as model 1.0 and it replaced the deployed model 0.0.

These iterations are repeated every 24 hours and the information of the models developed and deployed in the aforementioned scenario are summarised in Table 1, where the different updates of the versions of the model can be seen. Notice that the models that which are not deployed, are not included in that table.

In the first update, the RMSE of the model decreased from 22.16 kWh to 21.58 kWh in two iterations, that is forecast performance improved on average 0.58 kWh per day. Then, ten days later, the RMSE improve 0.30 kWh and that difference in error decreases in subsequent iterations. The rate improvement of the estimations depends on the data and the execution time, although the error is expected to reach a stable state. The process is still running and the model is continually being updated in order to provide estimates of further electric demand as accurate as possible.

It is expected that the better the model fits in forecasting historical data, the more accurate future forecasts will be. This can be different in case an unexpected sharp change in data statistical properties changed. That can be seen in Figure 3, where forecast values obtained with the different model versions are shown (to facilitate the visualisation, it was decided not to show the forecasts of version 0.0). As it can be observed, the estimated values and the actual values are increasingly similar as the version of the model is updated.

Conclusions

AI systems, and more precisely, ML-based models are key enablers for the implementation of different energy-efficiency strategies in buildings. However, their seamless exploitation, adaptation and evolution when they are deployed into production is still an open issue nowadays. In this article, a software framework called PRENERGET

Table 1: Snippet of the version update report of the deployed models

Version	Initial Training Date	Final Training Date	RMSE (kWh)	First Forecast Date
0.0	2020-03-30	2021-03-29	22.16	2021-03-30
1.0	2020-04-01	2021-03-31	21.58	2021-04-01
2.0	2020-04-11	2021-04-10	21.28	2021-04-11
3.0	2020-04-12	2021-04-11	21.25	2021-04-12
4.0	2020-04-14	2021-04-13	20.93	2021-04-14

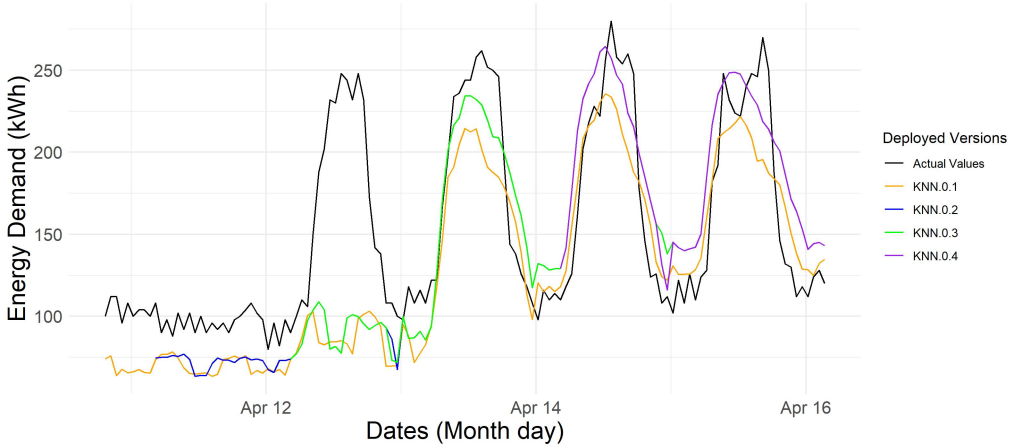

Figure 3: Evolution of the electric demand forecasts from the different versions of the deployed models

has been presented, aimed at facilitating the exploitation of ML models, and maintaining and if possible, improving their performance over time.

PRENERGET's modular architecture facilitates the deployment of forecasting models. Analysts can concentrate on developing the models in their environment based on R, or another programming language. People closer to systems management can implement them in a simple way as they only have to link the databases used and program when the invocations to the models will be made to obtain the desired predictions. In this way, it will be very easy to have several forecasters of different variables such as energy, temperature or occupancy applied to different facilities such as rooms, machines or entire buildings.

ML models are unable to keep pace in today's fast-changing world and their performance tends to degrade with time. Dealing with this issue and ensuring that deployed ML models provide operational results requires from an intensive effort of data scientists and ML experts. Even worse, in environments where many models are deployed, this can end up being an insurmountable barrier that hinders the successful deployment of an energy efficiency system. PRENERGET reduces costs, time and errors derived from human intervention in ML model performance maintenance and improvement tasks by automating it. Results show that in, a rather limited period, the performance of models can be improved up to 6%.

Future Work

After evaluating the work done, two possible points for future improvement are identified. On the one hand, in some cases, missing data were identified after the execution of the forecasts. Furthermore, when the connection was re-established and the values were captured again, the first value registered was the accumulated value of all the missing values. Therefore, the values provided by the systems were not realistic due to the number of values that were not correctly received. A function that identifies this kind of failures and impute the missing values with the most suitable method before the execution of the forecaster is left to further research. It is expected that the results obtained after this modification provide more accurate results. On the other hand, the implementation of a degradation control system of the final model is to be implemented. As mentioned, the internal model error is expected to stabilise over time and updates to production models will become less frequent. In these cases, the evolution of the errors in the predictions could be evaluated to avoid the concept drift of the deployed model.

Acknowledgements

This work is partly supported by the project 3KIA (KK-2020/00049), funded by the SPRI-Basque Government through the ELKARTEK program and the REACT project which has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no. 824395.

References

Ahmed, R., Sreeram, V., Mishra, Y. & Arif, M. (2020), 'A review and evaluation of the state-of-the-art in pv solar power forecasting: Techniques and optimization', *Renewable and Sustainable Energy Reviews* **124**, 109792.

Albadi, M. H. & El-Saadany, E. F. (2007), Demand response in electricity markets: An overview, *in* '2007 IEEE power engineering society general meeting', IEEE, pp. 1–5.

Celik, B. & Vanschoren, J. (2021), 'Adaptation strategies for automated machine learning on evolving data', *IEEE Transactions on Pattern Analysis and Machine Intelligence* **43**(9).
URL: *https://ieeexplore.ieee.org/document/9366792*

Choi, T.-M., Chan, H. K. & Yue, X. (2016), 'Recent development in big data analytics for business operations and risk management', *IEEE transactions on cybernetics* **47**(1), 81–92.

D. Crankshaw, X. Wang, G. Z. M. J. F. J. E. G. I. S. (2017), Clipper: A low-latency online prediction serving system, *in* '14th USENIX Symposium on Networked Systems Design and Implementation (NSDI)', pp. 613–627.

Esnaola-Gonzalez, I., Jelić, M., Pujić, D., Díez, F. & Tomasevic, N. (2021), 'An AI-Powered System for Residential Demand Response', *Electronics* **10**.

Gama, J., Žliobaitė, I., Bifet, A., Pechenizkiy, M. & Bouchachia, A. (2014), 'A survey on concept drift adaptation', *ACM computing surveys (CSUR)* **46**(4), 1–37.

Gomez-Omella, M., Esnaola-Gonzalez, I. & Ferreiro, S. (2020), Short-term forecasting methodology for energy demand in residential buildings and the impact of the covid-19 pandemic on forecasts, *in* 'Proceedings of 40th SGAI International Conference on Artificial Intelligence', Vol. 12498, pp. 227–240.

Gómez-Omella, M., Esnaola-Gonzalez, I. & Ferreiro, S. (2020), 'Short-term electric demand forecasting for the residential sector: Lessons learned from the respond h2020 project', *Proceedings* **65**(1).
URL: *https://www.mdpi.com/2504-3900/65/1/24*

Gómez-Omella, M., Esnaola-Gonzalez, I., Ferreiro, S. & Sierra, B. (2021), 'k-nearest patterns for electrical demand forecasting in residential and small commercial buildings', *Energy and Buildings* **253**, 111396.

International Energy Agency (2021), 'World energy outlook 2021'.
URL: *https://www.iea.org/reports/world-energy-outlook-2021*

Juban, J., Siebert, N. & Kariniotakis, G. N. (2007), Probabilistic short-term wind power forecasting for the optimal management of wind generation, *in* '2007 IEEE Lausanne Power Tech', IEEE, pp. 683–688.

Lu, J., Liu, A., Dong, F., Gu, F., Gama, J. & Zhang, G. (2018), 'Learning under concept drift: A review', *IEEE Transactions on Knowledge and Data Engineering* **31**(12), 2346–2363.

Lusis, P., Khalilpour, K. R., Andrew, L. & Liebman, A. (2017), 'Short-term residential load forecasting: Impact of calendar effects and forecast granularity', *Applied Energy* **205**, 654–669.

Merkel, D. (2014), 'Docker: lightweight linux containers for consistent development and deployment', *Linux journal* **2014**(239), 2.

Muthusamy, V., Slominski, A. & Isahagian, V. (2018), Towards enterprise-ready ai deployments minimizing the risk of consuming ai models in business applications, pp. 108–109.

R. Chard, T. J. Skluzacek, Z. L. Y. B. A. W. B. B. S. T. I. F. K. C. (2019), 'High performance function as a service for science', *Serverless supercomputing* .

R. Chard, Y. Babuji, Z. L. T. S. A. W. B. B. I. F. K. C. (2020), funcx: A federated function serving fabric for science, *in* 'Proceedings of the 29th International Symposium on High-Performance Parallel and Distributed Computing'.

Simmhan, Y., Aman, S., Kumbhare, A., Liu, R., Stevens, S., Zhou, Q. & Prasanna, V. (2013), 'Cloud-based software platform for big data analytics in smart grids', *Computing in Science & Engineering* **15**(4), 38–47.

Tascikaraoglu, A. & Sanandaji, B. M. (2016), 'Short-term residential electric load forecasting: A compressive spatio-temporal approach', *Energy and Buildings* **111**, 380–392.

Unterberger, V., Lichtenegger, K., Kaisermayer, V., Gölles, M. & Horn, M. (2021), 'An adaptive short-term forecasting method for the energy yield of flat-plate solar collector systems', *Applied Energy* **293**, 116891.

VanDeventer, W., Jamei, E., Thirunavukkarasu, G. S., Seyedmahmoudian, M., Soon, T. K., Horan, B., Mekhilef, S. & Stojcevski, A. (2019), 'Short-term pv power forecasting using hybrid gasvm technique', *Renewable energy* **140**, 367–379.

Warren, P. (2014), 'A review of demand-side management policy in the uk', *Renewable and Sustainable Energy Reviews* **29**, 941–951.

Widmer, G. & Kubat, M. (1996), 'Learning in the presence of concept drift and hidden contexts', *Machine learning* **23**(1), 69–101.

Zhang, X. M., Grolinger, K., Capretz, M. A. M. & Seewald, L. (2018), Forecasting residential energy consumption: Single household perspective, *in* '2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)', pp. 110–117.

Zhuozhao Li, Ryan Chard, L. W. K. C. T. J. S. Y. B. A. W. S. T. B. B. M. J. F. I. F. (2021), 'Dlhub: Simplifying publication, discovery, and use of machine learning models in science', *Journal of Parallel and Distributed Computing* **147**, 64–76.

Žliobaitė, I., Pechenizkiy, M. & Gama, J. (2016), 'An overview of concept drift applications', *Big data analysis: new algorithms for a new society* pp. 91–114.