DeMand: A Tool for Evaluating and Comparing Device-Level Demand and Supply Forecast Models

Bijay Neupane
Aalborg University
bn21@cs.aau.dk

Laurynas Šikšnys
Aalborg University
siksnys@cs.aau.dk

Torben Bach Pedersen
Aalborg University
tbp@cs.aau.dk

ABSTRACT

Fine-grained device-level predictions of both shiftable and non-shiftable energy demand and supply is vital in order to take advantage of Demand Response (DR) for efficient utilization of Renewable Energy Sources. The selection of an effective device-level load forecast model is a challenging task, mainly due to the diversity of the models and the lack of proper tools and datasets that can be used to validate them. In this paper, we introduce the DeMand system for fine-tuning, analyzing, and validating the device-level forecast models. The system offers several built-in device-level measurement datasets, forecast models, features, and errors measures, thus semi-automating most of the steps of the forecast model selection and validation process. This paper presents the architecture and data model of the DeMand system; and provides a use-case example on how one particular forecast model for predicting a device state can be analyzed and validated using the DeMand system.

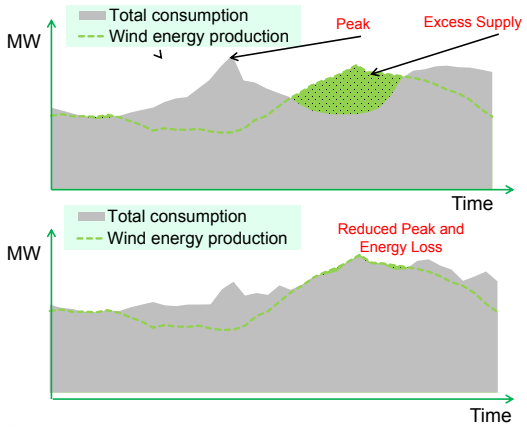


Figure 1: Balancing the Demand and the Supply from RES

1. INTRODUCTION

Renewable Energy Sources (RES) are increasingly becoming an important part of the future power grid. However, the dependence of RES production on weather conditions, such as periods with wind and sunshine, creates challenges related to balancing electricity demand with intermittent RES supply. This makes RES more difficult to integrate to the power grid, compared to traditional (fossil-based) energy sources.

To address the RES integration challenges, there are numerous smart grid projects aiming at the efficient utilization of intermittent RES production. For example, the TotalFlex [1] project proposes a Demand Response (DR) technique to actively control electricity consumption and production of individual household devices in order to confront the challenges of intermittent RES supply. The project utilizes shiftable aggregated demands from household devices (e.g., dishwashers, washing machines) to generate a demand and supply schedules that minimize differences between demand and supply, as shown in Figure 1. In TotalFlex (and many other DR projects), accurate and timely predictions of non-shiftable and shiftable energy are vital in order to generate effective schedules; otherwise forecast errors might lead to high-cost imbalances in the power grid. Therefore, underlying device-level forecast models of both non-shiftable and shiftable energy demands are required for being able to effectively plan electricity consumption and production.

The selection of the best forecast models for a variety of devices, data granularity, and forecast horizon is a challenging and resource-intensive task, mainly due to the (1) the diversity of the models, (2) the lack of proper tools, similar to [10], and (3) the unavailability of proper datasets that can be used to validate all these models.

First, the device-level energy demand highly depends on the device type and its functionality. For example, heating devices (e.g., a heat pump) operate for long durations and the demand for each timestamp depends on various factors such as climate, temperature, room size, past demands, etc. On the other hand, electric vehicles need to be charged for a couple of hours and energy demand depends on factors such as *current charge level*, *capacity*, *charging rate*, etc. Further, the energy demand for household devices (e.g., a dishwasher) depends on the user's behavior and other external factors such as the *time of use*, *duration of use*, *frequency of use*,etc. In addition, the amount of energy and the concrete energy profiles depend on the configuration of a device and in some cases are user-specific [7]. Therefore, it is hard to design a single model that considers all these factors and handles the stochasticity associated with device level forecasts. Thus, instead of a single generalizable model, a large variety of models are required to forecast device level demands for different types of household devices.

Second, the forecast model selection and validation process includes a number of steps which are often time-consum- ing (see Figure 2). In this process, researchers have to spend an enormous amount of time, especially, in data preprocessing, i.e., *data ex-*

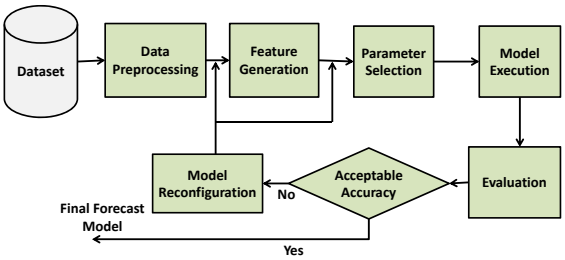


Figure 2: The process of forecast model selection and validation

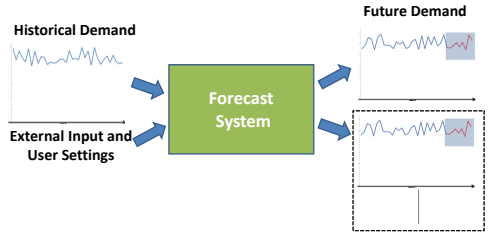


Figure 3: Inputs and outputs of the DeMand system

traction, *cleaning*, *transformation*, and *handling outliers and observation gaps*, and feature generation, where a set of features is generated repeatedly until a sufficient model accuracy is obtained. Here a *feature* is a variable, derived based on input dataset values or additional external information, that is assumed to be helpful for improving forecasting accuracy, e.g., temperature, wind speed, the day of the week, potentially, influencing demand and supply.

Third, we can find a number of works dedicated to forecasting and analyzing device-level demand [3, 9, 2, 7]. However, work on device-level forecasting is still limited, because experiments are typically fine-tuned for a particular dataset and usage patterns, and are hardly reproducible with the reported level of accuracy. Furthermore, efficient and precise extraction of all relevant device-level data is a challenging and still ongoing research [8, 4, 5]. Consequently, there is a lack of proper datasets containing high-resolution measurements of a large number of devices that include all the relevant external influences. The experiments are typically performed with private datasets containing measurements that are collected within the scope of a project and are not freely available. Even the freely available datasets only include measurements for limited (short) time durations and are often too noisy to perform any detailed analysis [6]. Lastly, there are no effective tools designed specifically for tuning and validating various device-level demand and supply forecasting models based on real measurements.

In the light of these challenges, we present the DeMand system that allows the user to analyse and validate various forecasting models using a number of provided datasets and built-in or user-defined functions. The system is designed to automate most of the preprocessing steps. At the same time, it provides flexibility for the user (a researcher or energy market player) to use either existing system modules or plug-in custom user-defined modules. Figure 3 shows the inputs and outputs of the DeMand system. The system offers the following features and functionality: i) a repository of available device-level datasets for evaluating and comparing forecast models, ii) access to existing (standard) forecasting algorithms, iii) dynamic generation of features for various forecast horizons and data resolutions, iv) support for various experimental configurations and generation of multiple forecast models, v) functionality to compare experiment results, and vi) easy integration of external features and learning algorithms.

The remainder of the paper is organized as follows. Section 2 describes the system architecture and functionalities. Section 3 describes its data model. Section 4 gives the use-case example of a system. Finally, Section 5 concludes the paper and provides future work directions.

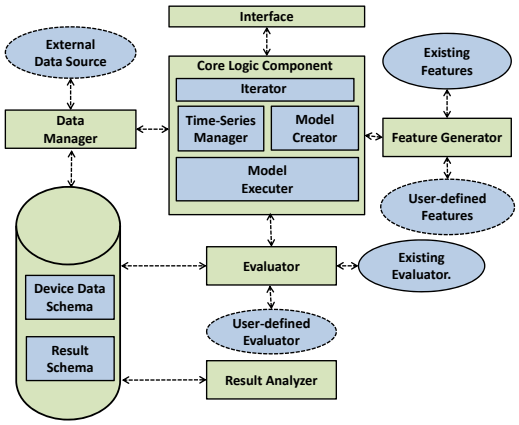


Figure 4: System Components

2. DeMand SYSTEM OVERVIEW

In this section, we present the architecture and the functionality of the DeMand system. The system is designed as a tool to automate experiments on device-level forecasting and to facilitate the comparison and re-(evaluation) of the existing experiments. Further, the system provides flexibility in using the available resources or uploading user-defined resources such as datasets, forecast models, evaluation metrics, etc. The user can define all necessary parameters and system configuration using the user interfaces. Once the user selects the timeseries and configures the experimental setup, the system provides all the available suggestion for the experiments such as a list of features, evaluation metrics, etc. Furthermore, the DeMand system is also envisioned to provide an open repository of device-level datasets that will be accessible to the research community for further experiments. Therefore, in addition to the graphical display in the interface, the experimental results and datasets are stored in the system database.

The DeMand system with the most essential components (rectangles) and their dependencies is shown in Figure 4. Here, the use of independent components for feature extraction, evaluator, and data management, etc. allows adding and removing system features in the plug-and-play fashion, making the system highly flexible and customizable for specific use-cases. We now present each of these components individually.

2.1 Interface Component

This components is designed to simplify and speed-up the process of setting up an experiment. Specifically, it offers a graphical user interface (GUI) and allows selecting a data source, a predictor, the values of configuration parameters, features, and error measures to use. It also provides visualization of time-series data. The user can set multiple values for the parameters to submit multiple tasks in a single execution. It also offers the visual representation of outcomes of the experiment, which plays a significant role in the forecast model analysis. Thus, in the end, the interface plots graphs for the experimental outcomes, according to the selected settings. For the multiple sets of parameters, the interface shows detailed plots for each configuration. If the user utilizes all the existing modules and functionality, the interface can reduce model analysis time drastically.

2.2 Core Logic Component

Core Logic is the central component in the system, and it orchestrates data manipulations and data exchange between other components according to user settings. The Core Logic component includes four different sub-components, shown in Figure 4, that automate the data preprocessing and parameter selection steps, shown in Figure 2.

Iterator The Iterator sub-component is responsible for parsing all the input parameters and determining the number of tasks to be executed. Here, a task is an independent execution of a forecast model with a particular predictor, dataset, parameters, etc. For example, if the user has selected two values of the probability threshold, e.g., in a classification task, then the iterator executes the same forecast model for each threshold value with the other parameters unchanged.

Model Creator The Model Creator sub-component creates an object that contains all the required parameters for the tasks. Then, all other (sub-)components fetch the parameters from this object. For multiple tasks, the sub-component updates object elements (parameters) that take more than a single value. The Model Creator sub-component is also responsible for persistent storage of model parameters until the completion of the current task.

Time Series Manager The Time Series Manager sub-component is responsible for the automation of all data preparation and manipulation tasks, such as data aggregation, noise filtration, filling observation gaps, etc. Further, it also acts as a communication bridge between the Data Manager and the Feature Generator components.

Model Executer The Model Executer sub-component executes the task with the provided feature set and configuration. If the experimental configuration already exists in the database, i.e., the experiment is not unique, the Model Executer terminates the current task and extracts the prediction results from the database. Further, the Model Executer handles errors in user-defined predictors/classifiers by replacing them with the default predictor; else it terminates the current task and requests the Iterator sub-component to delete all remaining tasks in the queue. At the end of the execution, it passes the predicted values to the (model) Evaluator component.

2.3 Data Manager Component

The Data Manager component is responsible for extracting time-series data from a user-defined source, i.e., a database or files in a user-specified location, and feeding extracted data to the System Core component. The Data Manager includes a data parser that transforms the raw data into the required format. After the data is successfully parsed, it is then persistently stored in the database for further analysis. This gives the user the comprehensive repository of datasets (e.g., for different device types) for future experiments, where the validation of forecast models in homogeneous environments becomes possible. Although the database is confined to the energy domain, there are no methods to validate the domain of the dataset except the required format. Thus, it is possible to upload timeseries from any domain without getting an error. However, a user can manually validate the timeseries before submitting the task by using the graphical interface (plot) provided by the system. The database schemas are described in Section 3.

2.4 Feature Generator Component

The Feature Generator component is responsible for the generation of all features chosen by the user. As discussed earlier, *features* are variables (e.g., temperature or the day of the week), derived based on the input dataset values (time series) or external information, specified to, potentially, improve forecasting accuracy. The features are generated by using pre-defined functions from the feature repository. Additionally, the user can define new functions for the specification of custom features.

Script 1: Python code for user defined feature

```
#INPUT: time series with date column at first
#OUTPUT: binary feature representing weekdays
# or weekend for each data point
from datetime import datetime as dt
def is_weekend(timeseries = None):
    if timeseries == None:
        raise TypeError('Data can not be null')
    elif type(timeseries[0]) is not dt.date:
        raise TypeError('First column must be a
                         datetime.date, not a %s'
                         % type(timeseries[0]))
    else:
        feature_series = []
        for item in timeseries:
            day_of_the_week = item[0].isoweekday()
            if day_of_the_week <=5: # not weekend
                feature_series.append(0)
            else: # is weekend
                feature_series.append(1)
    return feature_series #binary features
```

For example, Script 1 shows a simple user-defined function given in Python, which, for each data point in the original time series, identifies if a specific data point has been recorded during the weekend (value 1) or during a weekday (value 0). The component calls this user-defined function with the time series dataset as the input parameter. The function computes the features for each row in the time series and returns an output. The feature generator calculates the size of the output, i.e., the number of attributes, and increases the size of the feature vector by the respective number. In some cases, the device-level measurement dataset contains some sensitive context information, such as location, family size, age, group, occupation, etc. Therefore, this information is only accessible through the feature generator and is never revealed to the user. This approach facilitates the user when using sensitive information as features without requiring to disclose such information.

2.5 Evaluator Component

The Evaluator component receives as input the output of the predictor/classifier and evaluates its performance according to the chosen error measure and parameters. Further, it writes the experiment attributes and results to the database in the Result Schema for future reference and queries. The evaluator also invokes the functions to plot the graphical representation of the experimental results. The user can select existing error measures (out of many available ones), or define custom measures using a Python function, similarly to feature specification.

2.6 Result Analyser Component

The Result Analyzer is a component that processes the user queries and fetches the requested dataset from the system database. The user can write simple SQL queries to extract all the relevant data and results of an experiment satisfying certain conditions, such as type of predictor, dataset, forecast horizon, etc. Further, the user can write queries to compare experiments using a particular error measure. For example, let us consider a binary classification task where the objective is to predict a device state, i.e., idle (0) or active (1), at a particular hour in the future. Using the SQL query shown in Script 2, a user can query the results of the entire classification tasks based on the logistic regression model, ordered in the decreasing order of the Area Under the Curve (AUC) value. Further, from the list of available results, the user can select up to two

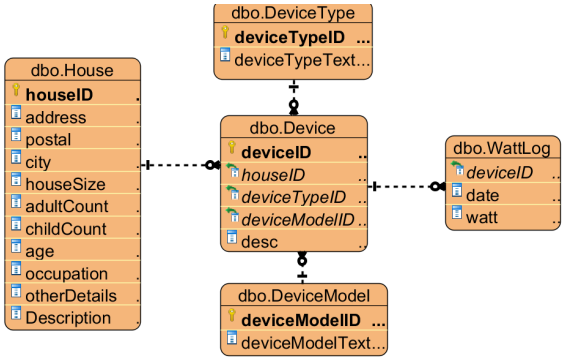


Figure 5: The schema of the Device Measurements Database

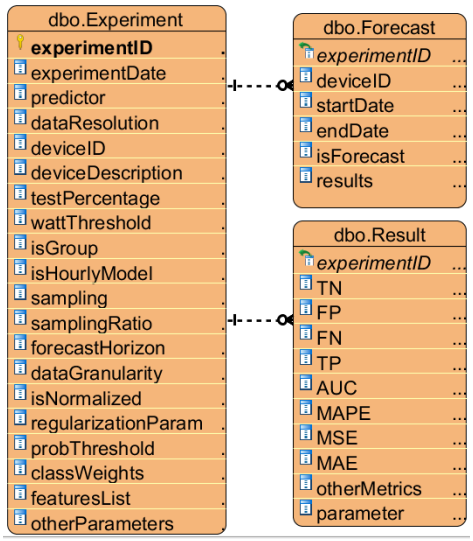


Figure 6: The Schema of the Results Database

experiments and compare their performance graphically. Finally, if needed, the earlier experiments can be re-executed on a new dataset using the same or new configuration of parameters.

Script 2: Sql Script to extract data of earlier experiments

```
Select a.*, AUC from Experiment a
left join
(select experimentID, AUC from Result) b
on a.experimentID = b.experimentID
where predictor = 'Logistic_Regression'
order by AUC desc
```

3. DATABASE SCHEMAS

The persistent storage of the device measurements and results is essential for being able to compare and (re-)evaluate different forecast models using multiple datasets and configuration parameters. Since the device level datasets are stored into the MSSQL database, currently we have used the same database system to store the experimental results. However, the DeMand system is envisioned to support diverse types of data, such as various time series, varying experiment configurations, results, etc. that cannot be confined to a strict schema. Therefore, the inclusion of non-relational and schema-less data models such as NoSQL, JSON would be a good choice as an add-on for the next version of the system. In the DeMand system, device measurements and results are separated and stored in the database using two different schemas, presented next.

3.1 Device Measurements Database

The schema of the device measurements database is shown in Figure 5. The *House* table stores all the information regarding the owner of the device, and the table *Device* stores device information. We consider that households can be represented by a large number of categories. As it is impractical to have columns for all possible categories, we thus have only a few columns for the generic categories, and the rest of the details are stored in the *otherDetails* column as a list of key-value pairs, such as *(regularization, .01),(windowSize, 7)*. The amount of energy depends on the type of the device. Therefore, the types of devices are stored in the *DeviceType* table. Further, even similar devices can show significant variation in the energy demand due to differences among their models; thus, the device models are stored in the *DeviceModel* table.

A new dataset uploaded by the user is stored in the same schema, after it is parsed and validated by the Database Manager component. The requirement for the new dataset to be eligible for storage is that it has to have a label for device type and a timestamp value for each data point. Further, we know that for the device-level dataset, additional context information is rarely available. Therefore, in the case of missing information, the tables are filled with default values in the mandatory (*NOT NULL*) fields and generated unique values in the *primary key* fields. Further, even if the user provides the values for the primary keys, such as, *houseID*, *deviceID*, etc., the system automatically appends unique values to avoid primary key conflicts. At present, the device measurements database contains the energy consumption profiles for 200+ devices from 13 different households, representing 14 different type of devices.

3.2 Results Database

The experiment results and configuration parameter values are stored in the three tables, shown in Figure 6.

Here, the Experiment table stores records for each performed experiment, together with all used configuration parameter values. However, actual parameters used in an experiment differ based on the type of predictor. Even the same family of predictor might require different sets of parameters depending on its implementation. For example, L1 regularized logistic regression has an extra penalizing parameter λ, unlike its simple implementation. Therefore, we have only a few columns in the *Experiment* table for the generic parameters, such as forecast horizon, data granularity, threshold values, etc. The remaining parameters are stored in the column *otherParameters* as a list of key-value pairs. Finally, the description of all the features used in an experiment is stored in the column *featureList*.

The *Forecast* table stores information about the complete time series or its fragment (defined by *startDate endDate*) used in an experiment. It also stores the output of the predictor and the corresponding test labels, where the predictor output values are distinguished from the test data by using binary values in the *isForecast* column, i.e., the value of 1 in the *isForecast* column indicates the predictor output values (forecasts).

The *Result* table stores the values of various predictor performance measures. As before, there exist a large number of measures for performance evaluation. Therefore, we have columns for only a few frequently used performance measures; and the values of new measures are stored in the *otherMetrics* column as a list of key-value pairs. A single experiment can produce more than one result depending on the chosen values of a parameter. For example, recall the classification task where the classifier produces two dif-

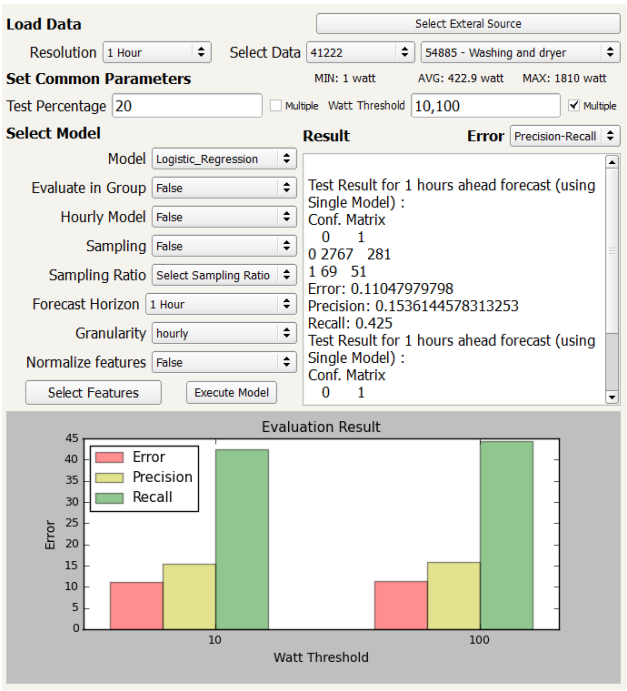


Figure 7: The main window of the DeMand system

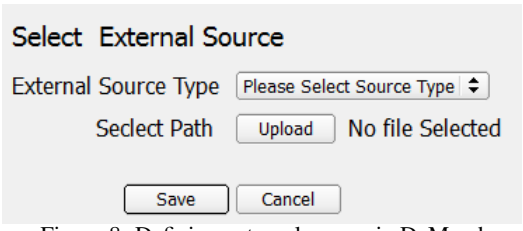


Figure 8: Defining external source in DeMand

ferent results – one for each probability threshold value. Therefore, the *Result* table can have multiple rows for single experiments.

4. DeMand USE-CASE EXAMPLE

In this section, we briefly walk-through the DeMand functionality using the real-world use-case of the device-level forecast model analysis. Here, we continue with the binary classification problem of predicting a device state for the next hour.

4.1 Execution of Forecasting

First, we start by making a choice of the dataset to use for the experiment. In our case, we select the consumption time series of a washer dryer from the existing database. Alternatively, we can also select a new dataset using the window shown in Figure 8. The system then automatically plots the selected dataset in the main window and also provides some general statistics on the dataset, such as minimum, average, and maximum demand (see Figure 7). This information is helpful in deciding the threshold value (in watts) for the segmentation of data based on a device state (active or ideal). In our example, we have selected two values 10 watts and 100 watts as the threshold (see Figure 7). As result, the system creates two experiments to generate a classification model for each of the threshold values.

Next, we select the percentage of time series to use as a test set (20% in our example) using the main DeMand system window, shown in Figure 7. The timeseries is sequentially split into training and test sets based on the selected split ratio. Then, we also choose a predictor to be used for classification, which, in our example, is the *Logistic Regression* with *L1 regularization* (LR-L1). Further, we configure an hour-ahead forecast model by selecting the forecast horizon and data granularity of 1 hour. Afterwards, we select the set of features which we think will help to improve the performance of the classifier. As shown in Figure 9, the system automatically provides a list of available features that can be used with the selected dataset. In our example, as seen in Figure9, we select all the available features. Further, we choose the precision-recall curve as the error measure for evaluating the classifier (see Figure 7).

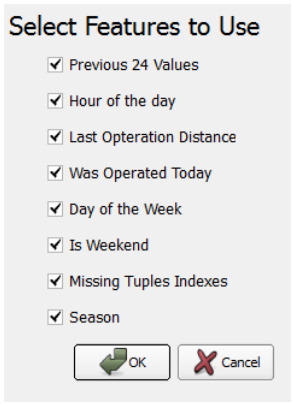
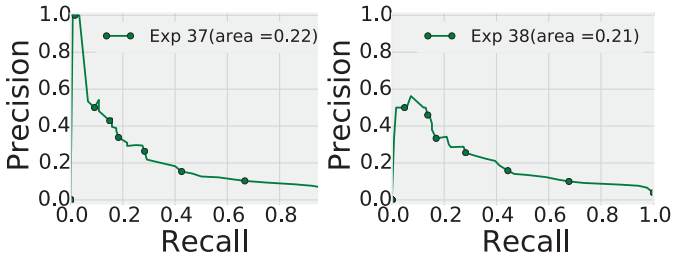


Figure 9: Feature selection in DeMand



(a) 10 watt (b) 100 watts

Figure 10: Precision-Recall curve: for various demand thresholds

Finally, we execute the experiments from the completely configured experiment. The system might still request some additional parameters specific to the predictor. In the case of user-defined predictors, all additional parameters have to be handled by the user. To set the parameter value, a user has to include a call to the $add\_parameter$ function with a parameter name and a value as input. In our example, the LR-L1 model requires the values of the penalty parameter λ and the probability threshold. After we provide the values of all required parameters, the system executes the experiment and stores results in the database along with all forecasted values.

4.2 Result Presentation

After the completion of all experiments, the results are presented in the main window, as shown in the Figure 7. The system plots the values of the classifiers according to the chosen parameters. In our example, the system plots performance values in terms of *error*, *precision*, and *recall* of two classifiers with different *threshold watt* parameter values. The system also provides a detailed description of the results in the main window as a textual description. Further, if we click on the individual plots, the system automatically shows

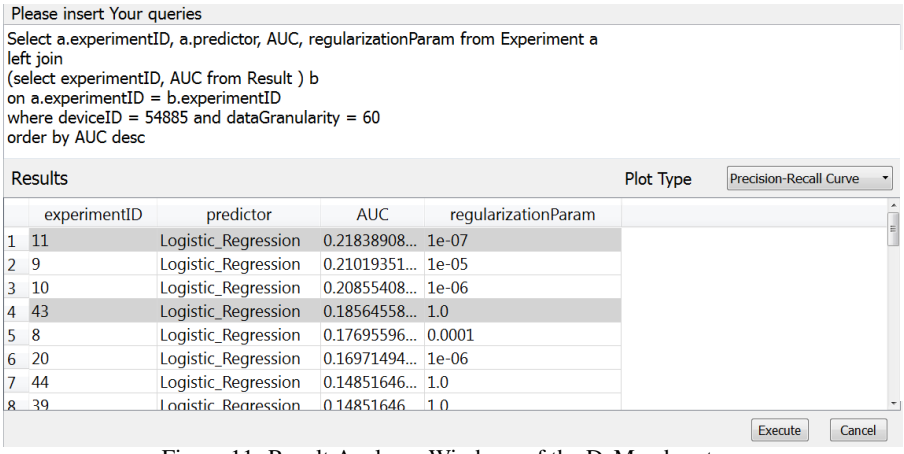


Figure 11: Result Analyzer Windows of the DeMand system

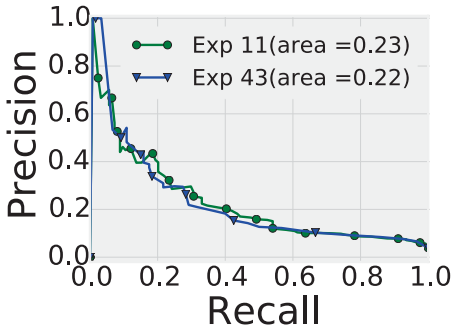


Figure 12: Comparison between experiments

precision-recall curves for each classifier, as seen in Figures 10a and 10b. For a detailed comparison of the results, we can also use the result analyzer, as illustrated in Figure 11. In this example, we query all experiments that has been performed with the same *deviceID* and *dataGranularity* parameter values, sorted based on AUC values. The output of the query can be seen in the lower section of the Figure 11. Here, the experiment at the top of the table has the best performance in terms of the selected error measure. Additionally, we can choose any two experiments for a visual side-by-side comparison, as shown in Figure 12.

As seen from the use-case example presented above, the DeMand system provides an analytical (decision-support) platform for comparing, validating, and choosing different forecast models and their parameter values. With the comprehensive model comparison information offered by the DeMand system, the user can decide on the best prediction model (algorithm) and select its parameters for a specific dataset or the given collection of datasets. As it can be seen from this use-case, the DeMand system with all its features and built-in functionality allows significantly reducing time needed to select and validate forecast models, compared to using general tools or hard-coded solutions requiring, typically, much more data pre-processing, system configuration, and result post-processing time.

5. CONCLUSION AND FUTURE WORK

In this paper, we have presented the DeMand system for fine-tuning, analyzing, and validating the device-level forecast models. The system offers a number of built-in device-level measurement datasets, forecast models, features, and errors measures; and allows users to evaluate and compare different forecast models based on different parameters, making device-level forecasting more accessible and efficient.

In this paper, we have presented the architecture and a data model of the DeMand system. We also provided the use-case example on how a forecast model for predicting a device state can be analyzed using the DeMand system. Thus, we showed that DeMand is an easy-to-use system automating most of the steps of the forecast model selection and validation process.

In the future, we plan to integrate additional features such as i) an API-centric architecture for the result analyzer, ii) a model and parameters recommender system, iii) a more flexible comprehensive data processor, and iv) ensemble learning. We foresee that the full potential of the DeMand system is to be unleashed, if the system is used repeatedly, possible by multiple users, allowing to build-up and utilize a large repository of device-level data and predictors.

6. ACKNOWLEDGMENTS

This work was supported in part by the TotalFlex project sponsored by the ForskEL program of Energinet.dk.

7. REFERENCES

[1] The *TotalFlex* project, 2014. `http://www.totalflex.dk/Forside/`.

[2] A. Alhamoud, P. Xu, F. Englert, A. Reinhardt, P. Scholl, and R. Steinmetz. Extracting human behavior patterns from appliance-level power consumption data. In *Wireless Sensor Networks*, Lecture Notes in Computer Science. 2015.

[3] A. Barbato, A. Capone, M. Rodolfi, and D. Tagliaferri. Forecasting the usage of household appliances through power meter sensors for demand management in the smart grid. In *Smart Grid Communications (SmartGridComm), 2011 IEEE International Conference on*, pages 404–409, 2011.

[4] D. Egarter, V. Bhuvana, and W. Elmenreich. Paldi: Online load disaggregation via particle filtering. *Instrumentation and Measurement, IEEE Transactions on*, 64(2):467–477, 2015.

[5] S. Gupta, M. S. Reynolds, and S. N. Patel. Electrisense: Single-point sensing using emi for electrical event detection and classification in the home. In *Proceedings of the 12th ACM International Conference on Ubiquitous Computing*, UbiComp '10, pages 139–148, 2010.

[6] J. Z. Kolter and M. J. Johnson. REDD: A Public Data Set for Energy Disaggregation Research. In *SustKDD workshop*, 2011.

[7] B. Neupane, T. Pedersen, and B. Thiesson. Towards flexibility detection in device-level energy consumption. In *Proceedings of the Second ECML/PKDD Workshop, DARE'14*, pages 1–16. 2014.

[8] O. Parson, S. Ghosh, M. Weal, and A. Rogers. An unsupervised training method for non-intrusive appliance load monitoring. *Artificial Intelligence*, pages 1 – 19, 2014.

[9] A. Reinhardt, D. Christin, and S. S. Kanhere. Can smart plugs predict electric power consumption?: A case study. In *Proceedings of the 11th International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services*, pages 257–266, 2014.

[10] R. Ulbricht, U. Fischer, L. Kegel, D. Habich, H. Donker, and W. Lehner. Ecast: A benchmark framework for renewable energy forecasting systems. In *EDBT/ICDT Workshops*, pages 148–155, 2014.