7

iURBAN Smart Algorithms

Sergio Jurado and Alberto Fernandez

Sensing & Control, Barcelona, Spain

Abstract

This chapter presents the smart algorithms developed in iURBAN. Prediction of consumption and production of energy, dynamic tariff comparison and demand response.

Keywords: Energy prognosis, Demand response, Dynamic tariff.

7.1 Introduction

This chapter outlines details for the following smart algorithm modules:

- "As is" generation and consumption forecasts using artificial intelligence (AI).
- Dynamic tariff comparison and demand response (DR) simulations.

The algorithms for DR analysis, renewable energy production analysis, and tariff analysis were developed with respect to the electricity grid. The algorithms are integrated with the virtual power plant (VPP) to perform energy analysis of the overall city in order to enable balancing energy at both local and centralized levels.

Based on forecasting weather data which is be stored in the Smart City Database (SCDB), the prediction algorithms are able to predict the energy consumption, production, transfer, and storage of energy at the city level. For example, for the next 24, 48, and 72 hours, they can advice how to reduce consumption (covered by the local decision support system.), how to integrate new technologies, how to optimize existing technologies, and how to determine the optimum time to buy and sell energy to the electricity and/or vehicle grids or to store energy from centralized DER sources and assist iURBAN decision support systems.

The prediction algorithms consist of 3 modules: the AI module, the tariff analysis module (including DR aspects), and the weather forecast module (weather data are obtained from professional weather data provider www.weatheranalytics.com).

7.2 "As is" Generation and Consumption Forecasts

7.2.1 Introduction

A large variety of AI techniques have been applied in the field of short-term electricity consumption forecasting, showing a better performance than classical techniques. Specifically, machine learning has been proven to accurately predict electric consumption under uncertainties. For instance, Khamis proposes in [1] a multilayer perceptron neural network to predict the electricity consumption for a small-scale power system, obtaining a better performance than with traditional methods, while Marvuglia et al. consider Elman neural network for the short forecasting of the household electric consumption with prediction errors under 5% [2]. Also in [3], a study of electric load forecasting is carried out with classification and regression trees (CART) and other soft computing techniques obtaining again better results than classical approaches.

Large-scale studies for comparing machine learning and soft computing tools have focused on the classification domain [4]. On the other hand, very few extensive studies can be found in the regression domain. In [5], Nesrren et al. carried out a large-scale comparison of machine learning models for time series forecasting. The study includes techniques such as K-nearest neighbors (KNN), CART regression trees, multilayer perceptron networks, support vector machines, Gaussian processes, Bayesian neural networks, and radial basis functions. The research reveals significant differences between the methods studied and concludes that the best techniques for time series forecasting are multilayer perceptron and Gaussian regression when applied on the monthly M3 time series competition data (a thousand-time series) [6]. Moreover, in [7], an empirical comparison of regression analysis is carried out, as well as decision trees and artificial neural network (ANN) techniques for the prediction of electricity energy consumption. The conclusion was that the decision tree and the neural network models perform slightly better than regression analysis in the summer and winter phases, respectively. However, the differences between the three types of models are quite small in general, indicating that the three modeling techniques are comparable when predicting energy consumption.

In recent studies, the good omen of AI techniques is shown. For instance, Jurado et al. [8] propose an entropy-based feature selection process to select the most important past consumption values and it is combined with machine learning and soft computing techniques. Moreover, these hybrid methodologies are compared against a typical statistical technique (ARIMA), resulting in higher accuracies for FIR, random forest, and artificial neural networks (NNs). Thus, hybrid methodologies combine the strengths of different techniques to achieve higher accuracies in predictions. The results showed in this study also highlight the adaptability and scalability of these models to buildings with different profiles (location, usage, etc.).

In addition, in contrast to other approaches where offline modeling takes considerable computational time and resources, the models discussed in [8] appear to generate fast and reliable models, with low computational costs. These models can be embedded, for instance, in a second generation of smart meters where they could generate on-site forecasting of the consumption and/or production in the next hours or even trade the excess energy with other smart meters.

Considering the strengths of some AI methodologies reported in the literature (prediction of unexpected changes, predict energy consumptions and productions in different types of buildings, etc.) and the experience inside the consortium with such techniques, AI models are the approach followed for the *generation and consumption forecast*.

7.2.1.1 Random forest

Random forest (RF) is a set of CART, which was first put forward by Breiman [9]. In RF, the training sample set for a base classifier is constructed by using the Bagging algorithm [10]. In traditional CART, each inner node is a subset of the initial data set and the root node contains all the initial data. RF is a combination of tree predictors such that each tree depends on the values of a random vector sampled independently and with the same distribution for all trees in the forest. RFs for regression are formed by growing trees depending on a random vector such that the tree predictor takes on numerical values as opposed to class labels. The RF predictor is formed by taking the average over *B* of the trees. Figure 7.1 shows a scheme of the random forest.

Assuming the following basic notations:

- Let the number of training cases be *N* and the number of variables be *P*
- Input data point $\rightarrow v = (x_1, \ldots, x_P) \in \mathbb{R}^P$
- Output variable $\rightarrow$ c
- Let the number of trees be *B*

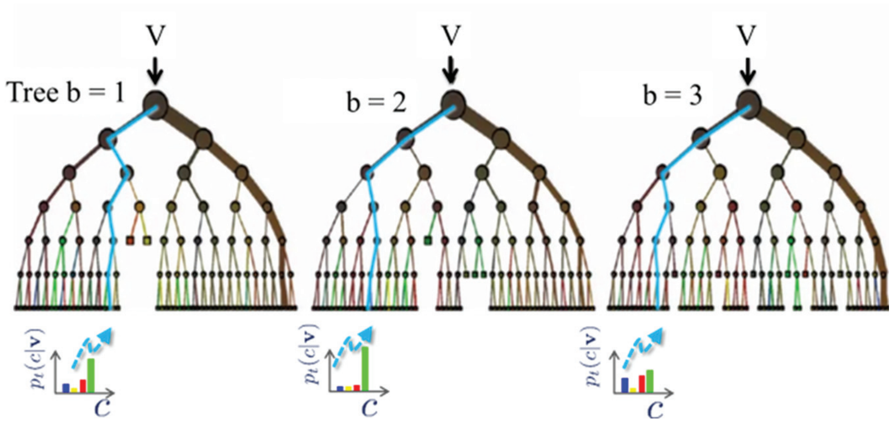


Figure 7.1 Random forest scheme containing three different trees.

The schematic RF algorithm is the following:

1. For b = 1 to B
   a. Draw a bootstrap sample Z^* of size N from the training data
   b. Grow a random forest tree T_b to the bootstrapped data, by recursively repeating the following steps for each terminal node of the tree, until the stopping criteria is reached:
      i. Select m variables at random from the P variables
      ii. Pick the best variable/split point among the m
      iii. Split the node into two daughter nodes
2. Output the ensemble of trees $\{T_b\}_1^B : p(c \mid v) = \frac{1}{B}\sum_1^B p_b(c \mid v)$

The size N of the bootstrap sample Z^* can go from a small size to the size of the whole data set. However, with large training data sets, assuming the same size can significantly affect computational cost. In addition, for big data problems such as the forecasting of consumption/production of all the buildings in a city, a good definition of the parameter N is mandatory. Moreover, there are different stopping criteria; two of the most commonly used are as follows: (1) until the minimum node size $n_{\min}$ is reached and (2) when a maximum tree depth is reached.

Although RF has been observed to overfit some data sets with noisy classification/regression tasks [11], it usually provides accurate results, generalizes well, and learns fast. In addition, it is suitable to handle missing data and provides a tree structured method for regression [12].

7.2.1.2 Artificial neural network

NNs are a very popular data mining and image processing tool. Their origin stems from the attempt to model the human thought process as an algorithm which can be efficiently run on a computer. Its origins date back to 1943, when neurophysiologist W. McCulloch and mathematician W. Pitts wrote a paper on how neurons might work [13], and they modeled a simple NN using electrical circuits. Some years later, in 1958, F. Rosenblatt created the perceptron, an algorithm for pattern recognition based on a two-layer learning computer network using simple addition and subtraction [14].

Many time series models are based on NN [15]. Despite the many desirable features of NNs, constructing a good network for a particular application is a nontrivial task. It involves choosing an appropriate architecture (the number of layers, the number of units in each layer, and the connections among units), selecting the transfer functions of the middle and output units, designing a training algorithm, choosing initial weights, and specifying the stopping rule.

It is widely accepted that a three-layer feed forward network with an identity transfer function in the output unit and logistic functions in the middle-layer units can approximate any continuous function arbitrarily well given sufficient amount of middle-layer units [16]. Thus, the network used in this research is a three-layer feed forward network (Figure 7.2). The inputs are connected to the output via a middle layer.

When working with univariate time series, the neurons of the input layer contain the present value and the relevant past values of the univariate time series, while the output is the value for the next time period, computed as described in Equation (7.1).

$$S(t+1) = f(s(t), \ldots, s(t-n)) \tag{7.1}$$

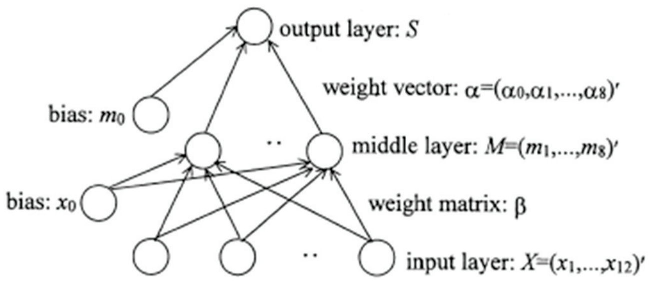

output layer: S
weight vector: $\alpha=(\alpha_0,\alpha_1,\ldots,\alpha_8)'$
bias: m_0
middle layer: $M=(m_1,\ldots,m_8)'$
bias: x_0
weight matrix: β
input layer: $X=(x_1,\ldots,x_{12})'$

Figure 7.2 Representation scheme of a three-layer feed forward neural network.

where n is the number of past values of variable s and f is a nonlinear function approximated by a multilayer feed forward neural network (FNN) [17].

Recurrent neural networks are able to obtain very good prediction performance, since their architecture allows that the connections between units form a directed cycle, which allows it to exhibit dynamic temporal behavior. Unlike feed forward NNs, recurrent networks can use their internal memory to process arbitrary sequences of inputs. Therefore, recurrent NNs are very powerful, but they can be very complex and extremely slow compared to feed forward networks. As mentioned earlier, one of the main objectives of this research is to find powerful prediction methodologies with low computational costs, which could be embedded in a smart meter and generate on-site forecasting of the consumptions and/or productions. This is the reason why recurrent NNs have been rejected in this work and a cooperative approach has been chosen instead, i.e., FSP and a feed forward neural network that uses, as input variables, the most relevant past consumptions values.

7.2.1.3 Fuzzy inductive reasoning

The conceptualization of the fuzzy inductive reasoning (FIR) methodology arises from the general system problem solving (GSPS) approach proposed by Klir [18]. This methodology of modeling and simulation has the ability to describe systems that cannot be easily described by classical mathematics or statistics, i.e., systems for which the underlying physical laws are not well understood [19]. A FIR model is a qualitative nonparametric model based on fuzzy logic. Visual FIR is a tool based on the FIR methodology (runs under Matlab environment), which offers a new perspective to the modeling and simulation of complex systems. Visual FIR designs process blocks that allow the treatment of the model identification and prediction phases of FIR methodology in a compact, efficient, and user-friendly manner [20]. The FIR model consists of its structure (relevant variables or selected features) and a pattern rule base (a set of input/output relations or history behavior) that are defined as if-then rules. Once the best structure (mask) has been identified, it is used to obtain the pattern rule (called behavior matrix) from the fuzzified training data set. Each pattern rule is obtained by reading out the class values through the "holes" of the mask (the places where the mask has negative values) and place each class next to each other to compose the rule.

Once the behavior matrix and the mask are available, a prediction of future output states of the system can take place using the FIR inference engine, as described in Figure 7.3. This process is called qualitative simulation. The FIR

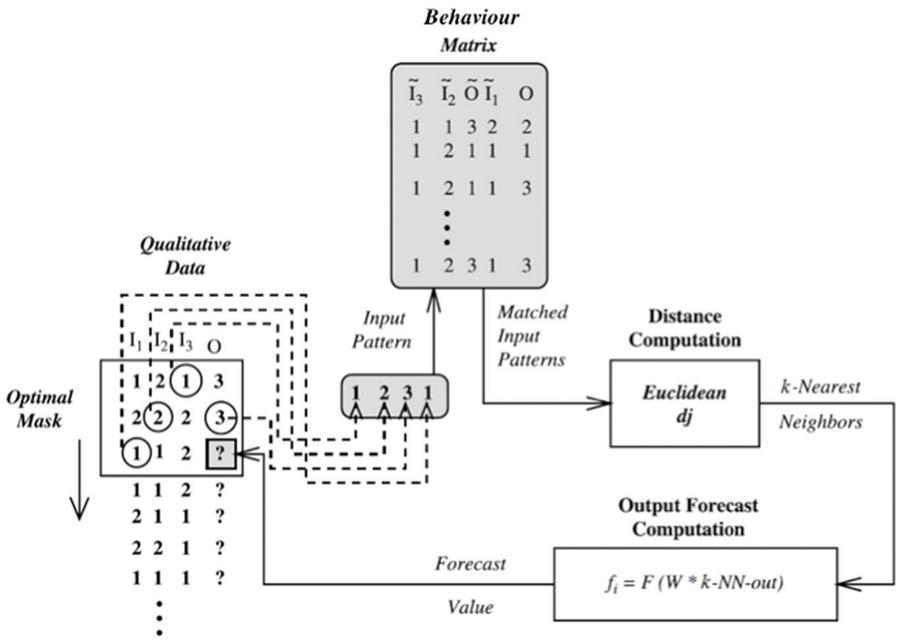


Figure 7.3 Qualitative simulation process diagram (with an example containing three inputs and one output).

inference engine is based on the KNN rule, commonly used in the pattern recognition field. The forecast of the output variable is obtained by means of the composition of the potential conclusion that results from firing the *k* rules whose antecedents have best matching with the actual state.

The mask is placed on top of the qualitative data matrix (fuzzified test set), in such a way that the output matches with the first element to be predicted. The values of the inputs are read out from the mask and the behavior matrix (pattern rule base) is used, as it is explained latter, to determine the future value of the output, which can then be copied back into the qualitative data matrix. The mask is then shifted further down one position to predict the next output value. This process is repeated until all the desired values have been forecast. The fuzzy forecasting process works as follows: The input pattern of the new input state is compared with those of all previous recordings of the same input state contained in the behavior matrix. For this purpose, a normalization function is computed for every element of the new input state and an Euclidean distance formula is used to select the KNN, the ones with smallest distance, that are used to forecast the new output state. The contribution of each neighbor to the estimation of the prediction of the new output state is a function of its proximity. This is expressed by giving a distance weight to each neighbor, as shown in Figure 7.3. The new output state values can be computed as a weighted sum of the output states of the previously observed five nearest neighbors.

The FIR methodology is, therefore, a modeling and simulation tool that is able to infer the model of the system under study very quickly and is a good option for real-time forecasting. Moreover, it is able to deal with missing data as has been already proved in a large number of applications [19]. On the other hand, some of its weaknesses are that as long as the depth and complexity increase, the computational cost increases too, and also the parameters to choose during the fuzzification phase (which can be mitigated using evolutionary algorithms to tune the parameters).

7.2.2 AI Generation and Consumption Forecast

7.2.2.1 Model generation

After an empirical analysis, we have finally selected RF for the first version of the prediction models. In contrast to most of the approaches previously explained, where offline modeling takes considerable computational time and resources, the technique used in the prediction algorithms, RF, appear to generate fast and reliable models, with low computational costs.

FIR is also a suitable technique with higher accuracies than RF but its development and implementation are more difficult than RF.

To perform a prediction, it is necessary to first generate a model. There are multiple strategies for the model generation, for instance to cluster similar installations and create a single model that would fit with all the installations within this cluster. The main advantage of this strategy is that only one model is created, and therefore, the computational cost and the storage capacity needed are very low. On the other hand, the accuracy may be lower than the approach, *one model one installation,* where a single model is created for each installation. In iURBAN, the model generated is unique for each installation. This would not be the best solution for a pilot with thousands of installations, because it would take high computational costs and increase storage capacity needs. However, since the iURBAN pilot has around 100 installations, the approach *one model one installation* is an affordable solution.

In Figure 7.4, the process to generate a prediction model and the online predictions is explained. In the left side of the figure, the *Model Generation* (1) is created/updated every week as default. The model is built-up taking all the historical past consumptions from the *Data Warehouse* (2) in the cloud.

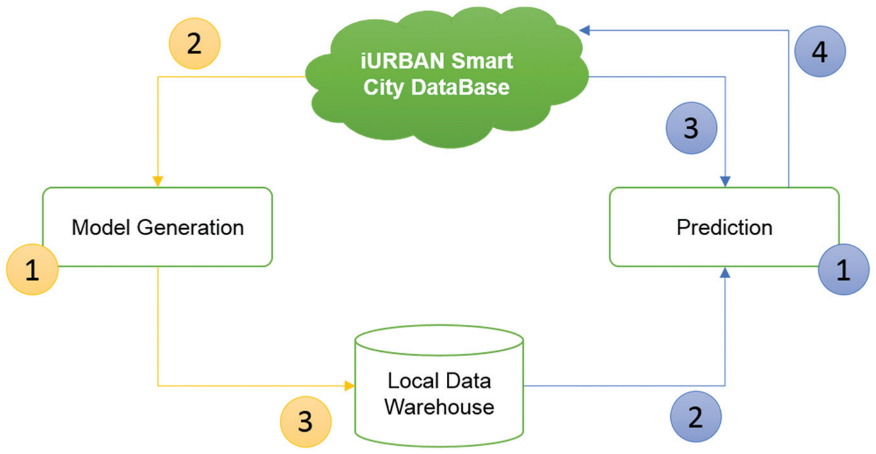


Figure 7.4 Prediction algorithm flow diagram.

After the model is created/updated, it is stored in the *Local Data Warehouse* (3) of the PC/server where it is executed.

On the other hand, in the right side of Figure 7.4, the *Prediction* (1) is performed every hour as default. When a prediction model is available in the *Local Data Warehouse,* it is loaded and used for the online prediction (2). The prediction is performed taking the last hours/days historical past consumptions from the *iURBAN SCDB* (3). After the prediction is performed, the results are sent to the *iURBAN SCDB* (4), where are available for any component in iURBAN.

7.2.2.2 Model and prediction configuration parameters

The granularity of the predictions is an important parameter because it defines the prediction intervals. Depending on the use case of the prediction data, granularity will be determined. For example, if the use case is to analyze potential blackouts, variability of renewables in the electricity grid, solar, and wind forecasting or real-time demand fluctuations, then the granularity level should be high because it requires a prediction at second level or even millisecond. This is not the case of iURBAN. On the other hand, if the use case is to create awareness about possible consumption/production at the end of the day, week, and/or month, at which hour is expected a peak, detect a day where higher consumptions or productions are expected, etc., then lower levels of granularity are needed. This is the case of iURBAN.

Highest granularity of data in Plovdiv and Rijeka is 15 min and 1 min, respectively. The smart algorithm modules are developed to accept any granularity level; however, based on the initial requirements of iURBAN, specifications, and LDSS/CDSS functionalities and graphical user interface (GUI), it has been decided that a good compromise of prediction interval is 60 min. The model is updated once per week.

7.2.2.3 Grids and levels

Prediction models run for master sensors of water, heating, gas, and electricity, as well as for global production of electricity and heating. Master sensors are those sensors measuring the total consumption or production in apartments, buildings, substations (heating), and combined heat and power (CHP) plants.

7.2.3 Development and Implementation

7.2.3.1 Code

The code was developed in Java. It consists of six different packages:

- *iurban.predictionengine.webapi* (contains one class):
  - *WebApiConsumer*. This class contains all the methods to interact with the SCDB; *HttpPost* and *HttpGet* to download historical consumption data and upload forecasting values.
- *iurban.predictionengine.models* (contains 5 classes):
  - *Attributes, Gap, Granularity, Outlier, Result, Stats, WeatherAnalytics, Sensor, Prediction, Installation, TimeSeries*. These classes contain the data structure to handle correctly the data obtained through the API.
- *iurban.predictionengine.utils* contains 15 classes. These classes contain common methods used in several processes of the model generation and prediction.
  - *ArrayUtils, Comparators, ErrorUtils, VectorUtils, WekaUtils, PropertiesUtils, etc.*
- *iurban.predictionengine.predictionmodel* that contains the two main classes for the prediction values generation:
  - *WekaTechnology*. This class contains the technology to design and generate the models in Weka.
  - *FirTechnology*. This class contains the technology to design and generate the models with FIR technique.
  - *PredictionOnTheFly*. Within this class, there are all the methods to perform the predictions with the model already generated.

- *iurban.predictionengine.prediction* is the package with the classes containing the main programs for the model generation and forecasting values generation.
  - *PredictionModelMain.* Class containing the main to create the models.
  - *PredictionOnTheFlyMain.* Class containing the main to generate the predictions.

7.2.3.2 Deployment

In order to deploy the AI forecasting models, it is necessary to first generate a *jar file* containing all the aforementioned classes, as well as the necessary libraries. The *jar* file can run in any PC or local/cloud server that has a Java Virtual Machine.

It has to be taken into account that the models generated are stored locally in the machine where the *jar* file is running.

Along with the *.jar* file, it is necessary to create a *config.properties* file containing the following information:

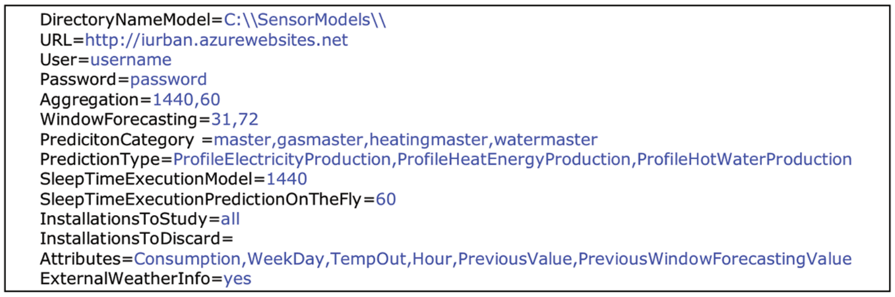
***DirectoyNameModel**:* Path where the models generated are stored.

***URL**:* Web API.

***User**:* iURBAN user with credentials to use the data.

***Password**:* Password of the user.

***Aggregation**:* Desired aggregation for the prediction in minutes. For instance, in the example above, the models and predictions are performed with daily (aggregation of 1440 min, which is equal to 24 h) and hourly data (aggregation of 60 min, which is equal to 1 h).



```
DirectoryNameModel=C:\\SensorModels\\
URL=http://iurban.azurewebsites.net
User=username
Password=password
Aggregation=1440,60
WindowForecasting=31,72
PredicitonCategory =master,gasmaster,heatingmaster,watermaster
PredictionType=ProfileElectricityProduction,ProfileHeatEnergyProduction,ProfileHotWaterProduction
SleepTimeExecutionModel=1440
SleepTimeExecutionPredictionOnTheFly=60
InstallationsToStudy=all
InstallationsToDiscard=
Attributes=Consumption,WeekDay,TempOut,Hour,PreviousValue,PreviousWindowForecastingValue
ExternalWeatherInfo=yes
```

Figure 7.5 config.properties file example.

***WindowForecasting**:* Horizon of the prediction. In the example above, the first value ("31") corresponds to the aggregation value "1440"; therefore, it is a daily prediction for the next 31 days. The second value ("72") corresponds to the aggregation value "60"; thus, it is an hourly prediction for the next 72 hours.

***PredicitonCategory**:* ConsumptionCategories of sensors that will be predicted.

***PredicitonTypes**:* Type of sensors that will be predicted.

***SleepTimeExecutionModel**:* How often the models are updated in minutes.

***SleepTimeExecutionPredictionOnTheFly**:* How often the prediction is performed in minutes.

***InstallationsToStudy**:* Installations in the data warehouse to perform the prediction.

***InstallationsToDiscard**:* Installations discarded to perform the prediction.

***Attributes**:* Attributes used to create the models and perform the predictions.

***ExternalWeatherInfo**:* Use of weather information to train the model and predicts.

The deployment of the AI forecasting models has been done in the cloud. The cloud service has been subcontracted to *Microsoft Azure*. The specifications of the instance are as follows:

- Windows server 2012 R2
- Number of CPU cores: 2
- RAM: 3.5 GB
- Local resource: 496.664 MB (496 GB)

7.3 Dynamic Tariff Comparison and Demand Response Simulation

7.3.1 Functionality

Dynamic tariff comparison and demand response simulation functionality are implemented as VPP functionality. The VPP is a back end calculation engine only. This means that there is no direct interaction between iURBAN tool users and the VPP. Users interact with the VPP via the CDSS GUI.

Based on historical consumption data for one or more city model prosumer objects, the VPP provides following functionality:

- Calculate energy costs for different dynamic tariffs[1] (in the following called *test tariffs*) against a tariff currently in place.
- Account for potential user behavior changes: Simulate how dynamic tariffs could influence consumption patterns of the selected city object(s)—e.g., reduced consumption during high price hours, increased consumption during low price hours (based on T3.3 work).

7.3.2 Stimulus/Response Sequence

The user interacts with the CDSS GUI to initiate tariff comparison and demand response simulation calculations. The user's simulation request is passed to the VPP via the SCDB-CDSS as shown in Figure 7.6.

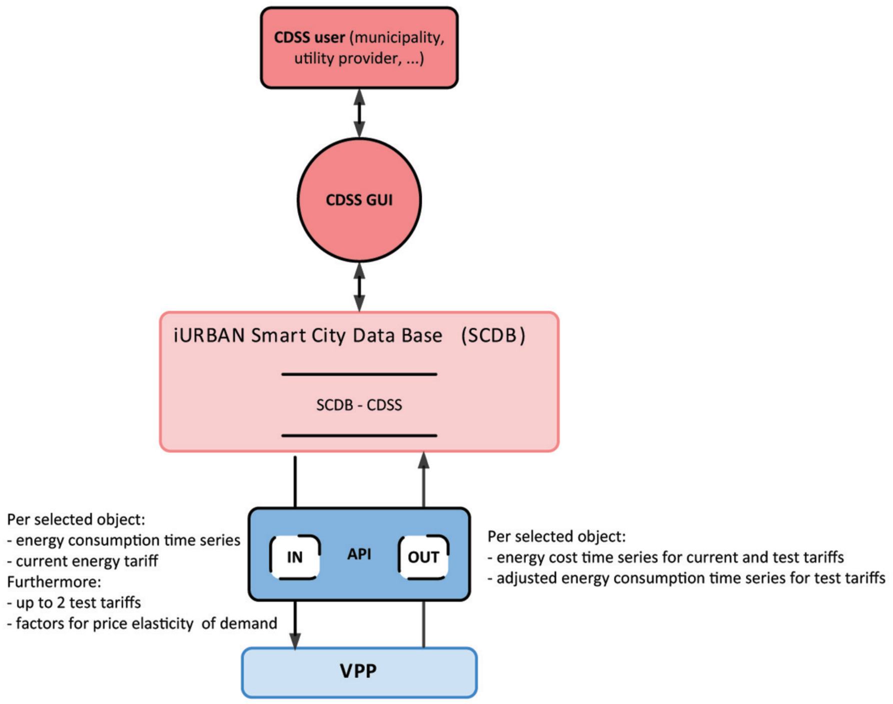

CDSS user (municipality, utility provider, ...)
CDSS GUI
iURBAN Smart City Data Base (SCDB)
SCDB - CDSS
Per selected object:
- energy consumption time series
- current energy tariff
Furthermore:
- up to 2 test tariffs
- factors for price elasticity of demand
IN
API
OUT
Per selected object:
- energy cost time series for current and test tariffs
- adjusted energy consumption time series for test tariffs
VPP

Figure 7.6 Data flow diagram: dynamic tariff comparison and demand response simulation.

[1]Note: In the context of iURBAN, the term "dynamic tariff" simply means that energy cost is a function of time. Dynamic tariffs are defined well in advance (e.g., adjusted only once every X months) and do not change in the short term. Other than dynamic tariffs, "(near-)real-time" tariffs change in the short term (e.g., announced 24 h in advance for the next day based on renewables generation forecast). "(Near-)real-time" tariffs are out of scope of iURBAN.

Per selected prosumer object *i*, the VPP expects the following input data:

- "As is" energy consumption time series (historical data): $cons\_as\ is_i(t)$
- "As is" energy tariff currently in place as a function of time: $tariff\_as\ is_i(t)$.

Optional input data per prosumer object *i*:

- Up to two alternative *test tariffs* selected for comparison $tariff\_test_{i,j}(t)$,
- Three floating point values representing low, medium, and high price elasticity of demand: $elast_i = (elast_{i,\mathrm{low}},\ elast_{i,\mathrm{medium}},\ elast_{i,\mathrm{high}})$.

If energy consumption or tariff data are missing for any of the selected city objects, the VPP does not carry out a calculation and returns an error message.

If all required data are available, the VPP carries out calculations and provides the following results data per selected prosumer object *i*:

- "As is" energy consumption time series: $cons\_as\ is_i(t)$
- "As is" energy cost time series: $cost\_as\ is_i(t)$
- If *test tariff(s)* (up to 2) have been supplied, per test tariff:
    - 3 energy consumption time series according to low, medium, and high price elasticity of demand
        - $cons\_test\_low_i(t)$, $cons\_test\_medium_i(t)$, $cons\_test\_high_i(t)$
    - 4 energy consumption cost time series according to none, low, medium, and high price elasticity of demand
        - $cost\_test\_none_{i,j}(t)$, $cost\_test\_low_{i,j}(t)$, $cost\_test\_medium_{i,j}(t)$, $cost\_test\_high_{i,j}(t)$

7.3.3 User Workflow

1. **CDSS** user actions:
    a. The user opens the CDSS GUI tariffs analysis module.
    b. The user selects a commodity of interest (e.g., electricity or gas).
    c. The user selects one or more prosumer object, i.e., buildings or other city objects of interest.
    d. For each prosumer object selected, the user assigns
        i. The current tariff (note: The CDSS is to provide a tariff database so that a user can easily select tariffs previously defined).

ii. 0 to 2 alternative tariffs (*test tariffs*) (note: To speed up the process, the CDSS GUI probably provides functionality to assign the same tariff to all selected objects).

iii. Optional: Values for low, medium, and high elasticity of demand (note: The CDSS GUI provides default values per commodity).

e. The user saves the simulation request under a specific name.

2. User input is stored in the **SCDB-CDSS** and parsed to the VPP from the SCDB-CDSS.
3. **VPP** actions:

   a. A new calculation job is created and added to the job queue.
   b. The simulation job is carried out.
   c. Simulation job results parsed back to the SCDB-CDSS.

4. The simulation results are stored in the **SCDB-CDSS**.
5. **CDSS user** actions:

   a. The user analyses simulation results.
   b. The user can select to see aggregated results for a selected group of objects. Aggregation calculations required for this are carried out by the CDSS.

7.3.4 Calculation Methodology

7.3.4.1 Price elasticity background

Price elasticity of demand is a simple concept used in economics to estimate the change of demand of a certain good if its price changes.

It is defined as follows:

$$\text{Price elasticity } = \frac{\%\triangle \text{ Quantity Demanded}}{\%\triangle \text{ Price}}$$

University of Freiburg carried out research on how well this concept can be applied in the context of energy demand.

Suggested reasonable ranges for elasticity factors for electricity and gas:

	Residential Electricity	Commercial Electricity	Residential Natural Gas
Short-run elasticity	–0.24	–0.21	–0.12
Long-run elasticity	–0.32	–0.97	–0.36

Absolute values of long-run elasticity were found to be higher than short-run ones. This indicates that users seem to require some time to adjust to different prices.

Note 1: When applying elasticity factors, the total consumption changes:

$$total_{consumption\left(real_{tariff}\right)} \neq total\_consumption(test\_tariff).$$

This is in agreement with real-world observations: Customer DR initiated by flexible tariffs has both load shifting and energy saving/"wasting" components:

- Load shifting: Equipment is used at a different time of the day, but in total, the same amount of energy is consumed.
- Energy saving: Consumption is reduced, e.g. by dimming light intensity during high-price periods.
- Energy "wasting": Consumption is increased, e.g., do not switch off the light in non-occupied rooms because electricity is cheap anyway.

7.3.4.2 Dynamic tariff comparison and demand response formula

If all required data are available, the VPP carries out calculations and provides the following results data per selected prosumer object *i*:

Demand response:

"As is" consumption

$$= cons\_as\ is_i(t)$$

Furthermore, if test tariffs (up to 2) have been provided consumption time series adjusted by price elasticity (low, medium, high) of demand:

$$cons\_test\_low_{i,j}(t)$$
$$= \left(\left(elast_{\ i,low} \times \left(\frac{tariff\_as\_is_i(t)\ -\ tariff\_test_{i,j}(t)}{tariff\_test_{i,j}(t)}\right)\right)\right.$$
$$\left.\times\ cons\_as\ is_i(t)\right) + cons\_as\ is_i(t)$$

$$cons\_test\_medium_{i,j}(t)$$
$$= \left(\left(elast_{\ i,medium} \times \left(\frac{tariff\_as\_is_i(t)\ -\ tariff\_test_{i,j}(t)}{tariff\_test_{i,j}(t)}\right)\right)\right.$$
$$\left.\times\ cons\_as\ is_i(t)\right) + cons\_as\ is_i(t)$$

$$cons\_test\_high_{i,j}(t) = \left(\left(elast_{\ i,high} \times \left(\frac{tariff\_as\_is_i(t)\ -\ tariff\_test_{i,j}(t)}{tariff\_test_{i,j}(t)}\right)\right) * cons\_as\ is_i(t)\right) + cons\_as\ is_i(t)$$

Dynamic tariff:
"As is" energy cost:

$$cost\_as\ is_i(t) = cons\_as\ is_i(t) \times tariff\_as\ is_i(t)$$

Furthermore, if *test tariffs* (up to 2) have been provided:

$$\begin{aligned} cost\_test\_none_{i,j}(t) &= cons\_as\ is_i(t) \times tariff\_test_{i,j}(t) \\ cost\_test\_low_{i,j}(t) &= cons\_test\_low_{i,j}(t) \times tariff\_test_{i,j}(t) \\ cost\_test\_medium_{i,j}(t) &= cons\_test\_medium_{i,j}(t) \times tariff\_test_{i,j}(t) \\ cost\_test\_high_{i,j}(t) &= cons\_test\_high_{i,j}(t) \times tariff\_test_{i,j}(t) \end{aligned}$$

7.3.5 Assumptions and Limitations

- In contrast to other VPP calculations, tariff comparison and DR simulation calculations are carried out for historical data only (considering forecast data as well would not provide extra value).
- The user may not define the time span for tariff analysis: Per default, up to one year of historical data is analyzed.
- Tariff analysis and DR calculations are to be carried out on prosumer object level only. This means that no aggregation to a higher level or re-calculation of VPP network status based on changed consumption patterns of city objects is required.
- The VPP shall provide some kind of job batch queue mechanism to allow a user to schedule several tariff comparison and demand response simulation runs. This implies that different tariff comparison and demand response jobs are carried out independently from each other. If a user requests to store results of more than one tariff comparison and DR simulation at a time, this needs to be implemented as functionality of the system connecting to the VPP (SCDB-CDSS).
- Tariff comparison and DR simulation consider energy consumption only; energy generation is not considered.[2]

[2] One reason for this decision is that feed-in tariff structures are comparatively complex in iURBAN demo cities. The project consortium agreed to focus on energy consumption only.

- The dynamic tariff data model only needs to support tariffs of a format which can be expressed as function of time (=time series). Aspects such as standing charges and capacity charges are not supported.

7.4 Conclusions

This chapter has presented the two main smart algorithms within iURBAN. Prediction of consumption and production of energy were demanded by households as a must feature, receiving good feedback about its representation through the graphical user interface.

Variable tariff and demand response simulation capabilities have been describing initial assumptions as well as a use case showing the expected iteration of end users with VPP (the tool which integrates the algorithms) through the CDSS interface.

References

[1] Khamis, M. F. I., Baharudin, Z., and Hamid, N. H. (2011). Electricity forecasting for small scale power system using artificial neural network. *Power Engineering and Optimization Conference (PEOCO) 5th International*, pp. 54–59.

[2] Marvuglia, A., and Messineo, A. (2012). Using recurrent artificial neural networks to forecast household electricity consumption. *Energy Procedia*, 14, 45–55.

[3] Tranchita, C., and Torres, A. (2004). Soft computing techniques for short term load forecasting. *Power Syst. Conf. Expo.*, 1, 497–502.

[4] Caruana, R., and Niculescu-Mizil, A. (2006). An empirical comparison of supervised learning algorithms. *Proceedings of the 23rd International Conference on Machine Learning (ICML2006)*, pp. 161–168.

[5] Nesreen, A. K., Amir, A. F., Neamat, G., and Hisha, E. H. (2010). An empirical comparison of machine learning models for time series forecasting. *Econometric Rev.,* 29, 594–621.

[6] http://www.forecasters.org/data/m3comp/m3comp.htm

[7] Tso, G. K. F., and Yau, K. K. W. (2007). Predicting electricity energy consumption: A comparison of regression analysis, decision tree and neural networks. *Energy*, 32, 1761–1768.

[8] Jurado, S., Nebot, A., and Mugica F. (2015). Hybrid methodologies for electricity load forecasting: Entropy-based feature selection with machine learning and soft computing techniques. *Energy*, 86, 276–291.

[9] Breiman, L. (2001). Random forests. *Machine Learning*, 45 (1), 5–32.

[10] Breiman, L. (1996). Bagging predictors. *Machine Learning*, 24 (2), 123–140.

[11] Kleinberg, E. (1996). An overtraining-resistant stochastic modeling method for pattern recognition. *Annals of Statistics* 24 (6), 2319–2349.

[12] Li, Y., Wang, S., and Ding, X. (2010). Person-independent head pose estimation based on random forest regression. *17th IEEE International Conference on Image Processing (ICIP)*, pp. 1521–1524.

[13] McCulloch, W., and Pitts, W. (1943). A logical calculus of ideas immanent in nervous activity. *Bullet. Mathematic. Biophy.*, 5(4), 115–133.

[14] Rosenblatt, F. (1958). The perceptron: A probalistic model for information storage and organization in the brain. *Psychol. Rev.*, 65(6), 386–408.

[15] Alon, I., Qi, M. and Sadowski, R. J. (2001). Forecasting aggregate retail sales: A comparison of artificial neural networks and traditional methods. *J. Retail. Consum. Serv.*, 8, 147–156.

[16] White, H. (1990). Connectionist nonparametric regression: multilayer feed forward networks can learn arbitrary mappings. *Neural Networks*, 3, 535–549.

[17] Chow, T. W. S., and Cho, S. Y. (2007). Neural Networks and Computing: Learning Algorithms and Applications. 7, 14–15.

[18] Klir, J., and Elias, D. (2002). *Architecture of systems problem solving*, 2nd. Edn. Plenum Press, New York.

[19] Nebot, A., Mugica, F., Cellier, F., and Vallverdú, M. (2003). Modeling and simulation of the central nervous system control with generic fuzzy models. *Trans. Society Model. Simulat.,* 79(11), 648–669.

[20] Escobet, A., Nebot, A., and Cellier, F. E. (2008). Visual-FIR: A tool for model identification and prediction of dynamical complex systems. *Simulat. Modell. Practice Theory*, 16, 76–92.