(FP7 614100)

D5.2 Data Analysis and Forecast for Energy Consumption

29 July 2014 – Version 1.0

Published by the IMPReSS Consortium

Dissemination Level: Public

Project co-funded by the European Commission within the 7th Framework Programme and the Conselho Nacional de Desenvolvimento Científico e Tecnológico
Objective ICT-2013.10.2 EU-Brazil research and development Cooperation

Document control page

Document file: D5.2 Data Analysis and Forecast for Energy Consumption.docx
Document version: 1.0
Document owner: Eduardo Souto (UFAM)

Work package: WP5 – Data Storage, Analysis & Decision Support
Task: T5.2 Data Analysis and forecast for energy consumption
Deliverable type: P

Document status: [X] approved by the document owner for internal review
[X] approved for submission to the EC

Document history:

Version	Author(s)	Date	Summary of changes made
0.1	Eduardo Souto (UFAM)	13/07/2014	First Draft.
0.2	Thiago Rocha(UFAM)	15/07/2014	Second Draft.
0.3	Weslen Sousa(UFAM)	17/07/2014	Third Draft.
0.4	Eulanda Santos(UFAM)	18/07/2014	Fixed Data Mining and Machine Learning concepts.
0.5	Francisco Ivan(UFAM)	19/07/2014	Fixed Optimization concepts.
0.6	Lucas Gomes (UFPE)	21/07/2014	Fixed typos and revised the document.
1.0	Eduardo Souto(UFAM)	29/07/2014	Final version submitted.

Internal review history:

Reviewed by	Date	Summary of comments
Carlos Kamienski (UFABC)	26/07/2014	Accepted with minor corrections and comments
Ferry Pramudianto (FIT	28/07/2014	Accepted with minor corrections and comments

Index:

1. Executive summary

This deliverable focuses on showing how developers can use the IMPReSS platform to get insight from the massive amount of data acquired from smart objects connected to the IMPReSS Cloud. By the means of simulation scenarios, users will be capable of analyzing information provided by those smart objects, in order to extract knowledge from them.

Different statistic, optimization, data mining and machine learning tools may be used to help developers on extracting knowledge. First, this deliverable will show a general overview to highlight how IMPRESS will provide these strategies to the developers. Thus, it will focus on the statistics tools, since the optimization, data mining and machine learning tools will be discussed in the next deliverables.

In order to reach these goals, this deliverable is divided as follows. The first section discusses some concepts about statistics, optimization, data mining and machine learning algorithms. Then, sections two and three present the extended version of the API architecture for the Data Analytics module. Finally, section four shows some test case scenarios which were executed in order to lighten the developers. These scenarios are provided to help the developers to understand how they will use the Data Analytics module API, which is part of the IMPReSS API.

2. Concepts

This section discusses the main concepts employed in this deliverable. These concepts were used to understand and develop an extended version of the Data Analytics Application Programming Interface (API), which was first proposed in deliverable 8.2 [1]. First, data mining and machine learning concepts will be shown, then optimization and finally statistics.

2.1. Data Mining and Machine Learning

Extracting patterns and knowledge from large amounts of information is an important step for the solution of several tasks, such as remote sensing, financial forecasting, intrusion detection and fraud, information retrieval on the Internet, filtering spam emails, among others. Solutions proposed to solve such tasks are usually based on two inter-related fields: Machine Learning and Data Mining.

Data mining focuses on the discovery of (previously) unknown properties in the data, such as the analysis step of knowledge discovery in databases, while machine learning focuses on prediction, based on known properties learned from the training data. Machine Learning techniques can be divided into two groups: supervised and unsupervised. Supervised learning algorithms are most used to forecast a value and require a definition of a known outcome or target for the scenario. The objective of supervised learning is classification, for discrete target values, or regression, for continuous output. Unsupervised learning algorithms are usually used to recognize a model structure that is relevant in a set of data [2]. Clustering algorithm are examples of unsupervised learning methods. The Data Analytics module API will provide several classification, regression, association and cluster algorithms.

2.1.1. Classification

Handwritten digits recognition, business modeling, and credit analysis are examples of classification problems. As a type of supervised learning, a classification algorithm builds a model that is used to assign labels to the unknown examples. The data used to generate the models may include energy consumption information such as cost and time to allow the prediction of classes of behaviors. The input or training data for a supervised learning algorithm requires the presence of attributes to represent each training sample, as well the classes assigned to the training samples [3].

2.1.2. Regression

Regression algorithms are tools employed to predict a number as the output of a problem. Profit, sales, mortgage rates, house values, square footage, temperature or distance could all be predicted using regression techniques [4]. For example, a regression model could be used to predict the energy cost of a house based on number of rooms, number of equipment and other factors.

2.1.3. Cluster

Clustering techniques try to identify natural clusters of data objects according to a similarity measure, for instance Euclidian distance. The members of the same cluster are more similar to each other than they are in comparison with members of other clusters. The goal of clustering analysis is to find high-quality clusters such that the inter-cluster similarity is low and the intra-cluster similarity is high [4].

2.1.4. Association

Association strategies are used to identify some attribute value conditions that occur frequently together in a given set of data. Association analysis is widely used in transaction data analysis for directed marketing, catalog design, and other business decision-making process. Traditionally, association is used for market basket data analysis such as 90% of the people who buy milk also buy bread [4].

2.2. Optimization

Optimization consists on the effort of maximizing or minimizing an objective function by systematically choosing input values from within a desired range. And, thereafter, compute the value of the objective function. The main issue in optimization applications is to solve a problem finding a solution or a set of solutions with the most effective objective values (the lowest value in minimization problems or the highest value in maximization problems) under some given constraints [5]. The Data Analytics module API will provide a set of optimization algorithms to solve problems in energy consumption scenarios. Since these scenarios will typically involve a large initial amount of data, metaheuristics should be used due to the exponential complexity of an exhaustive search.

2.2.1. Metaheuristic

The objective of a heuristic is to produce a solution in a reasonable time that is good enough for solving the problem at hand. The chosen solution may not be the best among all possible solutions for the problem [6], i.e. there is no guarantee that the optimal solution will be found. However, metaheuristics traditionally employ strategies in order to create a process capable of escaping local minima and perform robust search in the solution space of a problem, thus seeking solutions very close to the global optimum solutions [7]. Greedy Search, Genetic Algorithms, Simulated Annealing and Particle Swarm Optimization are examples of the most popular metaheuristics. Evolutionary and Swarm optimization algorithms are described below.

2.2.2. Stochastic Algorithms

Stochastic algorithms are predominantly global optimization algorithms and metaheuristics that manage the application of local search procedures. This set of algorithms provides various different strategies to allow the generation of a variety of starting points in order to employ local search for refinement. This process is repeated leading to unexplored or potentially better areas of the search space [8]. Some examples of algorithms of this group are Random search, Adaptive Random Search, Stochastic Hill Climbing, Iterated Local Search, among others.

2.2.3. Evolutionary Algorithms

Evolutionary Algorithms belong to the Evolutionary Computation field of study and is concerned with computational methods inspired by the process and mechanisms of biological evolution. Similarly, as the process of evolution by means of natural selection proposed by Darwin. The mechanisms of evolution describe how evolution actually takes place through the modification and propagation of genetic material. Evolutionary Algorithms are concerned with investigating computational systems that resemble simplified versions of the processes and mechanisms of evolution toward achieving

the effects of these processes and mechanisms, namely the development of adaptive systems [8]. Therefore, Evolutionary Algorithms mimic nature's evolutionary principles to conduct its search towards an optimal solution, even though the optimal solution may not be achieved.

2.2.4. Swarm Algorithms

Swarm intelligence is the study of computational systems inspired by the 'collective intelligence'. Collective Intelligence emerges through the cooperation of large numbers of homogeneous agents in the environment. Examples include schools of fish, flocks of birds, and colonies of ants. Such intelligence is decentralized, self-organizing and distributed through out an environment. In nature, such systems are commonly used to solve problems such as e effective foraging for food, prey evading, or colony re-location [8]. Particle Swarm Optimization (PSO), for instance, simulates the behaviors of bird flocking or fish schooling. Each individual of the population is called particle. All particles have fitness values which are evaluated during the optimization process towards finding the best particle in the whole population.

2.2. Statistics

Statistics is the study of the collection, organization, analysis, interpretation and presentation of data. When analyzing data, it is possible to use several statistics methods [9]. The reason to use statistics is to determine what information is reliable and which predictions can be trusted. The data stored has particular representation and relevant significance. Therefore, it is possible to apply statistic functions aiming to represent data from an energy consumption scenario, which may be used, for instance, to predict consumption or generate metrics to apply rules to solve a problem. The Data Analytics module API will provide several statistical methods.

3. Architecture

This section shows an overview of the Data Analytics module API. The main goal of this API is to provide to the developers tools related to data analytics, precisely statistics, optimization, data mining and machine learning algorithms. This API will be part of the IMPRESS Cloud and will be available through a web service. Figure 3.1 shows the Data Analytics module API Overview.

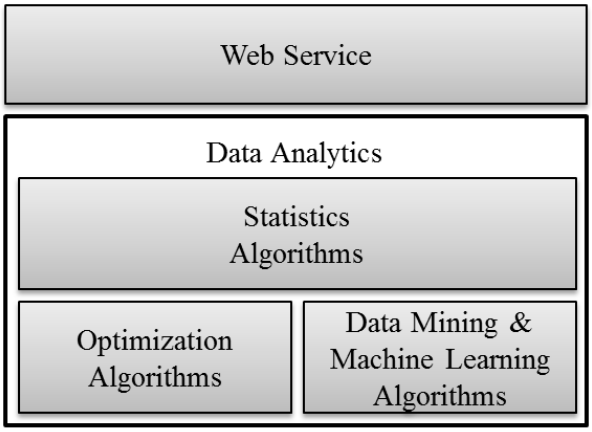

Figure 3.1. Data Analytics API Overview.

There is a web service, which implements the Data Analytics module API, in order to allow developers to have access to the Data Analytics module API. Deliverable 8.2 presents details about how the web services URLs should be formatted in order to guarantee consistency among IMPReSS' different modules. The second part of Figure 3.1 shows how the data analytics process is divided at the IMPReSS Platform. Data Analytics entails the process of examining large amounts of data to uncover hidden patterns, unknown correlations and other useful information that can be used to generate better decisions. With big data analytics, data scientists and other users can analyze huge volumes of data that conventional analytics and business intelligence solutions cannot do. This API offers four groups of algorithms to analyze raw data: statistics, optimization, data mining and machine learning algorithms. These groups are better depicted in Figure 3.2.

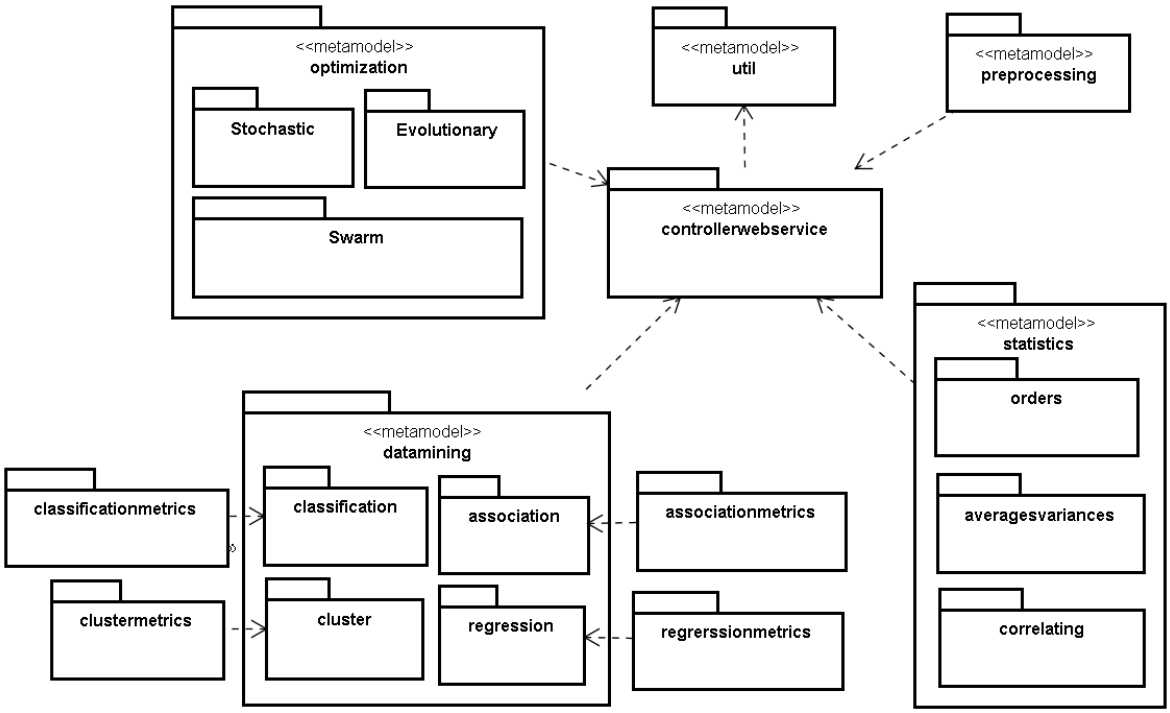

Figure 3.2. Data Analytics module API package structure.

The design of this API follows patterns from the Java Specification Request (JSR) [10] through the creation of packages. Furthermore, data mining algorithms were taken from the scikit-learn library [11], statistics algorithms from pymc library [12] and optimization algorithms from Clever Algorithms [8].

Before describing data mining, optimization and statistics, there are other packages that compose the Data Analytics module API. The first one is the **impress.util** package that consists of some utility classes which may be used to assist the other packages, for example, a class can be used to call a method to format data. The second package is the **impress.controllerwebservice** that has classes to expose the Data Analytics module API through a web service and makes it available to developers. Deliverable 8.2 has more details about the web service specification. The last package is the **impress.preprocessing** that consists of several classes to allow data preprocessing work before any data analytics algorithm is called. **Impress.preprocessing** algorithms are shown in Table 3.1. The preprocessing step may involve several strategies, for instance, when there are data with missing features, imputation methods should be employed to substitute missing values with meaningful estimates in order to allow machine learning algorithms to be used. Also, when data is available in different scales, normalization and standardization preprocessing should be used to allow the effective use of learning methods based on distance measures, such as Euclidian distance.

Table 3.1: Preprocessing algorithms.

Classes	Description
Binarizer([threshold, copy])	Binarize data (set feature values to 0 or 1) according to a threshold
Imputer([missing_values, ...])	Imputation methods for completing missing values
KernelCenterer	Center a kernel matrix
LabelBinarizer([neg_label, ...])	Binarize labels in a one-vs-all fashion
LabelEncoder	Encode labels with value between 0 and n_classes-1.
MinMaxScaler([feature_range, copy])	Standardizes features by scaling each feature to a given range.
Normalizer([norm, copy])	Normalize samples individually to unit norm
StandardScaler([copy, ...])	Standardize features by removing the mean and scaling to unit variance
binarize(X[, threshold, copy])	Boolean thresholding of array-like or scipy.sparse matrix

label_binarize(y, classes[, ...])	Binarize labels in a one-vs-all fashion
normalize(X[, norm, axis, copy])	Normalize a dataset along any axis
scale(X[, axis, with_mean, ...])	Standardize a dataset along any axis

Figure 3.3 consists of several data mining and machine learning algorithms, as well as metrics that each group uses to evaluate their algorithms. These algorithms and metrics will be explained below:

- **impress.datamining.classification**: This sub-package defines classes that implements classification algorithms.
- **impress.datamining.association:** This sub-package defines classes that implements association algorithms.
- **impress.datamining.cluster:** This sub-package defines classes that implements clustering algorithms.
- **impress.datamining.regression:** This sub-package defines classes that implements regression algorithms.
- **impress.datamining.classification.metrics:** This sub-package defines metrics classes to evaluate data mining classification algorithms.
- **impress.datamining.association.metrics:** This sub-package defines metrics classes to evaluate association algorithms.
- **impress.datamining.cluster.metrics:** This sub-package defines metrics classes to evaluate data mining clustering algorithms.
- **impress.datamining.regression.metrics:** This sub-package defines metrics classes to evaluate data mining regression algorithms.

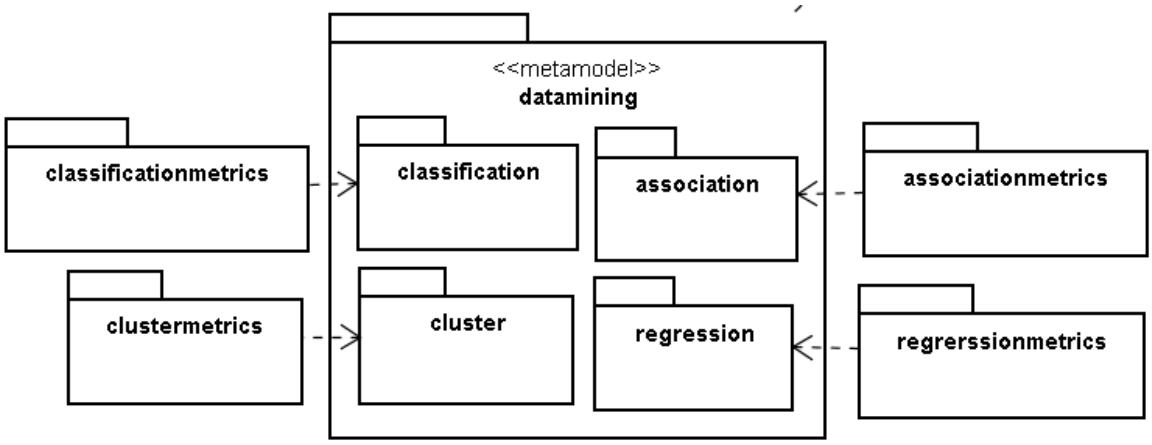

Figure 3.3. Data Mining package structure.

The algorithms of each package cited above are shown in the tables bellow:

Table 3.2: Classification algorithms.

Classes	Description
RandomForestClassifier([...])	Ensemble of decision tree classifiers generated by random forest.
RandomTreesEmbedding([...])	Ensemble of totally random trees.
AdaBoostClassifier(...[, criterion])	Ensemble of classifiers generated by the AdaBoost ensemble generation method.
GradientBoostingClassifier([loss, ...])	Ensemble of classifiers generated by the Gradient Boosting method.
GaussianHMM([n_components, ...])	The Hidden Markov Model classifier generated with Gaussian emissions

GaussianNB	The Naive Bayes Classifier generated with Gaussian functions
NearestCentroid([metric, ...])	The prototype-based Nearest centroid classifier.
SVC([C, kernel, degree, gamma, coef0, ...])	The Support Vector Machines classifier C-Support Vector Classification for nonlinear separated problems.
LinearSVC([penalty, loss, dual, tol, C, ...])	Linear Support Vector Classification for linear separated problems.
DecisionTreeClassifier([criterion, ...])	A decision tree classifier.

Table 3.3: Evaluation metrics for classification algorithms.

Classes	Description
accuracy_score(y_true, y_pred[, ...])	Accuracy classification score.
average_precision_score(y_true, y_score)	Compute average precision (AP) from prediction scores
classification_report(y_true, y_pred)	Build a text report showing the main classification metrics
confusion_matrix(y_true, y_pred[, ...])	Compute confusion matrix to evaluate the accuracy of a classification
jaccard_similarity_score(y_true, y_pred)	Jaccard similarity coefficient score
log_loss(y_true, y_pred[, eps, ...])	Log loss, aka logistic loss or cross-entropy loss.
precision_recall_curve(y_true, ...)	Compute precision-recall pairs for different probability thresholds
precision_score(y_true, y_pred[, ...])	Compute the precision
recall_score(y_true, y_pred[, ...])	Compute the recall
roc_auc_score(y_true, y_score)	Compute Area Under the Curve (AUC) from prediction scores

Table 3.4: Clustering algorithms.

Classes	Description
DBSCAN([eps, min_samples, metric, ...])	Perform DBSCAN clustering from vector array or distance matrix.
KMeans([n_clusters, init, n_init, ...])	The simple K-Means clustering algorithm
MiniBatchKMeans([n_clusters, init, ...])	Mini-Batch K-Means clustering algorithm
MeanShift([bandwidth, seeds, ...])	MeanShift clustering algorithm
Ward([n_clusters, memory, ...])	Ward hierarchical clustering: constructs a tree and cuts it.
NearestNeighbors([n_neighbors, ...])	Unsupervised learner for implementing neighbor searches.

Table 3.5: Evaluation metrics for Clustering algorithms.

Classes	Description
adjusted_mutual_info_score(...)	Adjusted Mutual Information between two clusters
adjusted_rand_score(labels_true, ...)	Rand index adjusted for chance
completeness_score(labels_true, ...)	Completeness metric of a cluster labeling given a ground truth
homogeneity_score(labels_true, ...)	Homogeneity metric of a cluster labeling given a ground truth
mutual_info_score(labels_true, ...)	Mutual Information between two clusters
normalized_mutual_info_score(...)	Normalized Mutual Information between two clusters
consensus_score(a, b[, similarity])	The similarity of two sets of biclusters.

Table 3.6: Regression algorithms.

Classes	Description
RandomForestRegressor([...])	Ensemble of random forest regressor.
AdaBoostRegressor(...[, criterion, ...])	Ensemble based on AdaBoost regressor.
GradientBoostingRegressor([loss, ...])	Gradient Boosting for regression.
IsotonicRegression([y_min, y_max, ...])	Isotonic regression model.
LinearRegression([...])	Ordinary least squares Linear Regression.
LogisticRegression([penalty, ...])	Logistic Regression classifier.
RandomizedLogisticRegression([...])	Randomized Logistic Regression
KNeighborsRegressor([n_neighbors, ...])	Regression based on k-nearest neighbors.
RadiusNeighborsRegressor([radius, ...])	Regression based on neighbors within a fixed radius.
SVR([kernel, degree, gamma, coef0, tol, ...])	epsilon-Support Vector Regression.

Table 3.7: Evaluation metrics for Regression algorithms.

Classes	Description
explained_variance_score(y_true, y_pred)	Explained variance regression score function
mean_absolute_error(y_true, y_pred)	Mean absolute error regression loss
mean_squared_error(y_true, y_pred)	Mean squared error regression loss
r2_score(y_true, y_pred)	R^2 (coefficient of determination) regression score function.

Package **impress.optimization** consists of several optimization algorithms, such as Tabu Search, Evolutionary Algorithms and Particle Swarm Optimization, as depicted in Figure 3.4 and explained below.

- **impress.optimization.stochastic:** This sub-package defines classes that implements stochastic optimization algorithms.
- **impress.optimization.evolutionary:** This sub-package defines classes that implements evolutionary optimization algorithms.
- **impress.optimization.swarm:** This sub-package defines classes that implements swarm optimization algorithms

The algorithms of each of these packages are shown in the tables bellow:

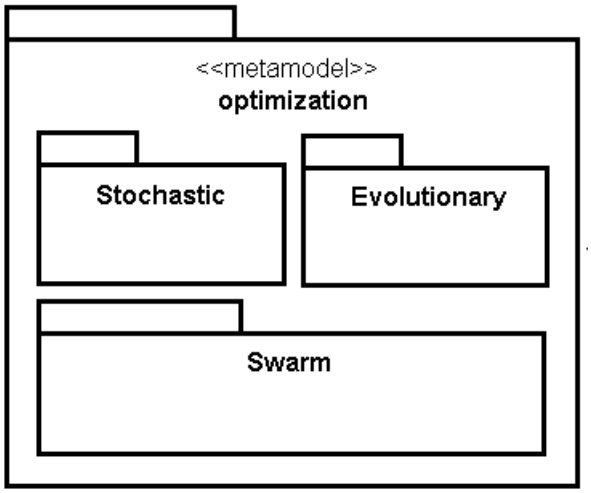

Figure 3.4. Data analytics module optimization package structure.

Table 3.8: Stochastic algorithms.

Classes	Description
tabusearch(<parameters>)	Tabu Search is a Global Optimization algorithm and a Metaheuristic or Meta-strategy for controlling an embedded heuristic technique.
randomsearch(<parameters>)	Random search belongs to the elds of Stochastic Optimization and Global Optimization.
scattersearch(<parameters>)	Scatter search is a Metaheuristic and a Global Optimization algorithm.

Table 3.9: Evolutionary algorithms.

Classes	Description
geneticalgorithm(<parameters>)	The Genetic Algorithm is an Adaptive Strategy and a Global Optimization technique.
nsga(<parameters>)	The Non-dominated Sorting Genetic Algorithm is a Multiple Objective Optimization (MOO) algorithm and is an instance of an Evolutionary Algorithm from the field of Evolutionary Computation.
spea(<parameters>)	Strength Pareto Evolutionary Algorithm is a Multiple Objective Optimization (MOO) algorithm and an Evolutionary Algorithm from the field of Evolutionary Computation.

Table 3.10: Swarm algorithms.

Classes	Description
pso(<parameters>)	Particle Swarm Optimization belongs to the field of Swarm Intelligence and Collective Intelligence and is a sub-field of Computational Intelligence.
beesalgorithm(<parameters>)	The Bees Algorithm beings to Bee Inspired Algorithms and the field of Swarm Intelligence, and more broadly the elds of Computational Intelligence and Metaheuristics.
antcolonysystem(<parameters>)	The Ant Colony System algorithm is an example of an Ant Colony Optimization method from the field of Swarm Intelligence, Metaheuristics and Computational Intelligence.

Package **impress.statistics** consists of several statistics algorithms, such as orders, average, variance and correlating, as depicted in Figure 3.5 and explained below.

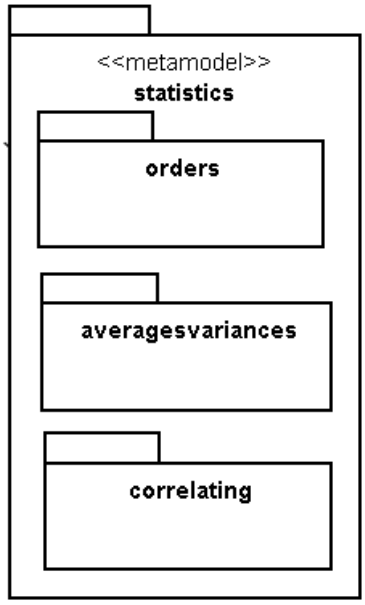

Figure 3.5. IMPRESS statistic package structure.

- **impress.statistics.orders**: This sub-package defines classes that implement the extraction of basics information, i.e. minimum and maximum.
- **impress.statistics.averagevariances**: This sub-package defines classes that implement the extraction of average and variances of the data.
- **impress.statistics.correlating:** This sub-package defines classes that implement the extraction of correlation coefficients.

The algorithms of each package are shown in the tables bellow:

Table 3.11: Order algorithms.

Classes	Description
amin(a[, axis, out, keepdims])	Returns the minimum of an array or minimum along an axis.
amax(a[, axis, out, keepdims])	Returns the maximum of an array or maximum along an axis.
nanmin(a[, axis, out, keepdims])	Returns minimum of an array or minimum along an axis, ignoring any NaNs.
nanmax(a[, axis, out, keepdims])	Returns the maximum of an array or maximum along an axis, ignoring any
ptp(a[, axis, out])	Range of values (maximum - minimum) along an axis.
percentile(a, q[, axis, out, overwrite_input])	Compute the qth percentile of the data along the specified axis.

Table 3.12: Average/Variance algorithms.

Classes	Description
median(a[, axis, out, overwrite_input])	Computes the median along the specified axis.
average(a[, axis, weights, returned])	Computes the weighted average along the specified axis.
mean(a[, axis, dtype, out, keepdims])	Computes the arithmetic mean along the specified axis.

std(a[, axis, dtype, out, ddof, keepdims])	Computes the standard deviation along the specified axis.
var(a[, axis, dtype, out, ddof, keepdims])	Computes the variance along the specified axis.
nanmean(a[, axis, dtype, out, keepdims])	Computes the arithmetic mean along the specified axis, ignoring NaNs.
nanstd(a[, axis, dtype, out, ddof, keepdims])	Computes the standard deviation along the specified axis, while
nanvar(a[, axis, dtype, out, ddof, keepdims])	Computes the variance along the specified axis, while ignoring NaNs.

Table 3.13: Correlating algorithms.

Classes	Description
corrcoef(x[, y, rowvar, bias, ddof])	Returns correlation coefficients.
correlate(a, v[, mode, old_behavior])	Cross-correlation of two 1-dimensional sequences.
cov(m[, y, rowvar, bias, ddof])	Estimates a covariance matrix, given data.

The next section focuses on some scenarios to show to the developers how they can use the Data Analytics module API to use any of the aforementioned algorithms. These scenarios, though, are focused on statistics algorithms. Optimization, data mining and machine learning algorithms will be worked with details in the next deliverables of WP5.

4. Use Cases

The potential applications of Data Analytics module API are wide-ranging. This section covers two use cases to show the usage of the statistical capabilities of the aforementioned API. A scenario with a database populated with real data based on a study from Massachusetts Institute of Technology (MIT) [13] was chosen to address the use of some data analytics features focused at the statistical algorithms such as standard deviation, mean, maximum, minimum, among others.

4.1. Scenario

The scenario is composed by 77 sensors installed in an apartment to collect data about human activity everyday. For the purpose of this scenario, only data related to appliance energy consumption was collected. Figure 4.1 has a preview of the dataset created from the data collected.

id integer	name character var	model character var	consumption numeric	priority integer	start_time bigint	end_time bigint	duration bigint	locations character var	day character var
35	Dryer Cloth	BSR10 10 kg	4800	4	31483	3895	-27588	Dinning roc	monday
26	Dishwasher	Brastemp At	780	3	31483	3895	-27588	Dinning roc	thursday
36	Toaster	Daily Ri25	800	3	31483	3895	-27588	Bathroom	saturday
25	washes clot	Brastemp At	300	1	31483	3895	-27588	Living roon	wedsnesday
35	Dryer Cloth	BSR10 10 kg	4800	4	31483	3895	-27588	Bathroom	sunday
20	Cooker	Electrolux	1180	4	31483	3895	-27588	Living roon	monday
36	Toaster	Daily Ri25	800	3	31483	3895	-27588	Bathroom	friday
28	Sewing mach	Singer Bril	70	1	31483	3895	-27588	Dinning roc	tuesday
35	Dryer Cloth	BSR10 10 kg	4800	4	31483	3895	-27588	Dinning roc	saturday
5	Air conditi	Split BBV0	822	3	31483	3895	-27588	Kitchen	monday
36	Toaster	Daily Ri25	800	3	31483	3895	-27588	Bathroom	thursday
24	Table lamp	Boyu	30	1	31483	3895	-27588	Kitchen	sunday
35	Dryer Cloth	BSR10 10 kg	4800	4	31483	3895	-27588	Dinning roc	thursday
20	Cooker	Electrolux	1180	4	31483	3895	-27588	Living roon	sunday
36	Toaster	Daily Ri25	800	3	31483	3895	-27588	Bathroom	wedsnesday
17	Electric fr	Britania Pr	90	1	31483	3895	-27588	Living roon	saturday
35	Dryer Cloth	BSR10 10 kg	4800	4	31483	3895	-27588	Dinning roc	friday

Figure 4.1. Dataset from appliance energy consumption.

The attributes from the dataset are explained below:

- **id**: Appliance identification id.
- **name**: Appliance classification by name.
- **model:** Appliance model identification.
- **Consumption_on:** Consumption about each appliance.
- **priority:** Priority definition for power consumption.
- **start_time:** Time at which the activity started in seconds.
- **end_time:** Time at which the activity ended in seconds.
- **duration:** Difference between start_time and end_time.
- **locations:** Appliance location.
- **day:** Number between 1 and 7 specifying the day of the week (1=monday, 7=sunday).

4.1. First use case

The first use case predicts the energy mean consumption per room for the next day, which is useful to know the average consumption and get a sense of how much this consumption is going to cost. To calculate this, the "mean" function from the Data Analytics module has to be called through the URL below.

*http://<domain>/statistics/averagesvariances/mean/***listnumber?values=[a,b,c]**

This URL creates a request to the IMPReSS Platform to execute the mean function. Their parameters are explained in Table 4.1.

Table 4.1: Mean function parameters

Parameters:	**values** *: array_like* Array containing numbers whose mean is desired. If *a* is not an array, a conversion is attempted.
Returns :	**m** *: ndarray, see dtype parameter above*

After the call the developer gets a response from the Data Analytics module with the mean energy consumption per room for the next day. Figure 4.3 shows an example of response in JSON format, this response is not final and can change with time.

Response

```
Content-
Type: application/json
Date: <date>
Server: <server>
{mean:
{
result: X
}
}
```

Figure 4.3. JSON return from an API call..

4.2. Second use case

The second use case reports the maximum and minimum consumption of each room, which is useful to know which place is consuming more energy. To calculate this, the "min" and "max" functions from the Data Analytics module have to be called, an example for the min function is depicted through the URL below.

*http://<domain>/statistics/averagesvariances/amin/***listnumber?values=[a,b,c]**

Table 4.2: Min function parameters

Parameters :	**values** *: array_like* Input data.
Returns :	**amin** *: ndarray or scalar*

After the call the developer gets a response from the Data Analytics module with the min consumption of each room. Figure 4.4 shows an example of response in JSON format, this response is not final and can change with time.

Response

```
Content-Type:
application/json
Date: <date>
Server: <server>
{min:
{
result: X
}
}
```

Figure 4.4. JSON return from an API call.

All the requests that can be made at the IMPRESS Platform are shown at Table 4.3, this list is not final and can change with time.

Table 4.3: Data Analytics module algorithms endpoints

Algorithms	URL
Data Mining Cluster	
DBSCAN([eps, min_samples, metric, ...])	/datamining/cluster/**DBSCAN**/<Parameters>
KMeans([n_clusters, init, n_init, ...])	/datamining/cluster/**KMeans**/<Parameters>
MiniBatchKMeans([n_clusters, init, ...])	/datamining/cluster/**MiniBatchKMeans**/<Parameters>
MeanShift([bandwidth, seeds, ...])	/datamining/cluster/**MeanShift**/<Parameters>
Ward([n_clusters, memory, ...])	/datamining/cluster/**Ward**/<Parameters>
NearestNeighbors([n_neighbors, ...])	/datamining/cluster/ **NearestNeighbors** /<Parameters>
Data Mining Cluster Metrics	
adjusted_mutual_info_score(...)	/datamining/cluster/metrics/**adjusted_mutual_info_score**/<Parameters>
adjusted_rand_score(labels_true, ...)	/datamining/cluster/metrics/**adjusted_rand_score**/<Parameters>
completeness_score(labels_true, ...)	/datamining/cluster/metrics/**completeness_score**/<Parameters>
homogeneity_score(labels_true, ...)	/datamining/cluster/metrics/**homogeneity_score**/<Parameters>
mutual_info_score(labels_true, ...)	/datamining/cluster/metrics/**mutual_info_score**/<Parameters>
normalized_mutual_info_score(...)	/datamining/cluster/metrics/**normalized_mutual_info_score**/<Parameters>
consensus_score(a, b[, similarity])	/datamining/cluster/metrics/**consensus_score**/<Parameters>
Data Mining Classification	
RandomForestClassifier([...])	datamining/classification/**RandomForestClassifier**/<Parameters>

RandomTreesEmbedding([...])	datamining/classification/**RandomTreesEmbedding**/<Parameters>
ExtraTreesClassifier([...])	datamining/classification/**ExtraTreesClassifier**/<Parameters>
AdaBoostClassifier(...[, criterion])	datamining/classification/**AdaBoostClassifier** /<Parameters>
GradientBoostingClassifier([loss, ...])	datamining/classification/**GradientBoostingClassifier**/<Parameters>
GaussianHMM([n_components, ...])	datamining/classification/**GaussianHMM**/<Parameters>
GaussianNB	datamining/classification/**GaussianNB**/<Parameters>
NearestCentroid([metric, ...])	datamining/classification/**NearestCentroid**/<Parameters>
SVC([C, kernel, degree, gamma, coef0, ...])	datamining/classification/**SVC**/<Parameters>
LinearSVC([penalty, loss, dual, tol, C, ...])	datamining/classification/**LinearSVC**/<Parameters>
DecisionTreeClassifier([criterion, ...])	datamining/classification/**DecisionTreeClassifier**/<Parameters>
Data Mining Classification Metrics	
accuracy_score(y_true, y_pred[, ...])	datamining/classification/metrics/**accuracy_score**/<Parameters>
average_precision_score(y_true, y_score)	datamining/classification/metrics/average_precision_score/<Parameters>
classification_report(y_true, y_pred)	datamining/classification/metrics/classification_report/<Parameters>
confusion_matrix(y_true, y_pred[, ...])	datamining/classification/metrics/confusion_matrix/<Parameters>
jaccard_similarity_score(y_true, y_pred)	datamining/classification/metrics/jaccard_similarity_score/<Parameters>
precision_score(y_true, y_pred[, ...])	datamining/classification/metrics/precision_score/<Parameters>
recall_score(y_true, y_pred[, ...])	datamining/classification/metrics/recall_score/<Parameters>
roc_auc_score(y_true, y_score)	datamining/classification/metrics/roc_auc_score/<Parameters>
Data Mining Regression	
RandomForestRegressor([...])	/datamining/regression/**RandomForestRegressor**/<Parameters>
AdaBoostRegressor(...[, criterion, ...])	/datamining/regression/ **AdaBoostRegressor** /<Parameters>
GradientBoostingRegressor([loss, ...])	/datamining/regression/**GradientBoostingRegressor**/<Parameters>
IsotonicRegression([y_min, y_max, ...])	/datamining/regression/**IsotonicRegression**/<Parameters>
LinearRegression([...])	/datamining/regression/**LinearRegression**/<Parameters>
LogisticRegression([penalty, ...])	/datamining/regression/**LogisticRegression**/<Parameters>
RandomizedLogisticRegression([...])	/datamining/regression/**RandomizedLogisticRegression**/<Parameters>
KNeighborsRegressor([n_neighbors, ...])	/datamining/regression/**KNeighborsRegressor**/<Parameters>
RadiusNeighborsRegressor([radius, ...])	/datamining/regression/**RadiusNeighborsRegressor**/<Parameters>
cross_decomposition.PLSRegression([...])	/datamining/regression/**cross_decomposition.PLSRegression**/<Parameters>
SVR([kernel, degree, gamma, coef0, tol, ...])	/datamining/regression/**SVR**/<Parameters>
Data Mining Regression Metrics	
explained_variance_score(y_true, y_p	/datamining/regression/metrics/**explained_variance_score**/<Para

red)	meters>
mean_absolute_error(y_true, y_pred)	/datamining/regression/metrics/**mean_absolute_error**/<Parameters>
mean_squared_error(y_true, y_pred)	/datamining/regression/metrics/**mean_squared_error**/<Parameters>
r2_score(y_true, y_pred)	/datamining/regression/metrics/**r2_score**/<Parameters>
Statistics Orders	
amin(a[, axis, out, keepdims])	/statistics/orders/**amin**/<Parameters>
amax(a[, axis, out, keepdims])	/statistics/orders/**amax**/<Parameters>
nanmin(a[, axis, out, keepdims])	/statistics/orders/**nanmin**/<Parameters>
nanmax(a[, axis, out, keepdims])	/statistics/orders/**nanmax**/<Parameters>
ptp(a[, axis, out])	/statistics/orders/**ptp**/<Parameters>
percentile(a, q[, axis, out, overwrite_input])	/statistics/orders/**percentile**/<Parameters>
Statistics Averages and Variances	
median(a[, axis, out, overwrite_input])	/statistics/averagesvariances/**median**/<Parameters>
average(a[, axis, weights, returned])	/statistics/averagesvariances/**average**/<Parameters>
mean(a[, axis, dtype, out, keepdims])	/statistics/averagesvariances/**mean**/<Parameters>
std(a[, axis, dtype, out, ddof, keepdims])	/statistics/averagesvariances/**std**/<Parameters>
var(a[, axis, dtype, out, ddof, keepdims])	/statistics/averagesvariances/**var**/<Parameters>
nanmean(a[, axis, dtype, out, keepdims])	/statistics/averagesvariances/**nanmean**/<Parameters>
nanstd(a[, axis, dtype, out, ddof, keepdims])	/statistics/averagesvariances/**nanstd**/<Parameters>
nanvar(a[, axis, dtype, out, ddof, keepdims])	/statistics/averagesvariances/**nanvar**/<Parameters>
Statistics Correlating	
corrcoef(x[, y, rowvar, bias, ddof])	/statistics/correlation/**corrcoef**/<Parameters>
correlate(a, v[, mode, old_behavior])	/statistics/correlation/**correlate**/<Parameters>
cov(m[, y, rowvar, bias, ddof])	/statistics/correlation/**cov**/<Parameters>
Optimization Stochastic Algorithms	
tabusearch(<parameters>)	/optmization/stochastic/ **tabusearch** /<Parameters>
randomsearch(<parameters>)	/optmization/stochastic/ **randomsearch** /<Parameters>
scattersearch(<parameters>)	/optmization/stochastic/ **scattersearch** /<Parameters>
Optimization Evolutioray Algorithms	
geneticalgorithm(<parameters>)	/optmization/evolutionary/ **geneticalgorithm** /<Parameters>
nsga(<parameters>)	/optmization/evolutionary/ **nsga** /<Parameters>
spea(<parameters>)	/optmization/evolutionary/ **spea** /<Parameters>

Optimization Swarm Algorithms	
pso(<parameters>)	/optmization/swarm//<Parameters>
beesalgorithm(<parameters>)	/optmization/swarm//<Parameters>
antcolonysystem(<parameters>)	/optmization/swarm//<Parameters>
Pre processing	
Binarizer([threshold, copy])	/preprocessing/**Binarizer**/<Parameters>
Imputer([missing_values, ...])	/preprocessing/**Imputer**/<Parameters>
KernelCenterer	/preprocessing/**KernelCenterer**/<Parameters>
LabelBinarizer([neg_label, ...])	/preprocessing/**LabelBinarizer**/<Parameters>
LabelEncoder	/preprocessing/**LabelEncoder**/<Parameters>
MinMaxScaler([feature_range, copy])	/preprocessing/**MinMaxScaler**/<Parameters>
Normalizer([norm, copy])	/preprocessing/**Normalizer**/<Parameters>
StandardScaler([copy, ...])	/preprocessing/**StandardScaler**/<Parameters>
binarize(X[, threshold, copy])	/preprocessing/**binarize**/<Parameters>
label_binarize(y, classes[, ...])	/preprocessing/**label_binarize**/<Parameters>
normalize(X[, norm, axis, copy])	/preprocessing/**normalize**/<Parameters>
scale(X[, axis, with_mean, ...])	/preprocessing/**scale**/<Parameters>

5. References

[1] IMPReSS 8.2 deliverable "Application Architecture for Energy Managemnt", 14 May 2014.

[2] Ian H. Witten, Eibe Frank, Mark A. Hall. Data Mining: Practical Machine Learning Tools and Techniques. Third Edition. 2011.

[3] Oracle® Database Concepts 10g Release 1 (10.1). Available in: http://docs.oracle.com/pdf/B10698_01.pdf. Date Acess: 07/11/2014.

[4] Oracle® Database Concepts 11g Release 1 (11.1). Available in: http://docs.oracle.com/cd/B28359_01/server.111/b28318.pdf. Date Acess: 07/11/2014.

[5] Business Dictionary. Available in: http://www.businessdictionary.com/definition/optimization.html. Date Acess: 07/11/2014.

[6] Russell, Stuart J.; Norvig, Peter (2003), Artificial Intelligence: A Modern Approach (2nd ed.), Upper Saddle River, New Jersey: Prentice Hall, ISBN 0-13-790395-2. Date Acess: 06/23/2014.

[7] GLOVER, F. e KOCHENBERGER, G. A. Handbook of Metaheuristics. Kluwer Academic Publishers, Boston. 2003.

[8] Brownlee J. Clever Algorithms: Nature-Inspired Programming Recipes. 2012.

[9] Dodge, Y. (2006) The Oxford Dictionary of Statistical Terms, OUP. ISBN 0-19-920613-9

[10] Hornick M, Oracle Corporation, Java Specification Request 73, Java Data Mining (JDM), 2004.

[11] http://scikit-learn.org/stable/index.html

[12] http://pymc-devs.github.io/pymc/

[13] E. Tapia, S. Intille, and K. Larson, Activity recognition in the home using simple and ubiquitous sensors, no. 1933. 2004, pp. 1–6.