CREATION OF A WEB APPLICATION FOR SMART PARK SYSTEM AND PARKING PREDICTION STUDY FOR SYSTEM INTEGRATION

AUTHOR: SERGI SOLER

SUPERVISORS: OSCAR BELMONTE,

ROBERTO HENRIQUES and ALBERT REMKE

PARTNERSHIP:

Hall Effect

init
institute of new imaging technologies

ABSTRACT

In the last few years, we have observed an exponential increasing of the information systems, and parking information is one more example of them. The needs of obtaining reliable and updated information of parking slots availability are very important in the goal of traffic reduction. Also parking slot prediction is a new topic that has already started to be applied. San Francisco in America and Santander in Spain are examples of such projects carried out to obtain this kind of information.

The aim of this thesis is the study and evaluation of methodologies for parking slot prediction and the integration in a web application, where all kind of users will be able to know the current parking status and also future status according to parking model predictions. The source of the data is ancillary in this work but it needs to be understood anyway to understand the parking behaviour. Actually, there are many modelling techniques used for this purpose such as time series analysis, decision trees, neural networks and clustering. In this work, the author explains the best techniques at this work, analyzes the result and points out the advantages and disadvantages of each one. The model will learn the periodic and seasonal patterns of the parking status behaviour, and with this knowledge it can predict future status values given a date.

The data used comes from the Smart Park Ontinyent and it is about parking occupancy status together with timestamps and it is stored in a database. After data acquisition, data analysis and pre-processing was needed for model implementations.

The first test done was with the boosting ensemble classifier, employed over a set of decision trees, created with C5.0 algorithm from a set of training samples, to assign a prediction value to each object. In addition to the predictions, this work has got measurements error that indicates the reliability of the outcome predictions being correct. The second test was done using the function fitting seasonal exponential smoothing tbats model. Finally as the last test, it has been tried a model that is actually a combination of the previous two models, just to see the result of this combination. The results were quite good for all of them, having error averages of 6.2, 6.6 and 5.4 in vacancies predictions for the three models respectively. This means from a parking of 47 places a 10% average error in parking slot predictions. This result could be even better with longer data available.

In order to make this kind of information visible and reachable from everyone having a device with internet connection, a web application was made for this purpose. Beside the data displaying, this application also offers different functions to improve the task of searching for parking. The new functions, apart from parking prediction, were:

- Park distances from user location. It provides all the distances to user current location to the different parks in the city.

- Geocoding. The service for matching a literal description or an address to a concrete location.

- Geolocation. The service for positioning the user.

- Parking list panel. This is not a service neither a function, is just a better visualization and better handling of the information.

FIGURE INDEX

PLOT INDEX

TABLE INDEX

INDEX

SECTION 1: INTRODUCTION

SECTION 2: SYSTEM REVIEW AND PARKING DATA USED

SECTION 3: DATA MODELLING

SECTION 4: IMPROVEMENTS OF THE SMART PARK ONTINYENT WEB SERVICE

SECTION 5: CONCLUSIONS AND FUTURE WORK

REFERENCES

SECTION 1:
INTRODUCTION

1.1-Introduction

In the last few years the concept of smart cities has appeared. A smart city basically uses digital technologies to enhance performance and wellbeing, to reduce costs and resource consumption, and to engage more effectively and actively with its citizens (Wikipedia). But it can be summed up as making citizen's life easier and optimizing city resources. Key 'smart' sectors include transport, energy, health care, water and waste.

Avoiding to waste time for drivers looking for a place where to park and reducing traffic congestion in urban areas are two general examples. If you are near to a parking lot, this information can be very useful. But if you are distant and you plan to arrive to a city in some hours or the next day, then you are not able to know whether a parking lot will have free slots when you are there. A driver might head to one parking lot, find it full and try another one consuming fuel and wasting time. If the driver would know in advance which parking lots have free slots, he would save time and the overall traffic would be reduced as well (Reinstadler et al. 2013).

Parking places monitoring is already done sometimes by many public and private facilities: universities, shopping centres and others. This is done by different parking slot detection technologies that are not the target of this work. This parking places information is displayed through variable-message sign situated at major roads or at the park entrances, but it might be disseminated through the Internet network.

For this work it has been used data from the Ontinyent Smart Park, it is a parking monitoring platform that enables to know the occupancy status of some public parks distributed along this city. It makes possible to store this data and create a dynamic database. This data will be used to create parking behaviour models.

If the parking areas follow some concrete temporal patterns or periodic behaviour, then modelling techniques and or machine learning methods can find out these patterns and provide forecasts for the future time. Having such an infrastructure to know this information and to store the occupancy records in time will allow using the data to build and test those parking behaviour models. After test these models, they will be used in order to predict occupancy status in future and guide the drivers to one of the parks based in these predicted data.

All this work should be supported by front end platform and an interface, for the displaying of all this information. Also it would enable the user interaction with the different services. For this reason a web application is made for parking information display and other functionally like routing, navigation and more.

1.2- Goals and Work structure

The main goal of this thesis is to reduce time in the task of look for a vehicle parking. To achieve this, reliable predictive models based on historical parking observations were built and tested. So, in addition to know parking information in real time, drivers can plan to arrive somewhere in future and check the availability for that time and park area according to these models. If they know in advance whether there is parking places or not, it will help to save time, and then reduces overall traffic as well. The level of time-saving has not been determined in this work, it was left out of the scope of this work.

Also it has been created a web application able to insert date times of arrival and return a prediction for that time.

The work done in this thesis has been:

1-Create and test parking behaviour models for parking occupancy predictions.

2-Create a web application in order to display the parking vacancies information and some other functions; and create the prediction service that choosing a date-time the service will show the predictions for that time in the future.

Beside the section I dedicate to the introduction, the content of this thesis is structured in the following sections.

- **Section 2** presents the new web application of the Smart Park Ontinyent and all the new functionalities. It lists the technologies and tools used for this mission. It is explained how are the new functionalities implemented and it shows the result of them.

- **Section 3** has the purpose of review the Smart Park Ontinyent system, infrastructure, data collection and data used in general. It focuses on data selection and it makes the first analysis and reflections about the data gathered.

- **Section 4** explains the most used data modelling techniques and it carries out a data pre-processing and a second analysis, then it is described the methodologies to implement three different modelling techniques which are tested afterwards.

- Finally, in **section 5** conclusions drawn are listed and future lines of research are defined.

SECTION 2:
SYSTEM REVIEW AND PARKING DATA USED

2.1-Spot vehicle detection description

The current SPO includes a total of eighteen parking clusters, in which have been installed two kinds of magnetic sensors, they are responsible to carry out the vehicle detection and provide to the system these data. They are going to be explained in this section. The first type of sensor is a fluxgate magnetometer and the other is an inductive loop detector.

1.-Magnetometer (fluxgate magnetometer): Fluxgate magnetometers works by detecting perturbation (magnetic anomaly) in the earth's horizontal and vertical magnetic field. Fluxgate magnetometers provide the advantage of being insensitive to weather condition such as snow, rain and fog. It is also more accurate and less susceptible than loops to stresses of traffic. As the technologies for wireless transmissions evolves, wireless RF link are also used to transmit data in some models. Among the disadvantages of using fluxgate magnetometers are the small detection zones in some model which requires multiple units for full lane detection as well as the close proximity required for accurate detection (Cheung et al., 2005; Mimbela and Klein, 2007).

2.-Inductive loop detectors: Inductive Loop Detectors (ILDs) are wire loops of various sizes which are excited with signals whose frequencies range from 10 to 50 kHz. The oscillation frequency of the inductive loop is directly controlled by the inductance of the loop which changes with vehicle presence. The sensor system proved to be a mature and well understood technology with large experience base and extensive research conducted. Besides that, its flexibility also allows for the implementation in a large variety of applications. The vehicle detection zone can be easily enlarged by combining the loops together. Compared with other commonly used techniques, ILD provides the best accuracy for count data. In fact, inductive loop sensors became the common standard for obtaining accurate occupancy measurements. (Idris et al, 2009)

From the SPO project point of view, a parking cluster or area provided by an inductive loop detector is called loop parking, and a parking with many magnetometers is called smart parking. As it is explained previously, they have different ways to detect parked vehicles, but also they have different ways of counting the vehicle-spots of the parking.

A loop park sums or subtracts vehicles to the parking status depending if a vehicle goes in or comes out of the park. So, each event modifies the park status, by -1, that means one car comes out the park, by 1, one car goes in and by 2, two vehicles in a very small differential of time go in. This kind of park works fine in the cases of parking clusters which only have a way to get in and get out, or only one way to get in and one to get out the park. Figure 2.1 shows two examples of this case of parks.

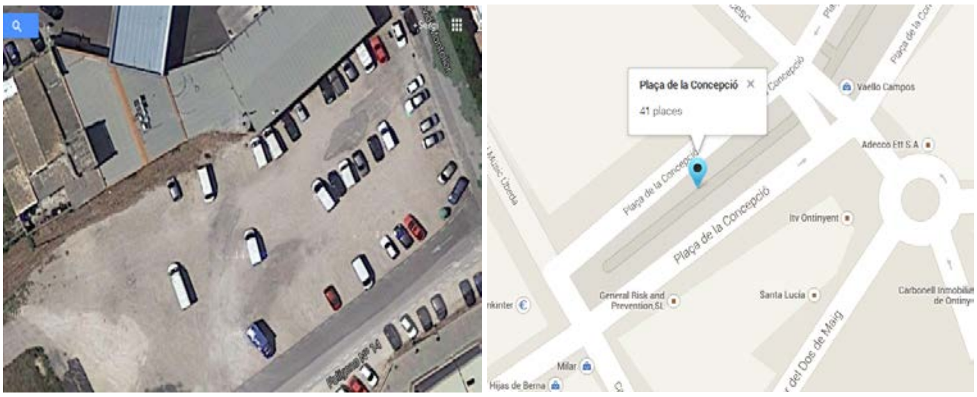


Figure 2.1 Loop parks in the SPO

The smart park is composed of many magnetometers, each of the spots of the parking cluster or area has a sensor under their pavement surface, then, each sensor is responsible of detecting a single vehicle parked above it. This kind of parking is usually a very well demarcated parking cluster or side streets where parking in line or in angle, where each parking place keeps invariable in time as in the figure 2.2. The smart parking CPU manages all the sensors within the parking area and processes data of each one in order to get parking lots amount information.

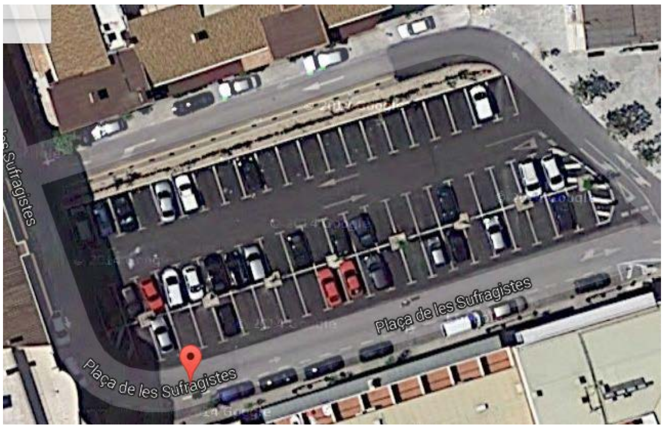


Figure 2.2 Example of Plaça de les Sufragistes smart park

2.2-Description of the current web service

Ontinyent Smart Park or Smart Park Ontinyent (next mentions as SPO) is a system for parking real time information. This system monitors in real time the available-occupied slots of public parking clusters of the city of Ontinyent. This system has two main parts, the parking information collector, made by magnetic sensors with different functions installed in each parking and one CPU for each parking that manages the sensors data and translate it to occupancy information. The second part is the display of the information and it is done by two different ways. The first one is a light panel at the parking entrance to notice the vehicle drivers about the occupancy information. And the second one is the website of the SPO where any user having a device connected to internet can consult the current status of the parks.

The current service of the SPO is in the http://ontinyent.halleffect.net/ address. Figure 2.3 shows how the website looks like. All the content area of the website is occupied by a map where the several parks are located. They are represented by markers and when clicking on them, it opens an info window popup where the parking name and status in real time is shown. The user can interact with the map zooming in and out, panning and changing the background from map to satellite view. It uses the Google maps JavaScript API under its conditions to render the map, set the controls and add the parking markers

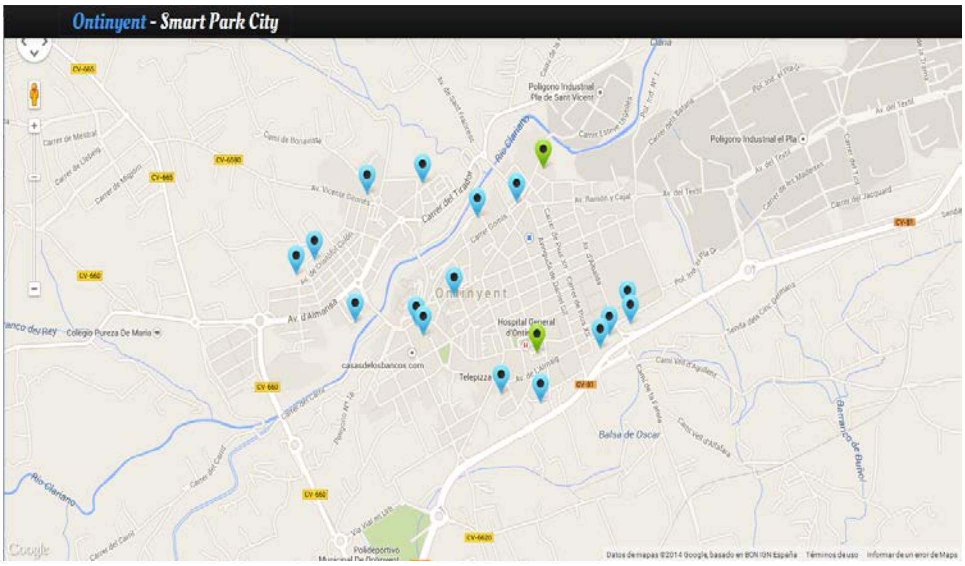
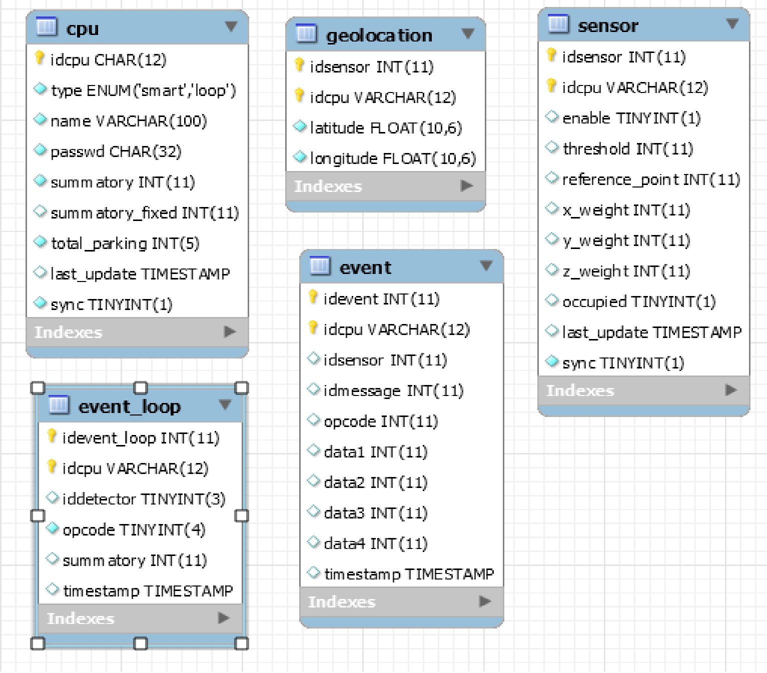


Figure 2.3 appearance of the current website of the SPO

2.3-Parking lot occupancy historical data

The SPO parking occupancy data has been stored in a MySQL database; it has occupancy data of 4 parking of a total of 18 that integrate the SPO. From these four parking clusters two are loop and three are smart ones, one of this parking has both kind of sensors, loop and smart. So, there is occupancy data for 4 parking.



cpu
idcpu CHAR(12)
type ENUM('smart','loop')
name VARCHAR(100)
passwd CHAR(32)
summatory INT(11)
summatory_fixed INT(11)
total_parking INT(5)
last_update TIMESTAMP
sync TINYINT(1)
Indexes

geolocation
idsensor INT(11)
idcpu VARCHAR(12)
latitude FLOAT(10,6)
longitude FLOAT(10,6)
Indexes

sensor
idsensor INT(11)
idcpu VARCHAR(12)
enable TINYINT(1)
threshold INT(11)
reference_point INT(11)
x_weight INT(11)
y_weight INT(11)
z_weight INT(11)
occupied TINYINT(1)
last_update TIMESTAMP
sync TINYINT(1)
Indexes

event
idevent INT(11)
idcpu VARCHAR(12)
idsensor INT(11)
idmessage INT(11)
opcode INT(11)
data1 INT(11)
data2 INT(11)
data3 INT(11)
data4 INT(11)
timestamp TIMESTAMP
Indexes

event_loop
idevent_loop INT(11)
idcpu VARCHAR(12)
iddetector TINYINT(3)
opcode TINYINT(4)
summatory INT(11)
timestamp TIMESTAMP
Indexes

Figure 2.4 SPO database schema diagram

The SPO database consists of 6 tables (ignoring the instruction table that there is no need to know anything from it), and the relations between them. Starting from the general and going to the particular the first table to start with is CPU. Each of the eighteen parking is managed by a CPU (*central processing unit).* These CPU's are managing one or more sensors that are responsible of detecting cars passing over them or parked above them. Depending on the type of sensor, they generate events when the magnetic field has changed, this is translated in car parked or car gone, and it is stored in the event_loop table if sensor type is loop or in event if type equal to smart. Sensors have been buried in a location which is stored in the geolocation table. Figure 2.4 shows the different tables and their keys, indexes and fields.

SECTION 3:
DATA MODELLING

3.1.-Introduction

Sherrod (2014) defines data modelling as one of the most useful applications of statistical analysis in the development of a model to represent and explain the relationship between data items (variables). Many types of models have been developed, including linear and nonlinear regression (function fitting), discriminating analysis, logistic regression, support vector machines, neural networks and decision trees. Each method has its advantages: there is no single modelling method that is the best for all applications.

Supervised and Unsupervised Machine Learning

Methods for analyzing and modelling data can be divided into two groups: "supervised learning" and "unsupervised learning."

A supervised machine learning system, acquires knowledge to the system, using inductive or deductive techniques on a set of training data. Involves analyzing a number of examples of the concept is being taught and from that analysis is built, or induce, a definition which is to describe the concept in question.

It is known such a system which employs human knowledge to solve problems that usually requires human intelligence, which means, a system that is able to solve issues efficiently, in a similar way as an expert person would do in the considered topic. The process "learns" how to model (predict) the value of the target variable based on the predictor variables, this is called training session of the process, and it is the main difference from an unsupervised system.

The expert systems are characterized by two components (Skidmore, 1989)

-A knowledge base that contents the data and the related rules to the data and the hypotheses that solves. In the case of parking occupancy, the hypotheses are the occupancy status to forecast.

-An algorithm which deduces the relations between the data and the hypotheses or classes to solve.

Decision trees, regression analysis and neural networks are examples of supervised learning. If the goal of an analysis is to predict the value of some variable, then supervised learning is the recommended approach.

Unsupervised learning does not identify a target (dependent) variable, but rather treats all of the variables equally. In this case, the goal is not to predict the value of a variable but rather to look for patterns, groupings or other ways to characterize the data that may lead to understanding of the way the data interrelates. Cluster analysis, correlation, factor analysis (principle components analysis) and statistical measures are examples of unsupervised learning.

3.2-Data modelling techniques

3.2.1-Neuronal Networks

The neuronal networks are a method of machine learning that the initial goal was to emulate the biological processes of the information (Corchado and Fyfe, 2004) In a neuronal network, each of the descriptive variable of an object (input layer) is multiplied to some weights (hidden layer) which are varying along of the training session until reaching the wanted solution (output layer). Once got the corresponding weights to the different variables, the neuronal network can be applied to the several cases to be used in obtaining a desired output. The main advantages of the neuronal networks versus other methods of classification are that they do not have to follow a normal distribution of the data, its ability of adaptation to complex and non linear patterns. (Jensen, 2005).

The structure of neuronal network applied for a parking occupancy modelling is usually composed by three layers (figure 3.1):

-an input layer, that can be made by historical parking occupancy data in many timestamps.

-one or many hidden layer that allows simulating the non linear patterns in the input layer

-an output layer that contains the parking occupancy status in the time or the variations of the parking in time (differentials)

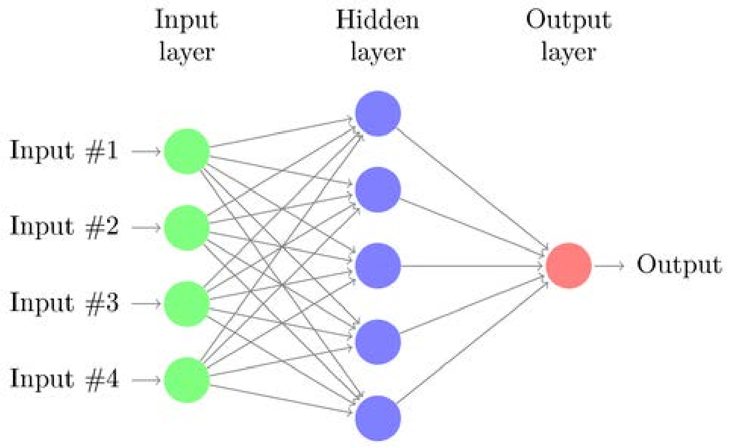


Figure 3.1 An example of a neural network

This methodology has been rejected for parking occupancy modelling because it has some disadvantages compared to methods based on knowledge. First, the knowledge network is in the form of weights, which prevents the intelligibility of class assignments made. These weights are hidden and cannot be modified by the operator (Tullis and Jensen, 2003). That is, the descriptive variables new knowledge for the user is not removed, but that knowledge extraction is internal to the network and not allocated to the user except to the class assignment done. Second, knowledge that can be an expert on a phenomenon is difficult to introduce into the network.

3.2.2-Decision Trees

The decision tees are a series of conditions organized in a hierarchic way with shape of a tree (Hernández Orallo et al., 2004). They are very useful for finding structures in a high dimension spaces and in problems that mix categorical and numeric data. The structure of decision trees (figure 3.2), where the rules are evaluated in order to test hypothesis, is the best suitable to express an expert system. (Jensen, 2005).

A decision tree is a way to represent the knowledge obtained along the inductive learning process. It can be interpreted as the result structure of the division of the representing space from a big set of examples or prototypes.

Pursuant to Hernández Orallo et al. Depending on the variable they predict, they can be classify in classification trees and regression trees. The classification trees are used to predict categorical variable, while the regression ones are used to predict continuous variables. A regression tree is similar to a classification tree, except that the Y variable takes ordered values and a regression model is fitted to each node to give the predicted values of Y.

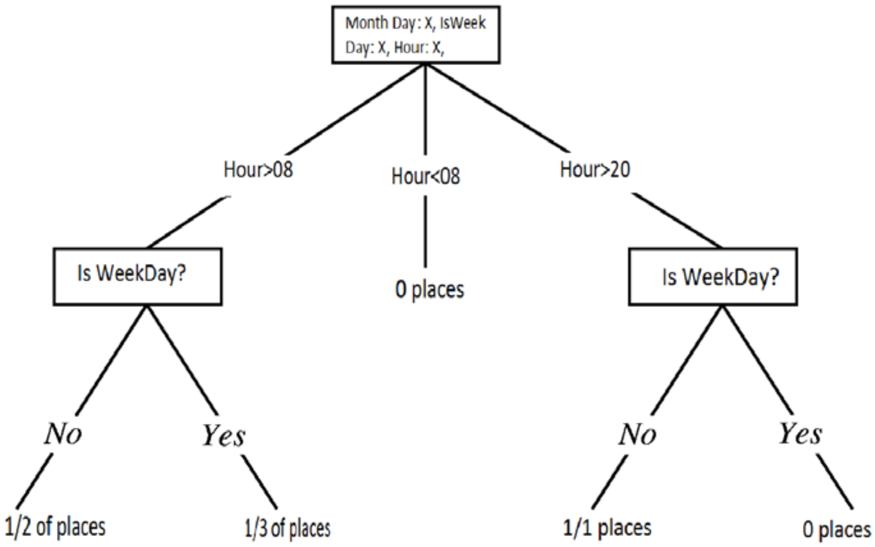


Figure 3.2 An example of decision tree and rules for parking modelling

Advantages and disadvantages

The main **advantages** of the decision trees are the following: (Hernández Orallo et al., 2004)

They can be applied to several tasks: classification, regression, clustering, etc.

They deal with continuous and discrete attributes.

They are flexible. They do not do any supposition about the data distribution, rather than some statistics methods. This characteristic allows to add discrete or logic data to the parking data like holiday (true or false) regardless the distribution and correlation between them.

They tolerate the noise, to non significant attributes and null data.

The extracted conditions are intelligible for the user.

There are software and libraries available and free in some cases.

They allow dealing with non lineal relation between features and classes.

Good with large data sets. Oates and Jensen (2008) have shown that for large data sets it can be the case that tree size will increase with the number of training examples while the accuracy of the tree is not affected by adding training e examples.

Among the **disadvantages** of the decision trees it should be mentioned they are not as precise as other methods like neuronal networks, they can model more arbitrary functions (nonlinear interactions, etc.) and therefore might be more accurate, provided there is enough training data. Moreover they are "weak", this is, they are rather dependent of the samples that they use for the training section. Two different samples belonging to the same distribution can generate very different trees. Other disadvantages are:

The predictions are discrete values.

The accuracy does not come with probabilities but the exactness do.

Cannot work on (linear) combinations of features.

Not well classification-prediction with more than around 60 different classes to classify-predict.

Ensemble classifier methods

The ensemble classifiers methods appeared with the intention of improve the accuracy of the predictions made by several methods of classification. These methods are based on the definition of several hypotheses that they combine, currently by voting to obtain the most trustable hypothesis. The combination of decision trees will be more reliable when more accurate and various to the other trees.

The boosting method (Freund, 1995) is an ensemble classifier method that once applied, it allows increasing the classification's accuracy. Boosting is used to minimize the sensibility of a classification or regression algorithm, a decision tree in this case, to the possible existing errors in the data features on the data input as well as the possible classification errors in the training data. This method has been described by Friedman et al. (2000) as one of the most important in the classification methodology in recent developing.

The systematic followed by boosting (Freund and Schapire, 1997; Quinlan, 1996a) to build ensemble classifier is to assign a weight to each sample of all training. The higher the weight of a sample, the greater its influence on the classifier. After an iteration, that is, in the construction of each model, the weight vector is adjusted to reflect the performance of the model so that the samples wrongly classified their weights are increased, while reducing the weights of the examples correctly classified. Thus it ensures that the model learned in the next iteration gives more importance to samples previously misclassified (Hernández Orallo et al., 2004). The main idea is that each new model attempts to correct the failures of previous models.

The boosting algorithm most widely used classification is the AdaBoost (Freund and Schapire, 1997). The algorithm starts learning a model from a set of training samples in which all have the same weight and whose sum is equal to unity. Then the model error is estimated as the sum of the weights of misclassified samples. If the error is greater than 0.5,that is, more than half of the sample were misclassified, the algorithm stops to be an error too high and that model is discarded. It also stops the algorithm if the error is 0 to be an accurate model.

3.2.3-Time-series function fitting

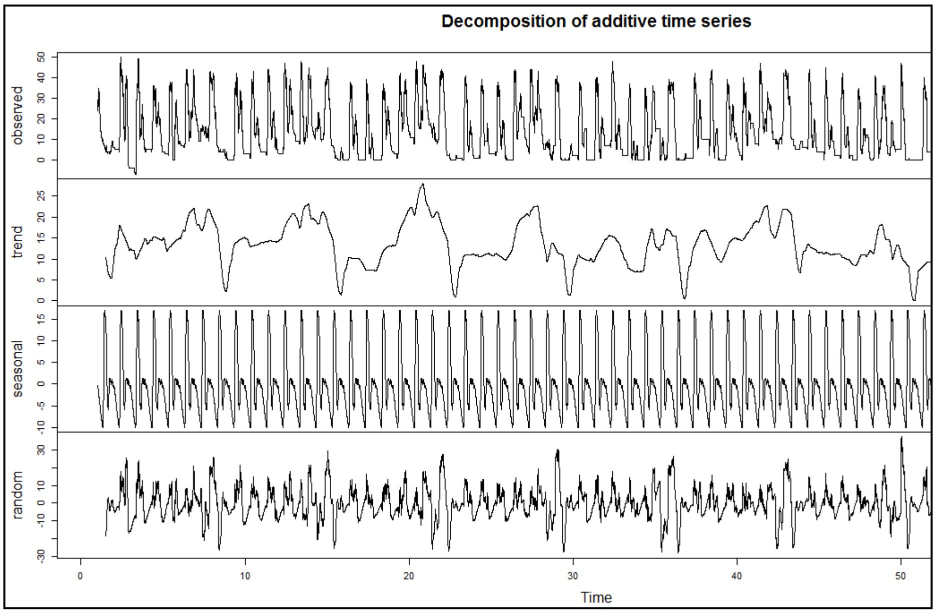
Basic concepts

A time series is a collection of observations of well defined data items made sequentially along the time (McCleary e Hay, 1980). For example, measuring the number of free slots of a parking lot each 5 minutes or measuring every minute the number of cars passing through a specific observation point results in a time series. A time series can be decomposed into three components:

1.-Seasonality, which can be identified by regularly spaced peaks and troughs having a consistent direction and approximately the same magnitude in every cycle. A seasonal pattern exists when a series is influenced by seasonal factors (e.g., the quarter of the year, the month, or day of the week). Seasonality is always of a fixed and known period. Hence, seasonal time series are sometimes called periodic time series.

2. Trend, which is defined as long term movement in a time series. It is the result of influences such as the long term trend of the parking cost or traffic growth.

3. Irregular components, which reflect the remaining parts after the seasonal and trend components of a time series have been estimated and removed. They result from short term fluctuations in the series which are neither systematic nor predictable.



Plot 3.1 Trigonometric decomposition parking occupancy in June

Many time series present high frequency multiple seasonal patterns, as an example, we have the parking occupancy data interpolated per 5 minutes interval of CEAM parking in Ontinyent, depicted in plot 3.1, presents a daily seasonal component and a weekly seasonal pattern. A longer version of this series might also exhibit monthly and annual seasonal patterns, but the limited sample size means that it cannot be explicitly modelled.

Most existing time series models are designed to accommodate simple seasonal patterns with a small integer-valued period (such as 12 for monthly data or 4 for quarterly data).

Seasonal exponential smoothing methods, which are among the most widely used forecasting procedures in practice, have been shown to be optimal for a class of innovations state space models. They are therefore best studied in terms of this framework because it then admits the possibility of likelihood calculation, the derivation of consistent prediction intervals and model selection based on information criteria (De Livera et al. 2011).

The traditional approaches of non-linear versions of the state space models exponential smoothing although widely used, suffer from some important weaknesses. Akram et al. (2009) showed that most of non-linear version are unstable, having infinite values of forecast variance beyond a certain forecast point, also other issues like error distribution and not availability of results of prediction distribution.

To handle a wider variety of seasonal patterns De Livera et al. (2011) proposed a tbats exponential smoothing modified model which has been carried out and tested with our parking occupancy data.

TBATS stands for Trigonometric (from Trend, Season and remainder decomposition, (see plot 3.1) Box-Cox trans-form, ARMA errors, Trend, and Seasonal Components. The BATS model is the most obvious generalization of the traditional seasonal innovations models to allow for multiple seasonal periods. However, it cannot accommodate non-integer seasonality, and it can have a very large number of states; the initial seasonal component alone contains m_T non-zero states. This becomes a huge number of values for seasonal patterns with high periods.

Model selection

In linear state space models the typical approach is to estimate unknown parameters like the smoothing parameters and the damping parameter using the sum of squared errors or the Gaussian likelihood. In this tbats context (De Livera et al. 2011) it is necessary also to estimate and perform the following parameters and coefficients:

I-The unknown Box-Cox transformation parameter w, and the ARMA coefficients.

II-Selecting the number of harmonics k in the trigonometric models

III-Selecting the ARMA orders p and q for the models

Advantages

Differentiability, integration and discretization due to the function fitting modelling.

Upper and lower confident intervals for forecasted values.

Disadvantages

There is not option to set upper and lower forecast values.

Previous forecasted values calculated when an only desired value wanted in a future scenario.

Only univariate data acceptance.

Fixed data time frequency (5min, 1 hour, 1 day, etc), it can cause lose and alteration of information from the original training dataset.

3.2.4-Goodness of fit measurements

The evaluation of the model performance in this work has been deployed by using an ensemble of training data and an ensemble for an evaluation. The training data are used to learn the tree in case of decision trees model or to apply time series analysis, whereas its performance is obtained as the deviance errors between the evaluation data and the predicted one.

Forecast errors are extremely useful to determine if the forecasting model is accurately predicting demand, the goal and/or the application of the model will determine whether the fitting it is valid or not. There are several performance measures of forecast error such as Mean Absolute Percentage Error (MAPE), Absolute Deviation (MAD), Mean Square Error (MSE), Root Mean Square Error (RMSE) (Rojas, 2006).

Mean Absolute Deviation (MAD)

The mean absolute deviation (MAD) is the average of the absolute deviation over all periods. MAD measures the average distance of the sample errors from the mean of the error values. If the value of MAD is large, it is reasonably to say that the errors in the data set are spread out (variable). In contrast to MSE, the MAD is very good at detecting 69 overall performance of the model. It does not concentrate largely on the error of individual observations. The MAD is given by

$$MAD = \frac{1}{n}\sum_{t=1}^{n}|e_t|$$

MAD is appropriate to use when the numerical difference between the forecast value and the actual value is important. This would correspond to the sum of the area under the curves of the two graphs, real data and forecasted one. So it is like an integration of the error along the tested interval.

Mean Absolute Scaled Error

The MASE was proposed by Hyndman and Koehler (2006) as a generally applicable measurement of forecast accuracy without the problems seen in the other measurements. They proposed scaling the errors based on the in-sample MAE from the naïve forecast method. Using the naïve method, we generate one-period-ahead forecasts from each data point in the sample. Accordingly, a scaled error is defined as

$$\mathrm{q_t} = \frac{\mathrm{e_t}}{\frac{1}{\mathrm{n}-1}\sum_{\mathrm{i}=2}^{\mathrm{n}}|\mathrm{Y_i} - \mathrm{Y_{i-1}}|}$$

The result is independent of the scale of the data. A scaled error is less than one if it arises from a better forecast than the average one-step, naïve forecast computed in-sample. Conversely, it is greater than one if the forecast is worse than the average one-step, naïve forecast computed in-sample. The mean absolute scaled error is simply:

$$MASE = mean(|q_t|)$$

Visualization

Other goodness of fit method it is based in visual skills derived from user experience Visualization is the graphical presentation of information, with the goal of providing the viewer with a qualitative understanding of the information contents.

By presenting data in a visually appealing manner, the user is presented with a quick qualitative understanding of the information. Representing large amounts of disparate information in a visual form often allows you to see patterns that would otherwise be buried in vast, unconnected data sets. Sometimes it is easy to see better the performance of a model fitting having a look to a well plot graph rather than extracting error measurements.

3.3-Aplication and results

3.3.1-Occupancy data extraction and selection

The first work to be done with this historical data is to analyze and extract some information from it. The approach in this work has been to **determine the number of parking with data and the start date and last update of the records**. Then, it can be possible to figure out which is the most suitable park to work with and which ones are not because of the lack of records or time.

Executing the following query in figure 3.3 will count the total number of days with data for the three smart parking grouped by their CPUs id.

```
SELECT `event`.`idevent`,  `event`.`idcpu`,  `event`.`idsensor`,

 count(distinct (date(`event`.`timestamp`)))

FROM `spark`.`event` group by idcpu;
```

Figure 3.3 Days with data grouped by smart park query

As a result of the query three rows are selected (table 3.1), one per smart parking and looking at the last column it shows the number o days with occupancy data in this database. 5, 4 and 5 are the total number of days for these smart parks, this makes this kind of parking not good for time series modelling and analysis, because this work finds a good approach to model occupancy data and makes prediction based on those models, and these models cannot be implemented with so few days as it has, it would need at least some weeks of data in order to catch weekly seasonally patterns to construct a more consistent model. So, this fact leaves no choice when it comes to dismiss this type of parking.

idevent	idcpu	Cpu name	idsensor	days
1	9059af4e22c9	Alex_Lab	6	**5**
88959	9059af503375	Sufragistes	142	**4**
143022	9059af5c1e9a	Concepcio	31	**5**

Table 3.1 Number of days with data per each smart park

Carrying out the same query for the loop parks gave the result of 126 and 144 day with occupancy data respectively for the Sufragistes and CEAM loop parks (table 3.2). That makes possible to use these two parking for data modelling and data analysis and finally apply predictions based on this models and test the model fitting.

It has been a handicap in this work the fact of the youth of the Smart Park Ontinyent project; only allowing collecting some months of occupancy data so that it has been applied modelling for monthly seasons rather to three-four monthly season or yearly season.

idevent	idcpu	Cpu name	days
234	b827ebeecef0	Sufragistes	**126**
19440	bc6a2996df13	CEAM	**144**

Table 3.2: Number of days per each loop park

The next step has been to know the time interval of those days for each of the loop park, this information has been extracted from the event_loop table as well by sorting the records by the timestamp, first ascendant and descendent. The Sufragistes loop park has records from the 21th of June of 2014 to 24th of October of the same year.

Moreover the CEAM loop park from May the 30th to November the 4th. The next query in figure 3.4 executed for each of the months will give as a result the number of changes or events in the parks for that month.

```
SELECT

`event_loop`.`summatory`, `event_loop`.`timestamp`

FROM `spark`.`event_loop`

WHERE `event_loop`.`idcpu` LIKE 'bc6a2996df13' and
date(`event_loop`.`timestamp`) between '2014-09-01' and '2014-09-30';
```

Figure 3.4 Events x months in a loop park query

	Sufragistes loop events	CEAM loop events
	Nº of places = 90	**Nº of places = 47**
June 2014	12.131	8.215
July 2014	>50.000	7.649
August 2014	25.892	4.998
September 2014	32.698	2.789
October 2014	28.789	1.568

Table 3.3 Summary of events per month and for the two loop parks

The Sufragistes Park has a capacity of 90 places and it is located 100m from the only hospital of Ontinyent and Vall d'Albaida region, the CEAM Park is located in the surroundings of the city and has a capacity of 47 places. Table 3.3 has been built executing the query at figure 3.4 for each month and parking.

As the author of this work has been informed, the SPO it is a young project. The Hall Effect Company (SPO developers) makes its own sensors and they are still in stage of developing and testing the sensors. This is the reason why it can be found strong variations between the amount of events among the months as well as the absence of data for some periods within this time interval.

After this advice, and after analyze the **Sufragistes** park event data, the author have decided to make the following assumptions in order to choose the date intervals to work with.

-It can be seen that there is relatively few data for June because only 9 days have records. The month of July suffered of inaccuracy; perhaps due to the effect of the high temperatures reached in that month, and it has produced around the double of events that has been defined as normal for one month, looking at the information this would be around 27.500 events per month +5.000 of standard deviation.

-Other assumption taken has been to reject the month of August (normal holiday month in Spain) because of the different behaviour for model training as well for testing, however this will be solved as soon there will be available data for the whole year making possible yearly season behaviour modelling.

-September and October have similar amount of events and are normal months in terms of school and work; October has data until the 24th day of the month.

After these reasoned assumptions, there are two capable months to apply parking occupancy modelling for monthly seasons which are **September** and **October** of 2014.

Now, the same decision must be taken for the **CEAM** park, this is to choose date periods to work with.

-From the point that it has not been considered August as normal month, it has been rejected for the same reason as previously.

-Analyzing the days with data of October, there is not data from days 3 to 10 and 12 to 20, so it has been rejected this month too.

-It is needed for some modelling methods like regression analysis the use of consecutive months for model goodness of fit test.

-**June** and **July** accomplish all requirements of normal months in terms of work season and present similar number of events.

Eventually, with the parking occupancy data collected until now, and made the previous data analysis and assumptions, it leaves the possibility to analyze, model the parking occupancy behaviour and test them. There are two loop parks, Sufragistes and CEAM, with 90 and 47 places respectively, and also two months per park with occupancy data to model and test, September and October for the first and June and July for the second. Next section nº 3.3.2.2 will explain the different modelling methods and techniques and section 3.3.2.3 will describe the methodology carried out for the task of modelling, and the steps done in this work.

3.3.2-Occupancy data analysis and modelling

3.3.2.1-Introduction

For data analysis and modelling the author of this work has chosen the R environment for these tasks. R is a free software programming language and software environment for statistical computing and graphics. The R language is widely used among statisticians and data miners for developing statistical software and data analysis. Among the reasons for choosing R, the most prominent are:

-Many forecasting existent packages which include algorithms like C.50, tbats, neural networks, etc...

-Documentation for every function, methods and classes for each package.

-Functional an object oriented language.

-Implemented classes for date-time handling and data conversion.

-R server that enables to establish communication between java and R for processing in the server side of the application.

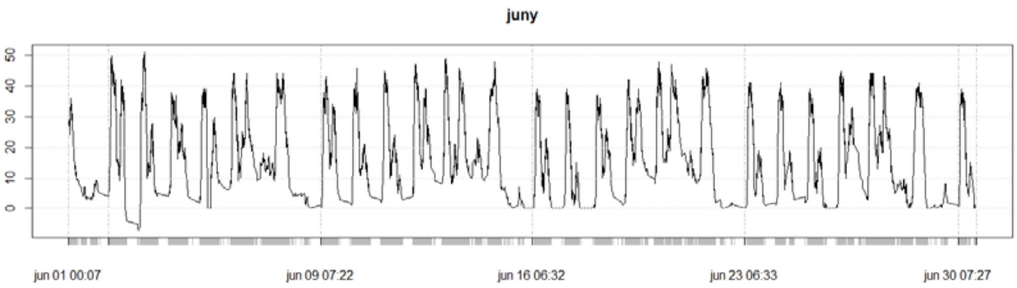
3.3.2.2-Data analysis

To start analyzing the monthly datasets of the parks we should to convert the data from database to a readable file to be processed by R or establish a database connection in R. For this work has been chosen to save the occupancy data in files. In case it is deliverable to work with new data available in the database at the same time we would use a database connection instead, but just for model experimenting it is easier to work with saved files.

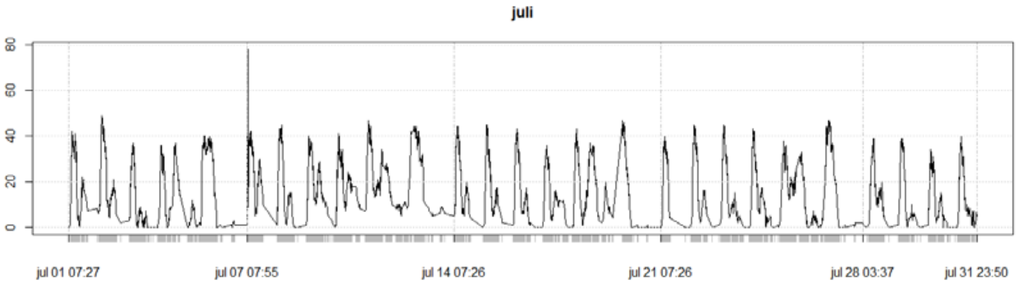
After R reads the source parking data files, it must be created an object of time series, and for this we need to use a class constructor called xts (extensible time series) contained in a package with the same name. Then we can plot these xts objects in a graph to see the behaviour of the parking occupancy versus the time. Figure 3.5 shows the r scripts to do that. The plots 3.2, .3, .4 and 3.5 present the data in a graphical representation.

```
>juny=read.csv("C:\path\\juny.csv")
>library(xts)
>juny = xts(juny, order.by=strptime(juny$timestamp, format="%Y-%m-
%d %H:%M:%S"))
>plot(juny
```

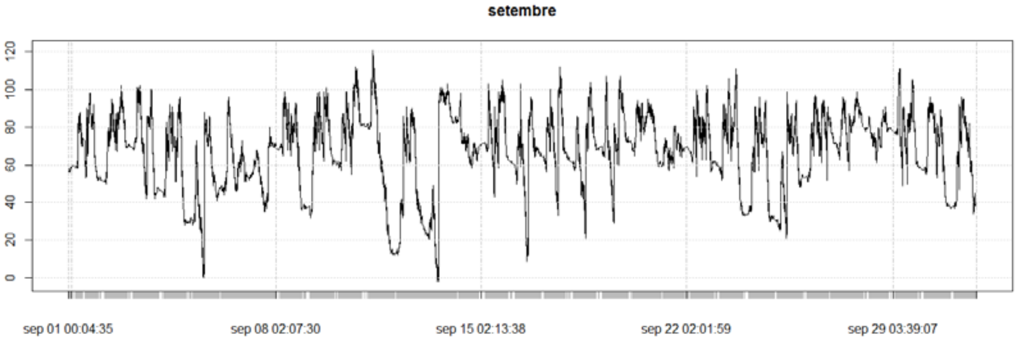
Figure 3.5 R scripts for reading, converting and plotting the June data of CEAM Park



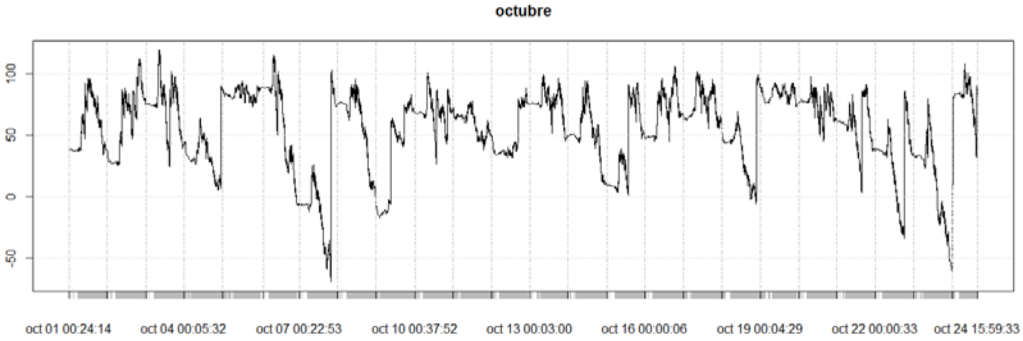
Plot 3.2 June data for CEAM Park plotted



Plot 3.3 July data for CEAM Park plotted



Plot 3.4 September data for Sufragistes Park plotted



Plot 3.5 October data for Sufragistes Park plotted

Remembering the maximum occupancy level of parks were 47 for CEAM Park and 90 for Sufragistes. The minimum amount of parked cars logically cannot be less than 0. So, taking a quick look at the plots of the parks, it can be seen than in June for CEAM and both September and October for Sufragistes have got amounts below zero, then we should consider all these values as not available data due to the inability of this fact, probably it could be used the relative information between measurements for these negative intervals, this is, the occupancy variation (Δoccupancy) of the data, but since it is needed the absolute value, it is not possible to know how far the error went and sum it to the variation. In the other hand there are values over the parking occupation limits, see the plots number 3.2 and 3.3, has values over 47 and 3.4 and 3.5 over 90. This fact can be two source of error, the first, like in the negatives values is due to sensor inaccuracy or just not well working, the other source may occur when a car or some cars go in the park being this totally full of parked cars, this will cause an over measurement of parked cars. This fact has explanation only in the case of loop park (which is the type we are using the data from), but not for the smart one.

This study is going to carry out two different modelling techniques. In one hand the C.5 algorithm which uses decision trees and in the other hand the multiple seasonality exponential smoothing function fitting for time series, tbats. One extra technique uses a combination of both models. Figure 4.4 represents the whole process from data pre-processing up to the model testing stage. It is going to be applied this process for two different park scenarios already known, CEAM and Sufragistes, 47 and 90 vehicle places.

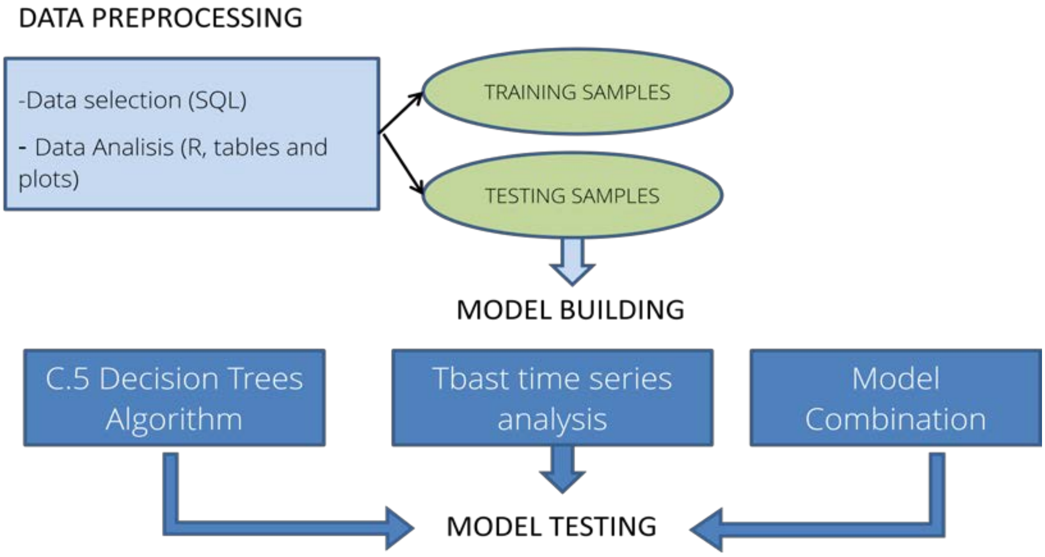


Figure 4.4 Parking data modelling process

For CEAM park it is going to be used one whole month as a training data, a second consecutive month will be used as a testing dataset in order to avoid the data correlation, and to see better whether the model suffers of overfitting, it means the model is too much fitted to the training samples, too specific and not general. If we had use the same dataset for training and testing the results would have been much better, because models present usually better results for the training dataset itself.

For Sufragistes park, it has been decided to test the model with the same training data in C.5 model, because it is observed in plots 3.4 and 3.5, and in data trigonometric decomposition (see section 3.2.3 tbats) the data nature seems to be not correct with lots of errors with negative values for occupancy going up to -50 units, and records that are exceeding a lot the maximum capacity of the park. Then just for a test it will be carried out the test thereby. As the tbats model can only make forecast for further data, not for the same time as for training one, then, it will be tested with the next month, which is October.

It was tested another supervise machine learning modelling techniques like support vector machines and artificial neural networks. But they did not give satisfactory results as C.5 and tbats models. So, it has been refused to present those tests in this work.

3.3.2.3-Modelling tests

3.3.2.3.1-C.5 Decision trees algorithm

The first approach in this section has been the decision trees for data modelling and data forecasting. As seen in the section number 3.2.2, decision trees is a supervised machine learning technique which extracts knowledge from a set of sample with the related labels, classes or values.

It has been used the C.5 algorithm with 100 boosting multiclassifiers trees, this is included in the C50 package for R.

First of all, C5 does not handle timestamps as complete field like other time series analysis; rather they need a vector of descriptors per sample. In order to get this, it has to be extracted time and date information from the event_loop timestamp field. From a timestamp of structure `2014-07-01 08:23:22` (YYYY-MM-DD HH:MM:SS), here is a list of information fields extracted from it:

Day of the week, which is going to define **from 1 to 7** what kind of day it is.

Weekday. Logical type true or false, indicating whether it is weekend or weekday. Additionally it could be included holiday days, for instance December the 25th, as a weekend day.

Hour of the day. Numerical value which indicates the hour of the day, hour of the day will include the minute's information as a decimal fraction added to the value. For example 15:30 will be 15,5 hours.

This vector of three elements is going to be the individual descriptor. Additionally it could be used the day of month and day of the year for more than one month and one more than year training data sets respectively. In the same way, when fine smart park (individual parking sensors) data available a set of new descriptors could be added. Kunjithapatham et al. have included in the same task of predicting parking lot availability in San Francisco (2010) the following descriptors:

Total occupied time

Total vacant time

Fraction time occupied

Fraction time vacant

Application

If the reader of this thesis reminds from section number 3.2.2, decision trees tolerate null data, so, being accurate, the negative values of parking occupancy should be converted to null and the data higher than the parking occupancy either could be coerced to null or to the maximum capacity of the park. Applying these concepts in R, we have:

```
>juny$summatory[juny$summatory<0] <- NA
>juny$summatory[juny$summatory>47] <- 47
```

Figure 3.7 R code chunk to clean the parking occupancy data

Now it is time to extract the date-time information, as explained previously it comes from the timestamp field. Then it will be constructed a data frame with all the descriptors of the training data. In R code that is:

```
>junyhour=hour(juny$timestamp) # extracting the hour of the time
>junyminute=minute(juny$timestamp) # extracting the minutes
>hour=junyhour + minute(juny$timestamp)/60 # adding the minutes to
hour
>weekDay= as.factor(isWeekday(as.timeDate(juny$timestamp))) # is
week day? True or false
>dayOfWeek=wday(juny$timestamp) # day of the week
>junyD=data.frame(juny$summatory,dayOfWeek,weekDay,hour) # binding
all into a dataframe
>make.names(names(junyD),unique = FALSE, allow_ = TRUE)
```

Figure 3.8 R code chunk for extracting date-time information

At this point the data it is ready for use it, using the C50 function it is going to be built the model that characterize our occupancy historical data. Figure 3.9.

```
trainX <- junyD [,-1]
trainY <- junyD [,1]
model <- C50::C5.0( trainX, trainY,trials=100)
```

Figure 3.9 R code chunk for building the parking occupancy model

Now, the model has been built and we need to predict the next month, July, by using this model, in order to see how fine it works, to see the goodness of feet by comparing real data with predicted data. To do this we have to predict in the same date-time points where there is data recorded. So, first, we need to extract the date-time information of July data as was done with June. Then, just need to predict using the predict function in C50 package (figure 3.10), which takes a new data frame with the predictors, and a model.

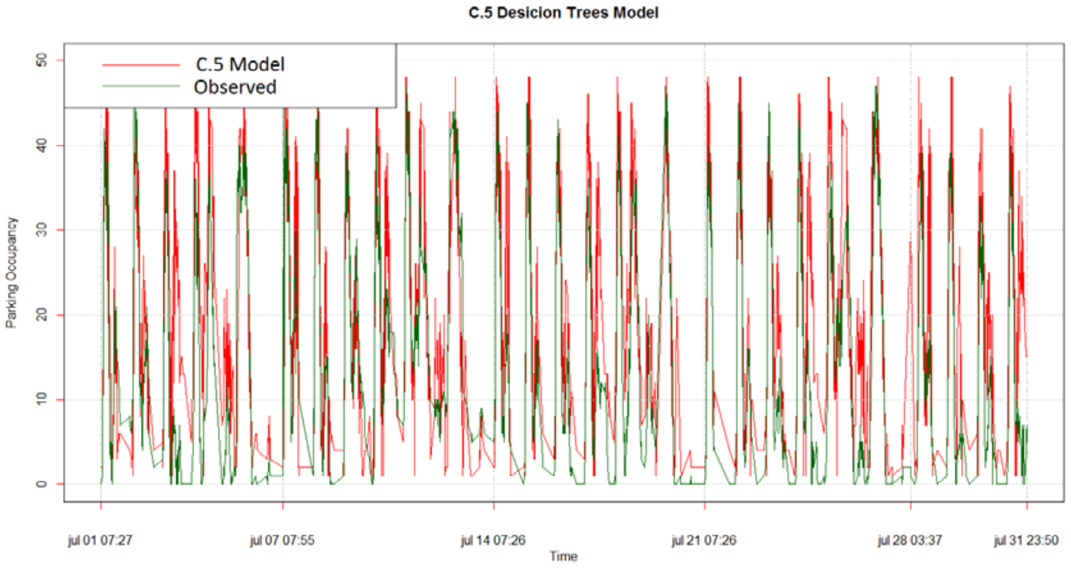
```
>p <- predict( model, testjuly, type="class" )
```

Figure 3.10 R code chunk for July predictions

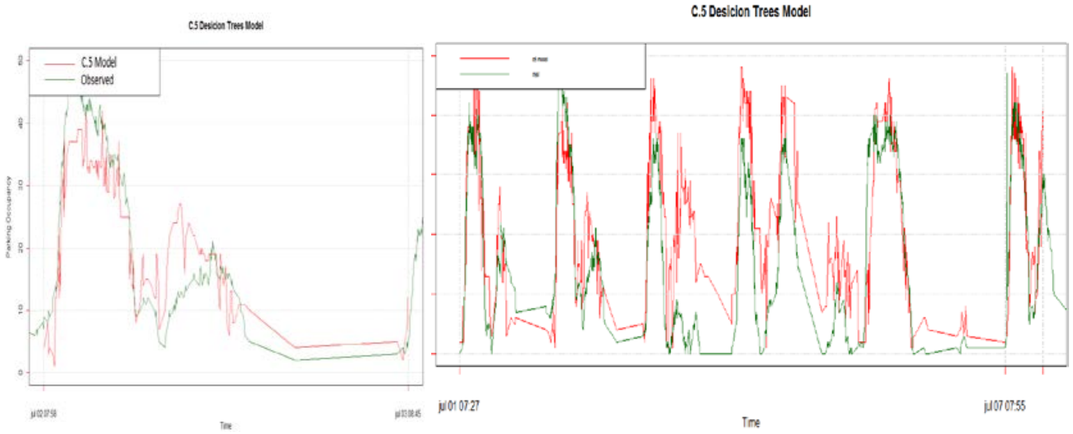
Finally, it will be plotted the predictions together with the real data. Besides it will be calculated the mean absolute deviation (MAD) which is giving an overall objective value of the model performance. Obtaining of MAD:

```
>summatory<-as.numeric(julyD$summatory)
>diffC5<-sum(abs(as.numeric(p)-summatory),na.rm=TRUE)
>MAD=diffC5/NROW(na.omit(as.data.frame(july)))
6.238463
```

Figure 3.11 R code chunk for getting the MAD error



Plot 3.6 CEAM July predictions



Plot 3.7 CEAM July predictions zoomed in for a day and for a week

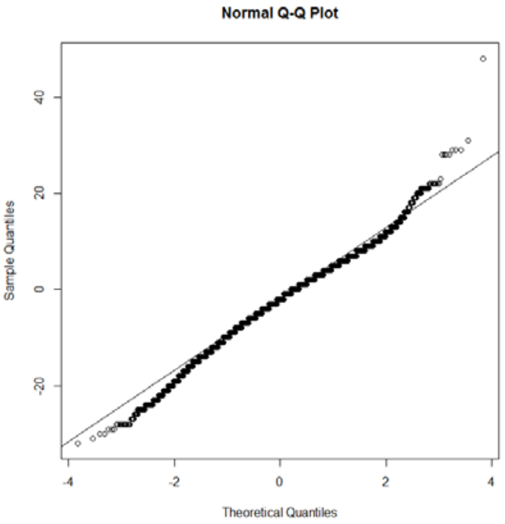
The model forecasting for the month of July has come out with a MAD error, also called MAE (mean absolute error) of 6.24 units, this means the arithmetic mean of the sums of the absolute deviations between real and forecasted data. If it is wanted to know the relative value of this error, it is just to divide that number between the park capacity. This will give an error rate of 13.27 % of parking capacity.

Plots 3.6 and 3.7 are showing a comparison between the real data and the C.5 model predictions in order to see the performance of the model beyond the error measurements. It can be observed that the model predictions fit very well and follow the real data very well too. Observing the plot nº 3.7, a day zoomed, we see one of the phenomena that captures more attention, which is that the model capture the sharp picks of the real data very well, in addition to capture the trend. In the other hand, there are points where the model cannot capture the real data; that would be caused due to the randomness factor of the behaviour. If longer data available in a future, it will be able to extract monthly and yearly information that maybe could improve the model performance. Another deduction extracted from the plots, is that the residuals of the model forecasted values are both above as well as below the real data, so it can be said that the model errors follow a Gaussian distribution centred on zero, then it can be extracted the standard deviation, variance, etc… of the error residuals.

```
> sd(as.numeric(p)-summatory)
[1] 7.690626
> mean(as.numeric(p)-summatory)
[1] 2.553275
```

Figure 3.12 Rcode chunk to determine standard deviation and mean of the error residuals

In the figure 3.12 is shown that the residuals mean is relatively close to 0 (2,5) and the standard deviation of residuals is 7,7~8 cars. And just for demonstrate the normality of the residuals, the plot 3.8 shows de Q test using the qqnorm function in R.



Plot 3.8 Q test for C.5 model error residuals

Repeating the same procedure but for the Sufragistes park, a 90 places park and with different behaviour, it is going to be apply again a C.5 historical occupancy data modelling. As a result we have got the following results:

MAD= 14.52524

For this park, the model has been tested for the same month of training, because of the reasons explained at the beginning of this section, the few trustily of the collected data. The relative result to its parking places is MAD/90=16,14 %s, which is a little beat worst than the model test on CEAM park. It must be remembered that CEAM park model was tested with different data than trained with.

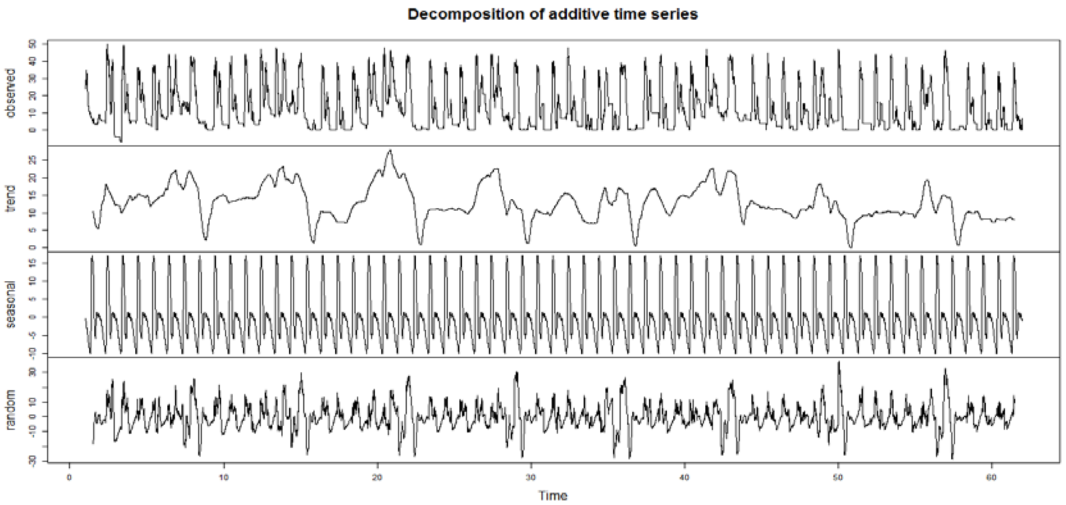
As explained in section nº 3.2.2 decision trees are prone to errors in classification problems with many class and relatively small number of training examples. The training number of examples would not be a problem in this work, but the number of classes to predict is, since there are parks with more than 90 places, that means more than 90 values to predict.

One possible solution might be the usage of regression trees instead a classification one. Deshpande, 2011 suggests using those when existent linear relationships between predictors and response, and use C4.5 and beyond when non linear relationships exist. Instances of regression trees algorithms are CART, GUIDE or M5'.

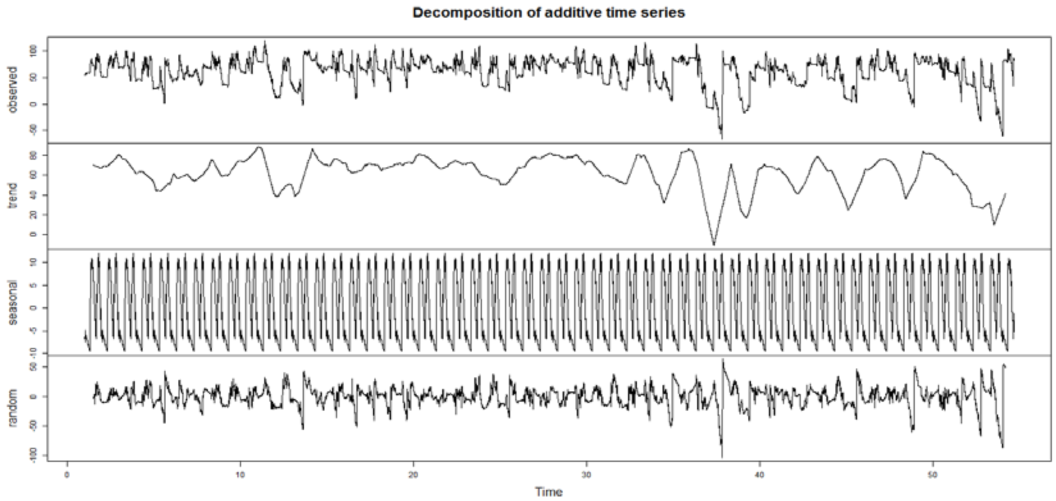
Another approach discussed by Mitchell, 1997 is to convert a continuous attribute or responses in discrete-valued ones. It can be accomplished by dynamically defining new discrete set of intervals. For the case of 90 parking places of a park would be to split the occupancy in n partitions, where n= 90/tolerance. The only question is how to select the best value for the tolerance. If we decide to use 5 parking places as a tolerance for this park, we have n=90/5=18 partitions. So n=1 will contain all the occupancy values from 0 to 4, n=2 from 5 to 9 and successively. With this approach the number of classes is reduced from 90 to 18, enhancing the performance of the model, but sacrificing variability in the training dataset classes.

3.3.2.2.1-Tbats time series modelling

Before start to work with time series data, it is needed to have another look to the training data. We need to understand a little bit the data we are working with in order to estimate the results we will get from the models. As is explained in De Livera et al. 2011 a decomposition of the data in seasonal, trend and random components will let us know the results that we can expect a priori, because this components are used by the model itself to characterize the data. So, looking at the decompositions in plot 3.9 and 3.10 it can be extracted some information for these two data sets. It is going to be extracted the proportions of the three components of the data and extract some conclusions from that information.



Plot 3.9 Trigonometric decomposition for June and July in CEAM

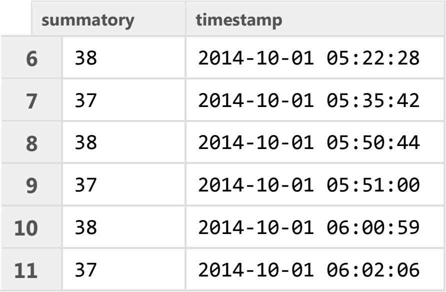


Plot 3.10 Trigonometric decomposition for September and October in Sufragistes

For CEAM (plot 3.9) the trigonometric decomposition shows a strong presence of seasonal and trend components. Trend component varies goes from 5 to 25 units and seasonal component rarely from 0 up to 15 units. Summing both it give a result of 5-40 units, what means that could be possible to control this amount depending on the time-date. The trend component appears to be also periodic for a week, so this means that can be used by tbats as a second season. The random component usually goes from 5 to 15 units but it is always higher in weekends, where the behaviour is not followed as in week days. To arrange this should be implemented a third season that consider the weekends and perhaps holidays as a different behaviour.

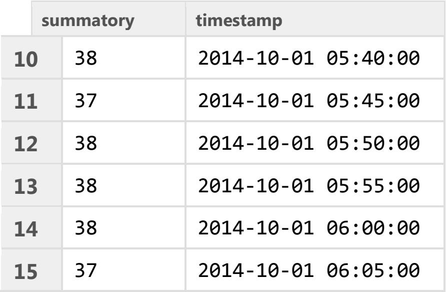
For Sufragistes (plot 3.10) case, the decomposition result differs from CEAM park, being the trend component very strong and not periodic, which will be difficult to capture by a model. Probably this fact has a lot to do with the errors in sensor measurements, because this data for September and October has some negative occupancy status and other records exceed a lot the maximum capacity. Therefore these errors would explain this result. This fact makes not clear how the model should be.

To start to work with time series functions fitting like in this case, exponential smoothing for multiple seasons (tbats), there are some restrictions where using this functions implemented in package “forecast” in R, this restriction must be taken into account before to start using them. Hyndman. 2014, says in his manual that tbats, ets an arima models for example, only allow to work with univariate data, that means, only one attribute apart from time can be modelled using this models. So far, this condition is accomplished because it is only wanted to model the occupancy. The second requirement is it to have equal intervals in data time difference; this means that difference in time of the records must be constant. Then we need to choose a time interval that will be considered representative enough but, not so much dense in frequency. For this work we have set the time interval in 5 minutes, because accomplish the previous hypothesis. In table 3.4 we see the consecutive records of events in a park, and it shows that sometimes this time interval it is longer than five minutes and sometimes shorter, so sometimes it will be applied interpolation and other times there will be extrapolation. The next table 3.5 shows the same data set after being processed.



	summatory	timestamp
6	38	2014-10-01 05:22:28
7	37	2014-10-01 05:35:42
8	38	2014-10-01 05:50:44
9	37	2014-10-01 05:51:00
10	38	2014-10-01 06:00:59
11	37	2014-10-01 06:02:06

Table 3.4 Consecutive records of events in Sufragistes Park



	summatory	timestamp
10	38	2014-10-01 05:40:00
11	37	2014-10-01 05:45:00
12	38	2014-10-01 05:50:00
13	38	2014-10-01 05:55:00
14	38	2014-10-01 06:00:00
15	37	2014-10-01 06:05:00

Table 3.5: The same data set after creating regular time intervals

To do this, it has been created a sequence of records with a fixed 5 minutes time interval, this is 12 records density in an hour and 288 in a day. This sequence it is empty now, but it will be fulfilled later with the historical data. Next scripts (figure 3.13) show how it is done this in R.

```
>juny = xts(juny, order.by=strptime(juny$timestamp, format="%Y-%m-
%d %H:%M:%S"))#creates a extensible time series object
```

```
>regjuny=seq(as.POSIXct("2014-06-01"),length.out=8640, by= "5
min")#creates a sequence of 5 minutes for the whole month

>regjuny= xts(rep(NA,8640), order.by=strptime(regjuny, format="%Y-
%m-%d %H:%M:%S"))# creates a xts with a empty attribute

>j5= merge(regjuny, na.locf(merge(regjuny, juny)[,2]),
join="inner")[,2]# merge both xts in a inner join
```

Figure 3.13 R code chunk for create regular time intervals in June date

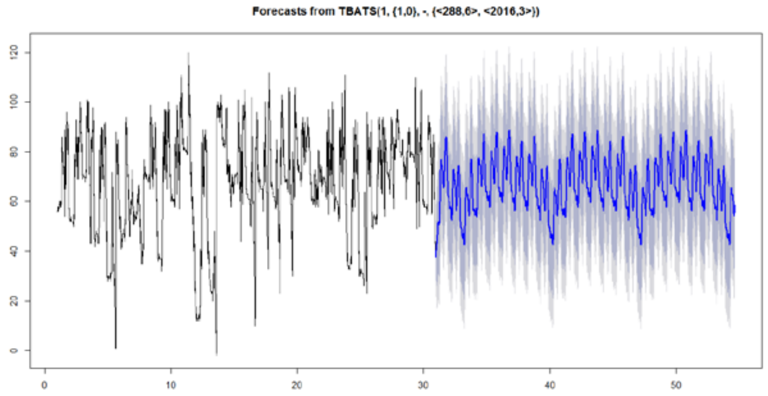
Now, it is possible to apply time series function fitting for the new June 5 minutes data set. Only one step more remains, these functions needs to know the seasonal periods in the data, and then we can carry out the modelling and the testing as well.

Application

Having seasonal periods of 288 (daily periodicity), there are 288 five minutes intervals in 24 hours and 288*7 for weakly periodicity. Now, the model is build following the R code shown in figure 3.14.

```
>y <- msts(j5, seasonal.periods=c(288,288*7))#defining the
seasonal periods
>fit <- tbats(y) # creates the tbats model for the data
>fc <- forecast(fit,h=288*31)# forecast for the month of July
```

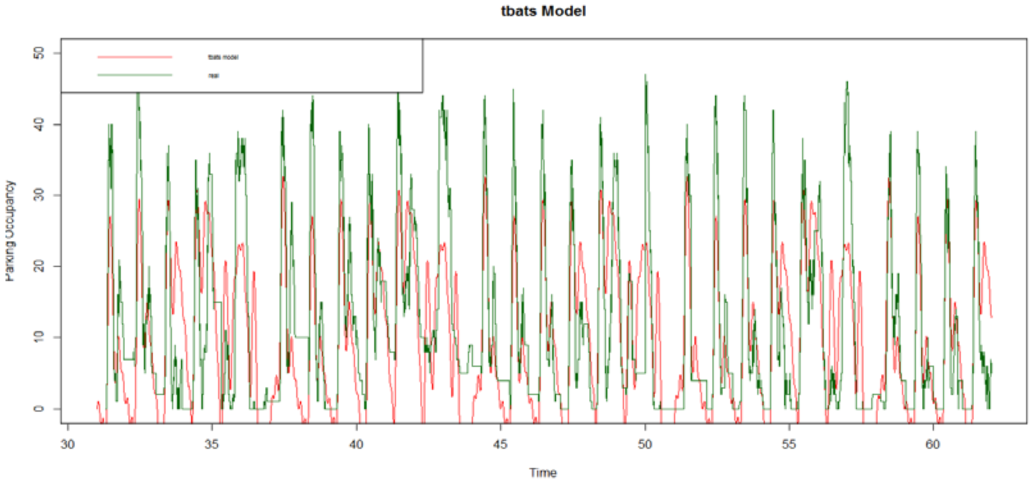
Figure 3.14 R code chunk to set the data frequency of the date-time object, apply tbats modelling and predict July



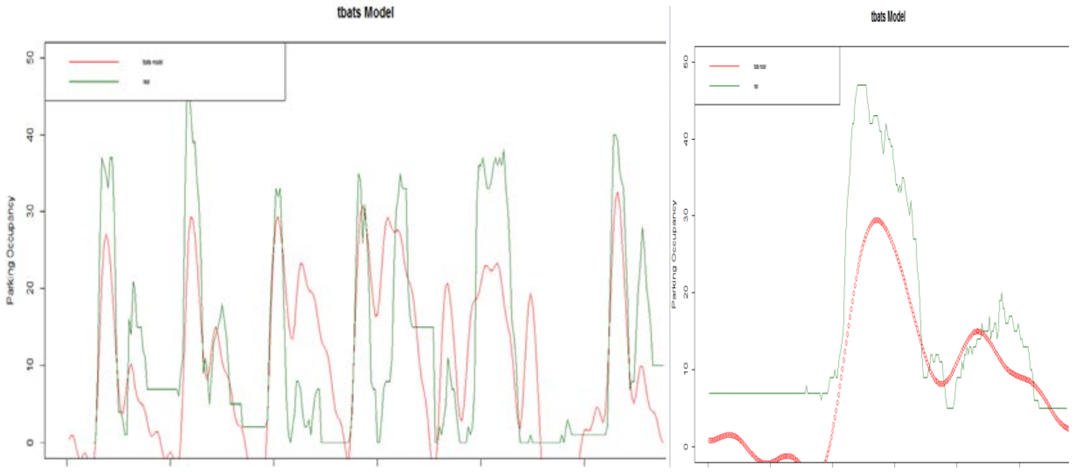
Plot 3.11 tbats forecasts

The plot 3.11 shows the result of plotting the forecast object, it is seen that apart from the forecasted values it contains the upper and lower confident values of the forecasts. The forecasted values come with the same dimensions and time interval of the input data, that is occupancy level and each forecast number 1 is for the 2014-06-01

00:00:00 date-time and the next go 5 minutes by 5 minutes. Let’s going to see the results of the forecasts compared with the real data.



Plot 3.12 forecasted data from tbats vs. real data month of July



Plot 3.13 forecasted data from tbats vs. real data month of July, zoomed in a week and a single day

```
>accuracy(fc, julytest)
MAD=6.659843
```

Figure 3.15 R code to get the model accuracy

Making the same arrange of time frequency interval for the month of July (test data set) in order to get the test and forecast values matched in time, it has been applied the accuracy method for the forecast class, which does exactly the same that was done for getting the MAD error in the decision tree model. As shown in figure 3.15, the MAD error is 6,66 units which is a 14,17% rate error relative to the park capacity. Remembering from the trigonometric decomposition of the occupancy behaviour for this park and for June the random factor was oscillating around 10-20 unities depending on the day and time, so the MAD error it is lower than this amount, that would mean that the model fitting has been better than the expected, but a little bit worst than decision trees model result.

Taking a look at the plots 3.12 and 3.13, it is observed the tbast model capture the trend of the data very well, but rarely reaches the relative picks of the data, it seems the tbats model it is a smoothes this behaviour. Also it can be distinguished the two seasons defined, daily and weekly.

Just repeating the same process but with the Sufragistes park data (September for learning and October for testing) had the next result:

```
>accuracy(fc, octotest)
MAD=20.71416 units
```

This is relative error of 90/20,7= 20,3% which is higher than the previous result, probably due to the lower reliability of this data.

3.3.2.2.3-Combination of Models

The last test carried out has been to combine the two modelling methods, the tbats time series modelling and C.5 decision trees modelling. The way it has been done was making use of the inverse variance-weighted concept from meta-analysis statistics branch.

Inverse-variance weighting is a method of aggregating two or more random variables to minimize the variance of the weighted average. Each random variable is weighted in inverse proportion to its variance. Inverse-variance weighting is typically used in statistical meta-analysis to combine the results from independent measurements.

$$\overline{X} = \frac{\sum_{i=1}^{n} x_i w_i}{\sum_{i=1}^{n} w_i} \quad \text{, where} \quad \mathcal{W}_i = \frac{1}{{\sigma_i}^2}$$

In our case, we have measurements (predictions) coming from different nature. Making use of its MAD error, it is possible to use both measurements by a weighted average. From the previous tests, this work has got the following MAD's errors for the CEAM park and for the month of July:

TBATS MAD error: 6.659843 units.

$\mathcal{W}_{\text{TBATS}}$=1/6.659843^2

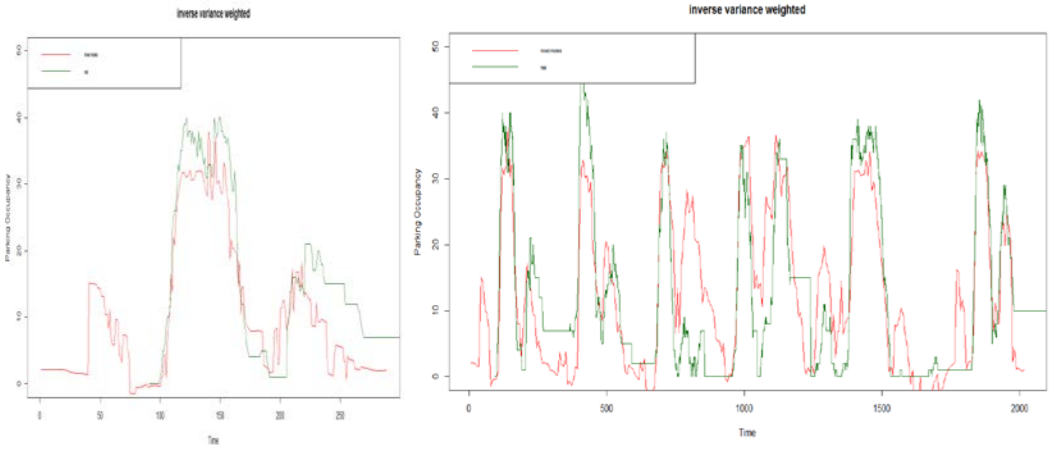
C.5 decision trees MAD error: 6.238463 units.

$\mathcal{W}_{\text{C.5}}$=1/6.238463^2

Due to less MAD error in C.5 model, the C.5 predictions will have a few much weight in the final value.

If this is applied, the weighted average of the measurements, for each future time parking occupancy prediction we will have a combination of two models that hopefully will improve the accuracy of the final predictions.

To test the final accuracy of this combination, it has been applied the same error measure of the past models, the MAD error and the plots of real and predicted data. The MAD error for five minutes time interval parking occupancy predictions for CEAM Park and the month of July has been 5.424384 units, which improves significantly the independent predictions of the C.5 and tbats algorithms (they were more than 6 units). So, it can be said that the model combination in this case was successfully effective. The plot 3.14 shows the result of the model combination.



Plot 3.14 Day and week predictions using a combination of models compared with real data

Finally table 3.6 shows a comparison of the errors for each of the tree modelling techniques used in this work.

	Absolute error (slots units)	Relative error(%) from 47
C.5 Algorithm	6,24	13,27%
Tbats	6,66	14,17%
Combination	5,42	11%

Table 3.6 Error comparison between modelling techniques

SECTION 4:
IMPROVEMENTS OF THE SMART PARK ONTINYENT WEB SERVICE

4.1 The new Smart Park Ontinyent web service developed

Application requirements

First to start describing the new functions developed for the application, it is going to be defined the requirements of this application in order to accomplish with the system needs. It is needed at least a service with the following inputs, date-time value and a park destination. The response will be a number of free places available. Then, also it is needed a parking information displaying. Looking at section 2.2 it is seen that the current web application uses a Google map and the marker it is contains the parking information. So, a priori these are the application requirements that must be taken into account for user interaction, a prediction service and a display of the information.

It was decided that the best way for building this application is to improve the current web application rather than make native system software. This is because a web application it is multiplatform and device independent. Also because it was thought users can use this application every time they want without the need of installing and uninstalling new software on their devices.

4.1.1-Tools, technologies and material

Tools and software installed and used for building the new web service

The following software and tools were used for making and testing the web service:

-Jetty server installed in the local host of the machine in port 9999. To review the web design, well working of the web and enable to see the effects of the requests to the server from the web client.

-Eclipse (IDE) for running the Jetty server, implementing and carrying out the restful services of the web services through the client requests.

-Adobe Dreamweaver and GitHub platform for editing and writing the html, css and JavaScript code.

-MySql Workbench for database management and carrying out the sql queries to extract information and subset the data from the smart park database.

-R studio environment for formatting the parking data, creating the parking occupancy models, graphs and statistics and the R server instance for response the java calls of the forecast service.

Technologies:

-Java, html5, css, JavaScript and R programming language.

- Jquery and jquery mobile JavaScript libraries.

- Ajax.

-Google maps API v3 for JavaScript.

-MapQuest directions web service.

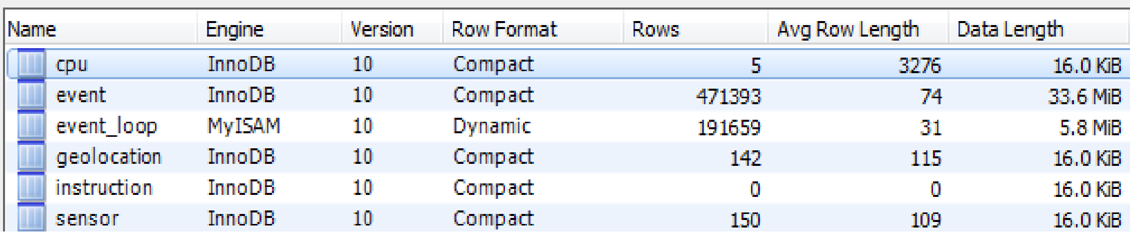
Data and material reused:

a) Template reused for the web application from the work done in the Opening Hours Based Pub Guide in Open Street Map subject of the Geospatial Technologies Master.

It has been used as a skeleton of the web application, this design is responsive to the screen's size so it is worth for using it with smartphones. It has been built employing jquery and jquery mobile css and JavaScript libraries.

b) Historical occupancy data got from the Smart Park Ontinyent sensors and collected by the Hall Effect group.

This data is stored in a Mysql database. This database schema is composed by five tables whose names are CPU, event_loop, event, geolocation, sensor and instruction. Table 4.1 is showing a summary of these tables. The occupancy data is stored in the event (for smart parks) and event_loop (for loop park) tables. Further it will be explained more about this database structure.



Name	Engine	Version	Row Format	Rows	Avg Row Length	Data Length
cpu	InnoDB	10	Compact	5	3276	16.0 KiB
event	InnoDB	10	Compact	471393	74	33.6 MiB
event_loop	MyISAM	10	Dynamic	191659	31	5.8 MiB
geolocation	InnoDB	10	Compact	142	115	16.0 KiB
instruction	InnoDB	10	Compact	0	0	16.0 KiB
sensor	InnoDB	10	Compact	150	109	16.0 KiB

Table 4.1 Summary of the database tables

c) Current web application of the Smart Park Ontinyent. Some features of the mapHandler class of the current service were taken to visualize the parking locations and its information there.

The map element was taken with the following methods already existing:

-Map handler class with the map variables and the following methods:

- Init method to create the map at some events at concrete map centre and scale.
- Refresh parking cluster information each time the map bounds have changed.
- Add clusters markers and generate the info windows when clicking on them.

Also it was used the getspots API of the SPO service: This API generates a json parking array. It is used in the new web service for getting the parking array that contains all the parking attributes, including the parking status when a get request is sent.

4.1.2-Functionality added to the current web service

The following features and functionality has been incorporated to the SPO web application:

1. **Geolocation**

User current location. Use location information is very useful in terms of parking information, this information can be used for measuring the distances to a park or to display the map centred on this location and have an overview of this position. This service gets the user current location by the html5 method getCurrentLocation(), the client browser will ask for the permission of the location information to the user and afterwards the user location will be shown on the map. The location accuracy depends on the device and the positioning method used to get the user current location, here it is shown some different positioning techniques used for the laptop and mobile users and its respective accuracy (Peter Brida, 2003):

<u>IP-based</u>

<u>Satellite based positioning.</u>

- GPS: Global Positioning System. 2-15m

<u>Network Based Positioning:</u>

-Cell of Origin: Approximating users' position based on the cell (tower) serving them 150m-3km.

-Time of Arrival: Calc. of device/base station distance based on propagation time of signal (done by service provider) 50-150m.

-Angle of Arrival: Recording received signals from different base stations (by device)>150m.

-Enhanced Observed Time Difference (EOTD): similar to ToA, but calculations done on the device instead.30-100m

2. Distance to parks function

In order to optimize the parking assistance of the SPO, it would be very useful to have a second parameter for parking decision, apart from the park status. That would be the distance to a park. Distance calculation from user location to the parks has been a functionality studied and carried out in this work.

Aside to forecast service which is explained later, this service could be done either in the client side or in the server side. To achieve this functionality, these are the possibilities that have been considered.

I-PostGIS and pgrouting functions.

PostGis is a well known spatial extension of Postgresql relational database management system. Pgrouting has the functions for topological network creation and analysis.

The first approach to get distance matrix from a location to park locations was to create the topology network of Ontinyent city, then apply the shortest path function for each of the parks and get these distances. The Ontinyent street network it is available from the "Cartociudad" geoportal of the Spanish Spatial Data Infrastructure.

Advantages: personal geodatabase for the system to update and modify the data.

Disadvantages: Data development, database installation in server, complication of the system architecture. Network construction for each new smart park in a different city and implementation.

II-Google directions API. Distance matrix

It is a distance matrix service based on Google maps cartography which gives the desired functionality for the system. The problem found was the usage limits of this API.

Google distance matrix limits.

100 elements per query.

100 elements per 10 seconds.

2,500 elements per 24 hour period.

Having 18 parks to measure, then it is 2500/18= 138 times/day this function could be used in our system. This fact prevents from using it as distance calculator service and makes it not valid for this work.

III-MapQuest directions web service

MapQuest directions is a service for matrix route as well as matrix distance of open street maps. Aside from the Google's one, it has no limit usage, but the amount of elements per query has to be less than one hundred. Also registration it is required for using this web service.

It is based on Open Street Maps data and it works fine for our requirements. Then it has been used for this application.

For the matrix request, it must be sent an http request with the following URL structure:

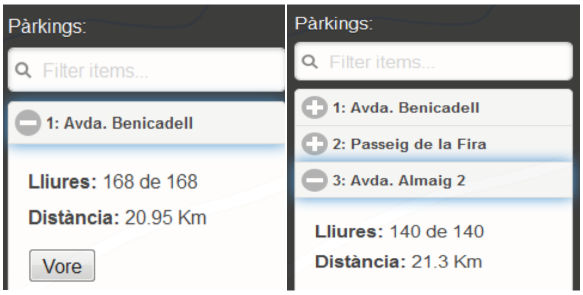
```
http://www.mapquestapi.com/directions/v1/routematrix?key=XX
&callback=renderMatrixResults&json={locations:["38.781832099999995,-
0.7867765",”38.820739,-0.596641”],options:{“allToAll:”false}}
```

The response to that request comes with the next structure:

```
renderMatrixResults({"allToAll":false,"distance":[0,13.01],"time":[
0,1325],"locations":[{"latLng":{"lng"....};
```

The distance field is in miles and time comes in seconds.

In JavaScript a function has been written, which constructs the right query structure for getting all distances. Also it reads from the response the distances and assigns them to the parks as well as converts the distances from miles to kilometres. Figure 4.1 shows two distances from a concrete location to two different parks.



Pàrkings:

Filter items...

1: Avda. Benicadell

Lliures: 168 de 168

Distància: 20.95 Km

Vore

Pàrkings:

Filter items...

1: Avda. Benicadell

2: Passeig de la Fira

3: Avda. Almaig 2

Lliures: 140 de 140

Distància: 21.3 Km

Figure 4.1 Distance in Km of two parks to user's location

3. **Geocoding search bar**

Provided by Google geocoding API. Geocoding is a process of describing a location (a point, a house, a town, a street, or a city) by for example a name or an address. This process matches those descriptions with geographic elements like polygons, lines or points.

Then the users can introduce a concrete address where they pretend to go and geocoding process will return coordinates of the place internally. The geocoding action will display the map centred and it will add a marker in that address in order to see the surroundings of that address. This event will store this location for use it in the routing service later on. The code Address function method does the geocoding service and the show Location method of the mapHandler class centres the map and adds the marker.

4. **Parking list panel**

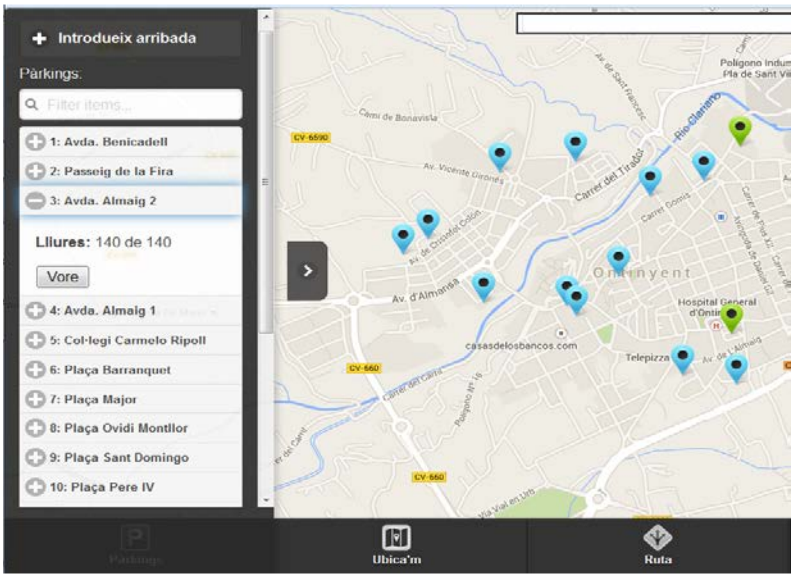
By issue of comfort, when this application is used by a cell user, it is faster and easier to see the parking information in a park list rather than go marker by marker checking this information in a map, for this reason it has been added to the current web application.

In this panel the user will see a list of the set of parks inside a collapsible panel in a better way of a map for a quick overview of the information and easier to interact with small screens (Figure 4.3). There it is shown the occupancy status of a parking and the user can go to the chosen parking in the map by clicking on 'move to' button. A filter box input is included as well, where the parks can be filtered by its name. Also by clicking in one parking, the user selects it for following operations like routing to the parking or f. Within this panel there are also the inputs fields, for selecting the parking you want to park in and the date and time you want to do it, in order to use these data for routing and forecasting services. The parking list is created by the createResultList() function from an array of parking objects in as an argument (Figure4.2).

```
function createResultList(parkings) {

        Here the result list is created from a
        parking cluster object array

}
```

```
{"spots":[],"clusters":[{"latitude":38.820739,"longitude":-
0.596641,"count":168,"free":78,"name":"Avda.
Benicadell","type":"staticpark","object":null,"spot_id":0}]}
```

Figure 4.2 Parking cluster object used by createResultList function



+ Introdueix arribada

Pàrkings:

Filter items...

- 1: Avda. Benicadell
- 2: Passeig de la Fira
- 3: Avda. Almaig 2

Lliures: 140 de 140

Vore

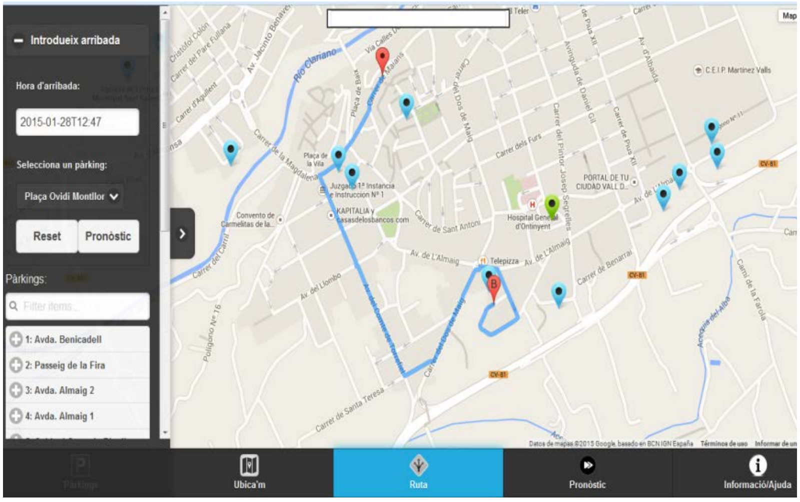
- 4: Avda. Almaig 1
- 5: Col·legi Carmelo Ripoll
- 6: Plaça Barranquet
- 7: Plaça Major
- 8: Plaça Ovidi Montllor
- 9: Plaça Sant Domingo
- 10: Plaça Pere IV

Figure 4.3 Parking list inside a panel

5. **Routing**

Imagine you do not know the street network of a city and the names of the different parks and places are not familiar to you. And this is where comes in the usefulness of this service, provided by Google directions API, it will display the route which should be followed in order to get the chosen parking.

The Google directions API needs for routing an origin and a destination. The origin is a variable that is taken from the user's location or from the address which has been introduced by the user previously. The destination it is a variable that comes from the coordinates of the parking selected in the panel list. Setting these two variables the service will display the route that the user should follow in order to get to the wanted parking. If the status of the request is ok the client gets a result object, the application will show the distance feature of this result object (Figure 4.4). Also the application will use the duration (time) of the route from this result object and it will be added automatically to the time arrival in the input box as a delay, for the occupation prediction service. Since this project uses google maps api for map displaying, the route visualization can only be drown by its google direction service, if there is a migration to for example Open Street Map cartography provider, then alternative routing services could be used. In the last stage of this thesis, and still not integrated, a navigation service will guide smartphones users making a call to the maps application for android.



Hora d'arribada: 2015-01-28T12:47

Selecciona un pàrking: Plaça Ovidi Montllor

Figure 4.4 Route result from input location to a selected parking

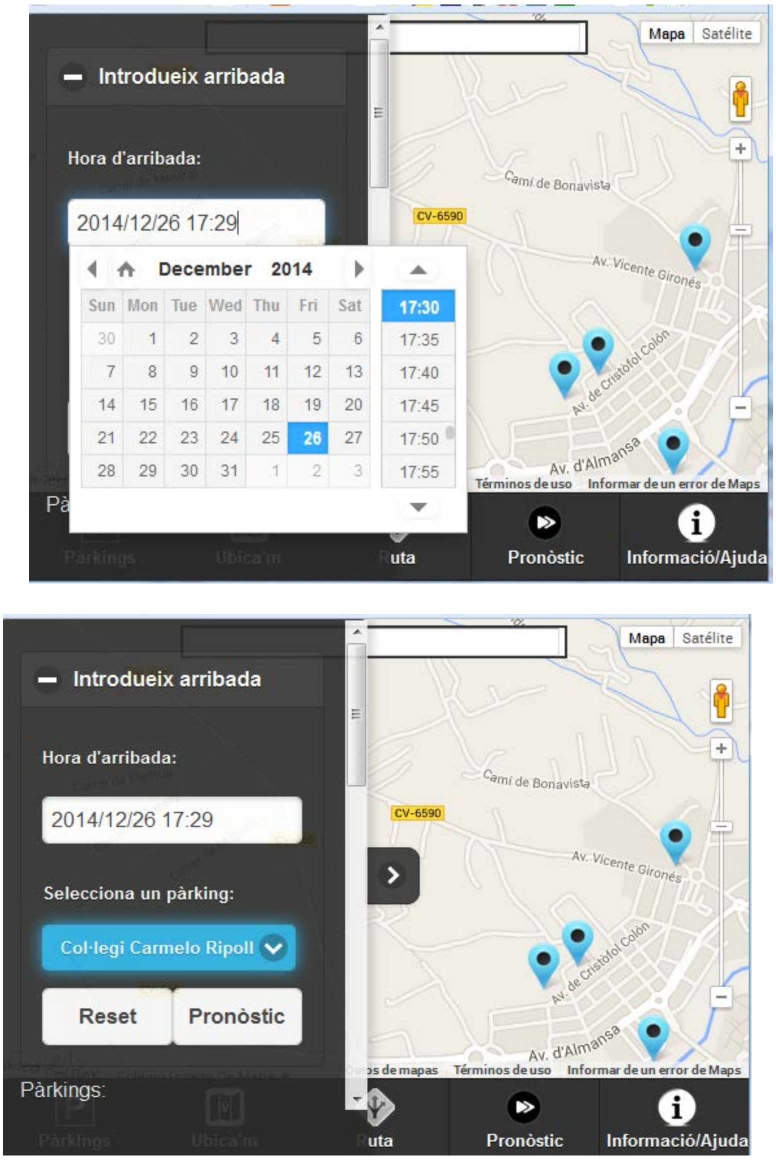
The routing service is carried out after clicking on the "Ruta" button and this event triggers the calcRout method of the mapHandler class.

```
calcRoute : function (lat, long) {

        Here with the paramaters latitude and longitude of
        the parking, the route is calculated and displayed.

}
```

6. Prediction Service

Finally, the last service joined on the application and the most relevant is the prediction service, as shown in the introduction of this work, a user that is wondering about the parking status for some future point, using this service it can find out the predictions according to the past observation.

The prediction service takes two inputs, time and parking of arrival from the user thorough the client application (figure 4.5), then, they are sent to the server and the server afterwards processes the data, returns the number of free places that there will be in the parking according to the model built for that parking.



Introdueix arribada

Hora d'arribada: 2014/12/26 17:29

December 2014

Sun	Mon	Tue	Wed	Thu	Fri	Sat
30	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31	1	2	3

17:30
17:35
17:40
17:45
17:50
17:55

Selecciona un pàrking: Col·legi Carmelo Ripoll

Figure 4.5 Selecting time of arrival and destination park

```
http://localhost:9999/services/hello/forecast?start=2014-12-26T17:30:00&park=5
```

Figure 4.6 Example of get request passing parking and time-date as attributes

The jetty server will have to send these attributes, park and date-time (figure4.6) to R, who is responsible of processing and returning the predictions for the parking occupancy status. This communication will be done through java and R server.

This “get” request is processed by java in the server side. The date-time attribute will be handled by a java class called DateTimeInfo. To predict the parking occupancy, the model implemented in R needs some date-time attributes. They will be extracted by the DateTimeInfo class. It will get the following information:

1.-Day of the week of the start attribute (Sunday, Monday...)

2.-Is week day. True or false value depending of week day or weekend.

3.-Hour of the day and minutes in a decimal format.

Then java has to establish communication with R through an instance of class Rconnection. And finally “.eval” method will process the r scripts like the command line does (figure 4.7).

Last thing is to give the parameter back to the client side as a json object (figure 4.8).

```
RConnection rc = new RConnection();
rc.eval("library(C50)");
rc.eval("d=data.frame(c("+a+"),as.factor("+b+"),c("+c+"))");
rc.eval("colnames(d) <- c(\"dayOfWeek\", \"weekDay\", \"hour\")");
int[] p =
rc.eval("as.numeric(predict(model,d,type=\"class\"))").asIntegers();
```

Figure 4.7 Java code chunk for establishing the R-java communication and passing the arguments

```
{"forecast":{"forecast":"44"}}
```

Figure 4.8 A get request like code nº 4.7 produces as a result this json

SECTION 5:
CONCLUSIONS AND FUTURE WORK

A web application able to handle date-time requests for parking occupancy predictions was made in order to allow users to have this information. Also new functionality added to it for parking assistance task. Routing, geolocation, and better visualization and interaction are these new functions. All of them have been tested in order to ensure the well working of the application. For newer versions some extra functions are navigation and parking lot dissemination. But with the current version the goals pretended are entirely accomplished. For an evaluation of this application it should be considered a non functional test of it, thinking about an application acceptance test as the best, because only few functions implemented so far for an usability test and there is already a help guide also included in the application. On the other hand integration and compatibility were carried out in this work.

Regarding the parking occupancy modelling, it has been demonstrated that it is possible to predict with accuracy enough future occupancies status in time. It is thought that larger periods of observations are necessary in order to evaluate the impact of external factors on parking predictions. Currently we have less than one year of historical data available, which is not enough to evaluate the impact of certain events or holidays on specific parking lots. As soon as such events or holidays are repeatedly present in the historical data, their importance or unimportance should be visible.

Apart from this consideration it is presented a list of future work that should be implemented to improve a lot the smart park performance.

Automation of the process

It would be very useful to have implemented an algorithm that performs the modelling process automatically. Just dumping the historical occupancy data of a park and some extra information, the algorithm was able to generate the best model that characterizes our data and makes predictions to a given dates. It would allow working with data modelling and prediction without knowledge of data mining and time series analysis, in addition to avoid repeating all the commands and process for each park or a new one.

Dynamic modelling

Cities are continuously moving, changing; ones faster or slower than others, but they do. The mail office has change location, or a new school has been built. So, a parking behaviour model can work very well now, but half year later maybe not. A good improvement for adapting that to possible new facts is to take into account the continuous changes in behaviour. A dynamic modelling would be a solution for this handicap.

As a first approach the way to do this might be the auto modelling every so often. This often times should be set by the smart park manager. The model would be created leaving out the oldest records and the more recent would be used for this new model. So a more realistic model would response better to the new scenarios in the parking behaviour.

Automatic Park allocation algorithm

Finally, the last consideration of improving the performance of the system would be to combine some variable of the current or future parking situation to allocate the user to a park. The variables are the distance in time to the different parks (they are known with the current platform) and the occupancy status of the parks at the same time. So the easier park allocation is the shortest park having free places.

This approach for park allocation can go further and more complicated using for example the forecasts for the time a user spends to get the park. This is, if the user needs 5 minutes to get the parking, and the forecast after 5 minutes says that the parking would be full, then the algorithm has to refuse this park. Other consideration is for instance the use of thresholds in the occupancy, this means, not to allocate parking having the 95% of capacity full, because possibly after few minutes could be no parking there.

In this area, it was tested for park allocation the concept of service area, a type of network analysis for determining the region that encompasses all accessible streets (streets that lie within a specified impedance). In this case were created the services area using the travel time in car as impedance attributed, and was created using the gvSIG, GIS open source software. As shown in figure 5.1, the relatively small dimensions of Ontinyent (40.000 population) result that in few more than 5 minutes almost every location within the city can be reached. So it was decided to leave this work out for other situations as could be the setting up of a smart park in a bigger city or area.

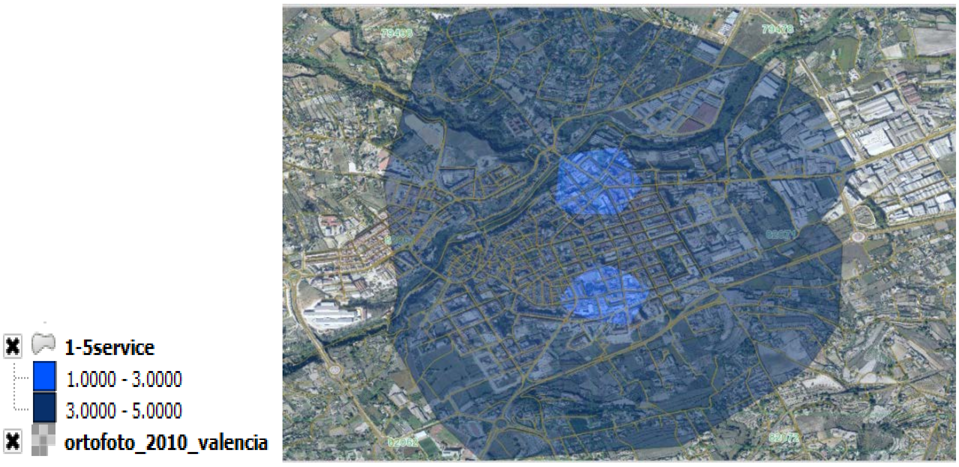


Figure 5.1 Service areas for CEAM and Sufragistes parks, 1min and 5min by 15km/h

REFERENCES

Akram, M., Hyndman, R. & Ord, J. 2009. Exponential smoothing and non-negative data. Australian and New Zealand Journal of Statistics 51 (4), 415–432.

Brída Peter. Location thecnologies for gsm, 2003. Department of Telecommunication, Faculty of Electrical Engineering, University of Žilina, Moyzesova

Cheung, S.Y., S. Coleri Ergen and P. Varaiya, 2005. Traffic surveillance with wireless magnetic sensors. Proceedings of the 12th ITS World Congress, November 6-10, 2005, San Francisco, pp: 1-13.

Corchado, E. and Fyfe, E. (2004). Redes neuronales artificiales. En J. Hernández Orallo, M. J.

De Livera A.M, R J Hyndman and R.D Snyder, 2011. Forecasting time series with complex seasonal patterns using exponential smoothing

Deshpande, B. 2011, *Immediate sharing,* What's New, Retrieved 23 December 2014, <http://www.simafore.com/blog/bid/62482/2-main-differences-between-classification-and-regression-trees/>.

Freund, Y. (1995). Boosting a weak learning algorithm for majority. Information and Computation, Vol. 121(2), pp. 256-285.

Freund, Y. and Shapire, R. E. (1997). A decision-theoretic generalization of on-line learning and an application to boosting. Journal of Computer and System Sciences, Vol. 55(1), pp. 119-139.

Friedman, J., Hastie, T. y Tibshirani, R. (2000) . Additive logistic regression: a statistical view of boosting. The Annals of Statistics, Vol. 28(2), pp. 337-407.

Hernández Orallo, J., Ramírez Quintana, M. J. and Ferri Ramírez, C. (2004). Introducción a la minería de datos. Madrid: Pearson Educación S.A..

Hyndman R J, Forecasting functions for time series and linear models, 2014. A manual for forecast package in R. Version 5.6.

Hyndman R.J, AB Koehler, 2006. Another look at measures of forecast accuracy. International journal of forecasting

Idris, M.Y.I., Y.Y. Leng, E.M. Tamil, N.M. Noor and Z. Razak,, 2009. Car park system: A review of smart parking system and its technology.. Inform. Technol. J., 8: 101-113.

Jensen, J. R. (2005). Introductory digital image processing (3ª ed.). Upper Saddle River (E.U.A.): Pearson Education, Inc.

Kunjithapatham A., A. Tripathy, Cheong, C. Chiriyankandath and T. Rao. Predicting Parking Lot Availability in San Francisco.

McCleary e Hay, R. McCleary, R. A. Hay, 1980 Applied time series analysis for the social sciences. Sage, Beverly Hills [u.a.], 1980. ISBN0803912056.

Mimbela, L.Y. and L.A. Klein, 2007. A summary of vehicle detection and surveillance technologies used in intelligent transportation systems. New Mexico State University, Tech. Report, 2007. http://www.fhwa.dot.gov/ohim/tvtw/vdstits.pdf.

Mitchell T., 1997. Decision Trees Learning. In *Machine Learning*, McGraw Hill, pp 52-81.

Oates, T. and D. Jensen. Large datasets lead to overly complex models: an explanation and a solution. In Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining. August 1998

Quinlan, J. R. (1996a). Bagging, boosting and C4.5. Proceedings of the 30th National Conference on Artificial Intelligence and the 8th Innovative Applications of Artificial Intelligence, 4-8 August, Portland, Oregon, E.U.A., pp. 725-730.

Ramírez Quintana and C. Ferri Ramírez, (Eds.), Introducción a la minería de datos. Madrid: Pearson Eeducación, S.A., pp. 327-352.

Reinstadler M., M. Braunhofer, M. Elahi, F. Ricci, 2013, *Predicting parking lots occupancy in Bolzano.* Institute of computer science, Free University of Bolzano, Italy

Rojas Daniel, 2006. *Revenue management techniques applied to the parking industry*. MA thesis. College of Engineering. University of South Florida

Sherrod, P. H. (2014) DTREG Predictive Modelling Software. www.dtreg.com

Smart City. (n.d.). *In* Wikipedia. Retrieved January 21, 2015, from <http://en.wikipedia.org/wiki/Smart_city.

Tullis, J. A. and Jensen, J. R. (2003). Expert system house detection in high spatial resolution imagery using size, shape and context. Geocarto International, Vol. 18(1), pp. 5-15.