University of Pisa

Master's degree in
Data Science & Business Informatics

Data Mining 2:
advanced topics and applications

Occupancy Detection

Carlo Alberto Carrucciu [533967]
Tommaso Cavalieri [597707]
Giacomo Lo Dico [600002]

June 22, 2020

Contents

1 Data Understanding

The dataset used for the analysis presented in this report contained experimental data recorded in a building with the purpose of classifying its room occupancy. Even though these records were initially collected for such task, further assignments were completed and the purpose of this report is to unveil them. The set of data provided was already divided into three different files: one training and two test sets. Their division was probably due to the different periods during which they were registered, which can be appreciated thorugh their temporal dimension, since they contained contiguous records if considered separately, but there were some hours of missing data between one another. In order to get a more general picture of the situation, such datasets were merged into a unique one, composed of **20560 records**. From that were then derived two new sets, *training* and *test*, with a 70-30 split, so that the work to be carried out takes place on a pure generalization of the data. After the split, the distribution of the target variable **Occupancy**, both in the training and test set, was checked to still be the same as the one in the original dataset. The features recorded and used for this analysis were the following:

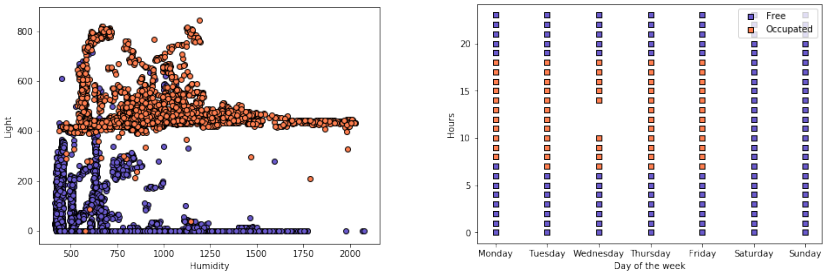
- **date** recorded as *year-month-day hour:minute:second*,
- **Temperature** in *Celsius*;
- **Humidity** in percentage;
- **Light** in *Lux*;
- **CO2** in *parts-per-million*;
- **HumidityRatio** in *kgwater-vapor/kg-air*;
- **Occupancy**, binary for the room status: 0=*not occupied* and 1=*occupied*.

1.1 Data exploration

	Temperature	CO2	Light	Humidity	HumidityRatio
Temperature	1.000000	0.449714	0.691153	-0.156826	0.205318
CO2	0.449714	1.000000	0.450187	0.299922	0.477996
Light	0.691153	0.450187	1.000000	-0.027657	0.226050
Humidity	-0.156826	0.299922	-0.027657	1.000000	0.932759
HumidityRatio	0.205318	0.477996	0.226050	0.932759	1.000000

Table 1.1 Correlation matrix

Before splitting the dataset, some of its characteristics were explored, such as how the distribution of the target attribute **Occupancy** was affected by the others, from which it emerged that the values of the variable **Light** appeared to be very influencing. In fact, it is possible to observe from the scatter plot in Figure 1.1 that two main levels can be identified for the values of **Light**. The areas of such levels are very well separated and the impact on **Occupancy** seems to be very high, which means that basing the decision whether to classify a record as *occupied* or not on this attribute would likely give good results. Further inspection will be provided in Section 3.



(a) Light vs Humidity (b) Occupancy distribution

Figure 1.1 Basic plots

Another attribute which needed some further investigation was **date**. After converting it to the right format it was possible to separate its information into two new attributes: **Date** (which indicated day, month and year) and **Time** (which indicated hours, minutes and seconds). For what concerns the former, the division between weekdays and holidays could be really interesting, even more than the date itself, and was therefore exploited in some tasks. In fact, by observing for example the values of the features during weekdays it could be possible to compare them with other weekdays' values recorded during another period of the year. Since from the data emerged that the building was not occupied during weekends and during the night time, it could be assumed that the building where the data were recorded contained mainly offices. In Figure 1.2 are shown two histograms of the records' distribution over time: the one on the left represents a week divided day by day, while the one on the right refers to a single day, divided hour by hour. The two different colours represent the values of the variable **Occupancy**.

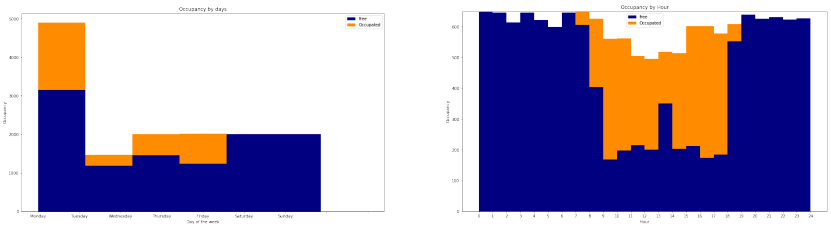


Figure 1.2 Occupancy by day and by hour

1.2 Dimensionality reduction

Even though the dataset used for this study did not have a lot of features, it seemed interesting to see if it was possible to reduce such dimensionality before continuing with other tasks. In fact, redundant or highly correlated features could be discarded in this way, i.e. the attribute **HumidityRatio** is a derived quantity from **Temperature** and **Humidity** and therefore could be redundant with such variables. The methods described in this paragraph were executed on all the attributes except the temporal ones. The first explored feature selection approach was based on the **variance** of the attributes, removing all the ones not meeting a prefixed **threshold** (=1); by doing so only **HumidityRatio** would be discarded, in fact its domain was the interval [0.002,0.007], which caused a very low variance for such attribute.

Variable	Score
Light	73703.44
Temperature	6658.29
CO2	4812.72
HumidityRatio	1070.50
Humidity	34.88

Table 1.2 ANOVA F-values

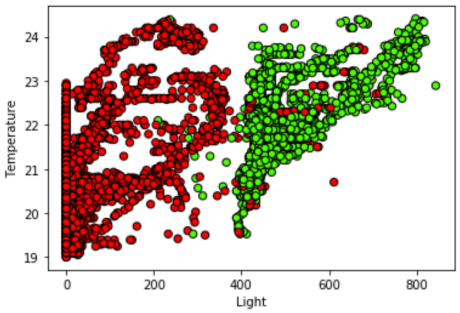
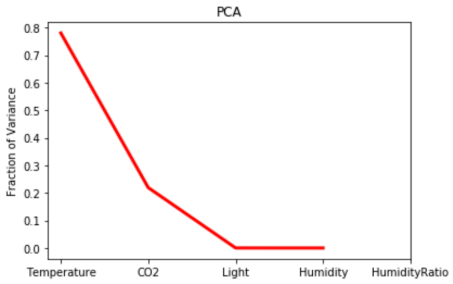


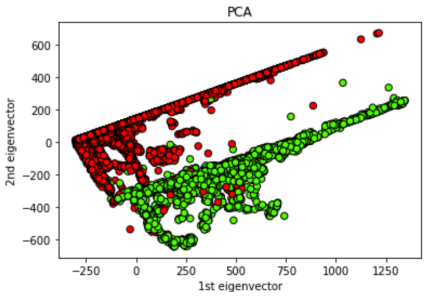
Figure 1.3 Light vs Temperature

Univariate feature selection was also implemented, assigning to each feature a score equal to their **ANOVA F-value** and then selecting the k best-scoring attributes to be kept. The results obtained are presented in Table 1.2, from which it can be appreciated that with this approach the variable **HumidityRatio** would prevail over **Humidity**. Another important conclusion that can be drawn from this table is that if we want to represent the separation between the records in a 2D plot, the two variables that could capture the most such characteristic are **Light** and **Temperature**, as

shown in Figure 1.3. Some of the classifiers that will be explored in Section 3 made it possible to do a **recursive feature elimination**: in this approach every feature receives weights according to the model (e.g. the coefficient of a linear model or the feature importance in Decision Trees) and then at each step the least important feature is pruned until the desired number of attributes is obtained. Decision Tree confirmed what was already anticipated in the previous paragraph, giving more importance to **Light** and selecting it as the only variable to be kept. Lasso kept **Ligth** and **CO2**, while the logistic regression saved only **Temperature**, which led to a worse classification than the others. The **Principal Component Analysis** was then applied, whose result are presented in Figure 1.4. It can be appreciated how **Temperature** and **CO2** seemed to capture most of the variance of the records and the the separation appeared to be clear on the main component too. It also seemed very interesting to observe how the classifiers worked in two dimensions, trying to define the decision boundary for each one of them. That is the reason why decision trees, KNN, Naïve Bayes and logistic regression were implemented, comparing the boundaries obtained for the two best features (Figure 1.5a) and for the two *eigenvectors* of the PCA (Figure 1.5b).

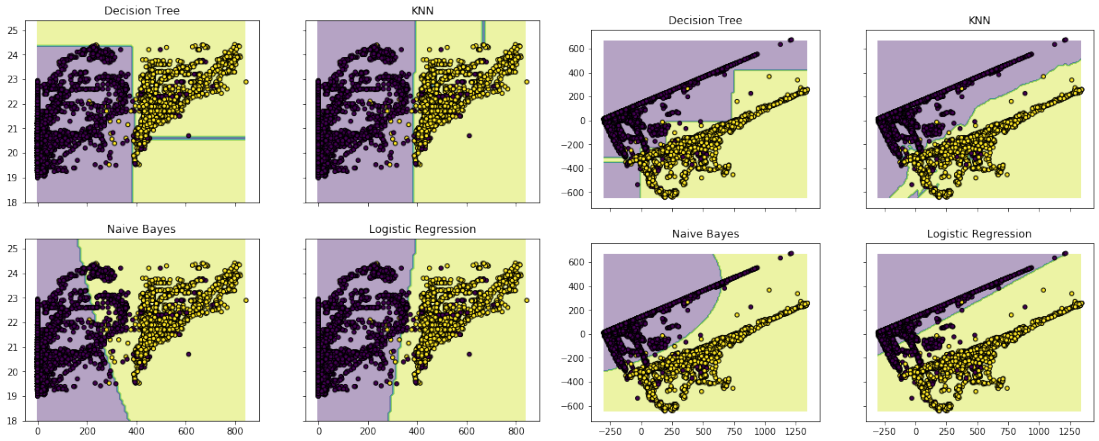


(a) Explained variance



(b) 1st vs 2nd eigenvector

Figure 1.4 PCA for 2 components



(a) Best 2D scatters

(b) PCA main component

Figure 1.5 Decision boundaries

Taking into consideration all the observations made, the decision was to use the result obtained from the variance threshold method and exclude **HumidityRatio** for the tasks that could be affected by it. Such decision was made because of its redundancy with **Humidity**, as highlighted by Table 1.1, and the fact that its values were calculated from the values of temperature and relative humidity and so were not a 'pure' measure for such records, but an aggregated value. The attribute **Humidity** was also preferable for the sake of comprehensibility, since it was already expressed in a 0-100 scale.

1.3 Outliers detection

The first and most basic outlier detection method to be applied was the visual approach: boxplots were generated, as shown in Figure 1.6, individuating a total of 1945 outliers in the whiskers of the plots made for the 4 attributes considered. Simple scatter plots between the variables were considered too. In the following paragraphs more advanced outlier detection techniques will be discussed. Such methods are not just simple visual approaches, but they exploit other characteristics of the records, such as distances between the records or others measures and statistics.

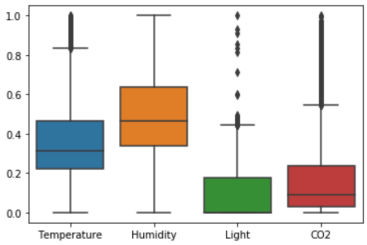


Figure 1.6 Boxplots

1.3.1 Distance-based

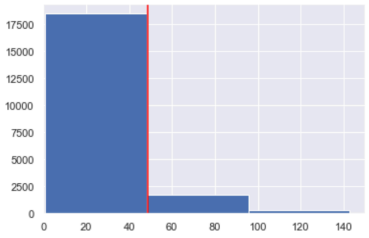


Figure 1.7 Distance-based outliers

The first advanced approach was distance-based, more specifically the one assigning to each point its distance from its kNN as outlier score, using k=100. This approach classified 2022 points as outliers, with an average outlier score of 81.79, against the 11.13 average score of the normal data. Another implemented distance-based approach was the one that considers every point as a vertex in a graph and uses their in-degree number as outlier score; such value is in fact equal to the number of reverse kNNs of the corresponding point. Using k=100 and a threshold equal to 50 for the number of RkNNs to detect outliers, 1459 anomalies were detected and their distribution is plotted in Figure 1.7.

1.3.2 Density-based

Unfortunately, distance-based approaches are very sensitive to density variations, which is why also density-based tecnhiques were tested. The foundation for this type of approaches is trying to compare the different densities of the points with the ones of their local neighbours, assuming that normal data would have similar values, while outliers are expected to have a considerably different density from their neighbours. The first explored method was DBSCAN; in fact, the clustering algorithm not only works without needing the number of clusters in advance, but it also identifies outliers as points not belonging to any of the individuated clusters. After looking for the best values for the parameters using the *knee method*, min points=8 and epsilon=0.04 were selected; the implementation of such algorithm resulted in the labelling of 125 point as outliers. The main problem with this type of approaches is how to compare the neighborhood of points from areas of different densities, which is why a technique using relative density was also implemented, namely the **Local Outlier Factor**. Such values are computed as the average of density ratios of a sample p and the densities of its nearest neighbours. Labelling the points having a LOF larger than 1 as outliers, 2840 of them were detected.

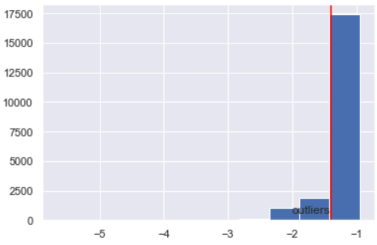


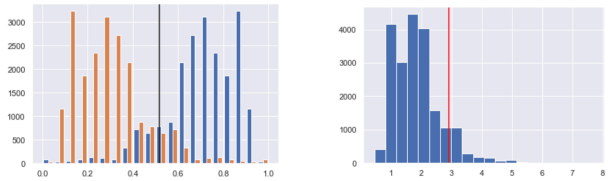
Figure 1.8 Density-based outliers

1.3.3 Angle-based

Another class of outliers detection techniques is the angle-based, whose idea is to examine the spectrum of pairwise angles between points, detecting outliers as the ones with a highly fluctuating spectrum. The **ABOD (Angle-baset Outlier Degree)** algorithm was therefore implemented. The selected number of neighbors was 20, contamination (the proportion of outliers to be detected in the data set) was estimated equal to 0.1 thanks to the previously observed results and the implemented method was the fast one, which only considers k-neighbours of the training points to reduce the computational cost, which could otherwise be up to $O(n^3)$. Unfortunately, this approach performed poorly in identifying outliers, which might be due to the fact that such techniques are usually useful for dealing with high-dimensional datasets, while the one analysed here presents only four features.

1.3.4 NN-based

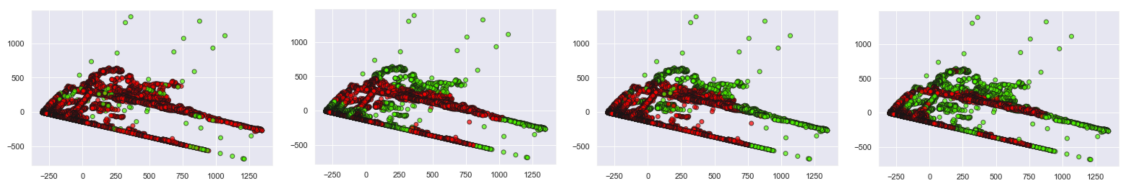
The last implemented method was the **Autoencoder**, which represents each record through the compression and decompression of a Deep Neural Network. Such technique is based on detecting outliers by observing the reconstruction error of the DNN and labelling as outliers the points that generate a large value for such error. The DNN used for the analysis was made of 4 hidden layers composed as follows (4,2,2,4) and after 50 epochs, 2056 outliers were detected, whose distribution is shown in Figure 1.9.



(a) Probability (b) Anomaly score

Figure 1.9 Autoencoder outliers distributions

Even though these techniques helped to identify some possible anomalies in the records, the detected outliers did not seem to differ much from the real distributions, hence it was decided not to eliminate such records in order to avoid losing their temporal contiguity. The only records that were eliminated were outliers in the **Light** distribution (only seven records), since such variable appeared to be highly influencing in classifying **Occupancy**. Moreover, when repeating all the presented outliers detection techniques after the elimination of such records, they all detected a smaller number of outliers apart from the one based on RkNN, which indicated that such records were labelled as outliers by almost every explored method. In Figure 1.10 are shown the plots using the PCA main component to represent the outliers identified by the various explained techniques.



(a) DBSCAN (b) LOF (c) Autoencoder (d) KNN

Figure 1.10 Outliers in the principal component

2 Regression

In this section a regression problem will be addressed, based on two different attributes of the dataset. All the possible pairs of features were explored (excluding the target variable) and observed graphically through a scatter matrix. From the inspected plots the relation that seemed more interesting was the one between **Temperature** and **Light**, which was therefore analysed.

	Linear	Huber
R2	0.478	0.473
MSE	22774.833	22998.213
MAE	121.016	119.727

Table 2.1 Regression evaluation

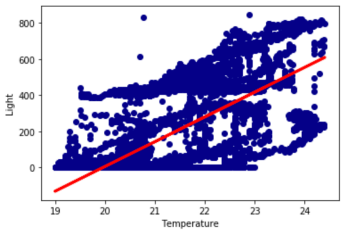


Figure 2.1 Linear regression

The linear regression approach was applied and evaluated through the typical measures: R2 score, Mean Squared Error and Mean Absolute Error, whose values are shown in Table 2.1. **Lasso** and **Ridge** approaches were not applied for this simple problem because they were retained unnecessary. In fact, the peculiarity of such methods is to deal very well with multicollinearity and the interpretability of the model, but these are problems that emerge only when there are more than one independent variables. **Huber** regression was also implemented, which should be an approach similar to the linear one but more robust to outliers. As it can be seen in Table 2.1 the results did not vary much, probably because of the lack of highly influencing outliers in the dataset. The relationship discovered with the linear regression between the analysed features can be expressed through the equation $Light = -2729.67 + 136.8 * Temperature$.

2.1 Multiple Linear Regression

Since the 'runner-up' between the tested 2D problems was the one composed by **CO2** and **Temperature**, it seemed interesting to include in the aforementioned regression problem the variable **CO2**. The so defined problem was used to perform various multiple regression methods such as linear, Ridge, Lasso, Huber, Theil and Bayesian Ridge. Apart from the Theil approach, which proved to be inadapt for the data, the others performed very similarly, which is why it was retained sufficient to present the results only for the multiple linear regression: **R2=0.502**, **MSE=544847.39** and **MAE=669.56**. It can be seen pretty clearly how there is a trade-off between R2 and the error scores, in fact moving from single to multiple linear regression the R2 does improve, but at the cost of having an MSE almost 24 times bigger. Because of such problem, the results obtained by including also **Humidity** were excluded from this report since the gain in terms of R2 was not big enough to justify the cost of the increase of the error measures. Figure 2.2 shows a 3D representation of the variables and the plane described by the equation resulting from the multiple linear regression, which was $Temperature = 20.1 + 0.0031 * Light + 0.00059 * CO2$.

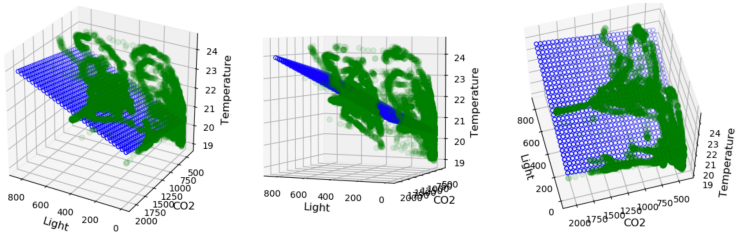


Figure 2.2 Multiple linear regression

3 Classification

The datasets prepared as described in Section 1 were then used for some further analysis. The first task to be implemented was classification: both basic and advanced methods were applied to the data collection, with the goal of constructing efficient models that were able to predict, based on the values of the other features, wether the target variable **Occupancy** took value 0 or 1. The evaluations of the implemented algorithms, which where trained on the aforementioned training set composed of **14384** records and tested on the set composed of **6168** instances, will be presented in this section.

3.1 Basic classifiers

3.1.1 K-Nearest Neighbor

The **KNN** classifier works by computing the distance between the record to be classified and every other record in the training set, basing its prediction on the most common value in the **neighborhood** of the point, precisely between its **K** nearest neighbours. The metric used to compute the distances between objects was the **Minkowski distance**, which allowed to avoid scaling the data. Since the described technique works only with numerical features, the temporal attributes were not considered for this step. The main concern for this algorithm is to find an appropriate value for k, which was searched for with two distinct procedures:

- **plotting accuracy for different values of k**: a range [0,50] was selected for k and the algorithm was implemented on the training set for each odd value in such range (to avoid draws when it comes to voting); this approach highlighted how low numbers of k (<17) performed better in terms of accuracy;
- **grid search**: this hyperparameter tuning technique was applied for various values of k, different algorithms (*Ball tree* and *KD tree*) and types of weights (*uniform* and *distance*); the best resulting combination was obtained using Ball Tree, which gave slightly better results assigning weights equal to the inverse of the distances rather than using the uniform ones.

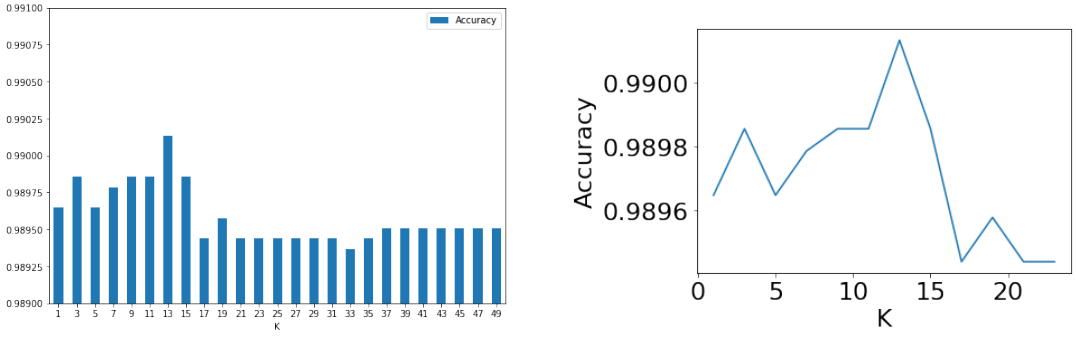
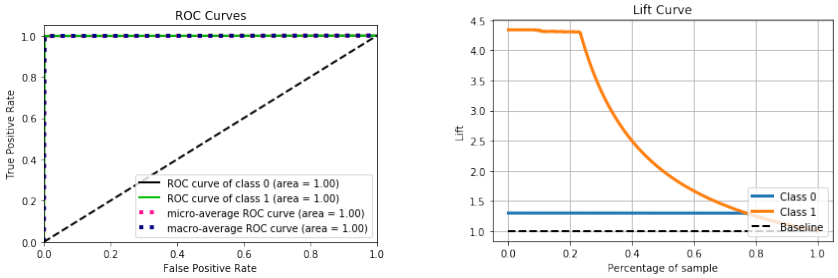


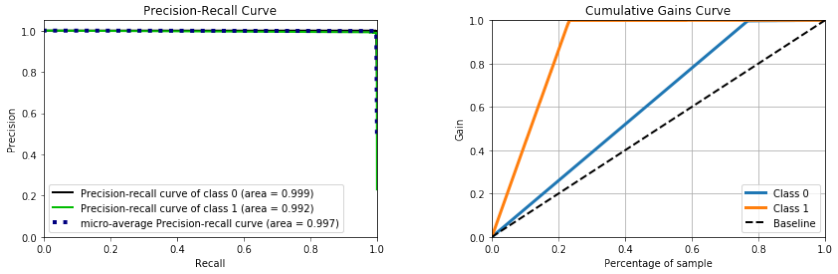
Figure 3.1 Accuracy for different values of K

The best classification performed with the KNN classifier was obtained with the output parameters of the grid search and with k equal to 13. **Cross validation** provided an **Accuracy of 0.9901** (+/- 0.002) and an **F1-score: 0.9863** (+/- 0.003). In Figure 3.2 it is possible to appreciate some graphs of the performance measures for the classification algorithm just described.



(a) ROC curve

(b) Lift curve



(c) Precision & Recall

(d) Cumulative gain

Figure 3.2 Evaluation graphs for KNN

3.1.2 Naïve Bayes

For what concerns the Naïve Bayes classifier, the best results were obtained with a **Gaussian** approach on the attributes **Temperature**, **CO2**, **Light** and **Humidity**. Such approach is one of the possible ways of dealing with continuous attributes for this classification method: it consists in estimating the probability density of the features by making a typical assumption: continuous values associated with each class are distributed according to a normal (or Gaussian) distribution. A **categorical** approach was then used on the binary attribute **Holiday** (a binary variable derived from **Date**) and on **Time**, which had been opportunely discretized after numerous attempts. Pretty good results were obtained by taking into consideration only the hour or counting also the minutes, but the best solution was to rearrange the values by keeping only the hour and the tens of the minutes (*e.g. 15:22 became 152*). Seconds were discarded because they would otherwise have led to *overfitting*. Similar results, with a coincident ROC curve, were obtained using the **Bernoulli** technique, which assumes that features are independent booleans describing inputs and therefore allowed to classify using all the attributes already mentioned, but this time altogether. This algorithm transformed every feature in a dummy variable via one hot encoding, for this reason numerical non-integer attributes were approximated beforehand to avoid overfitting. NB classified efficiently (accuracy higher than 94%) with all three approaches and the resulting ROC curves are shown in Figure 3.3, where unfortunately it is complicated to distinguish the curves for the Bernoulli and Gaussian methods since they almost coincide.

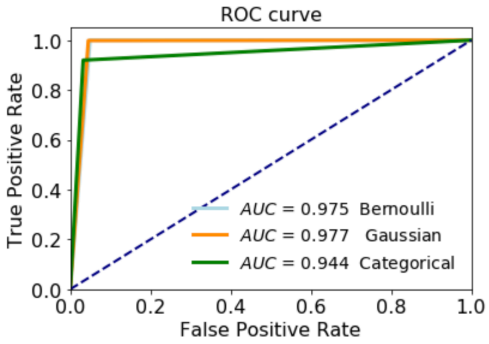


Figure 3.3 ROC curves for Naïve Bayes

3.1.3 Decision Tree

The Decision Tree classifier was then implemented and the obtained results will be exposed in this paragraph. One of its main drawbacks is its difficulty in dealing with continuous features. Obviously this not only applies to continuous numerical attributes, but also to the temporal ones, to which the following transformations were applied:

- **Date** became **Holiday**, which took value 1 during the weekend and 0 in weekdays.
- **Time** became **Hour** after being splitted in 24 intervals and one hot encoded.

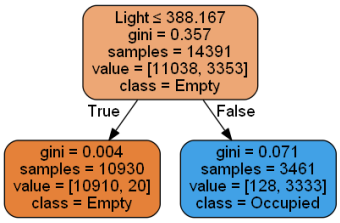


Figure 3.4 Decision stump

The dataset obtained after these transformations was composed of **14384 rows × 30 columns**. Both trees shown in Figure 3.4 and 3.5 reached an accuracy of 99% from which it can be observed that such result was already achieved after the first split, like in a **decision stump**, and the classification was not improved with the following splits. With a threshold for **Light** equal to 388.167, the data set was splitted in **Occupancy=1** when the records assumed an higher number and **Occupancy=0** otherwise. Even though the results were already very satisfying, in order to observe possible changes, the size of the minimum number of records in a leaf was decreased: the best accuracy was reached with 3 as the minimum number of samples required to be at a leaf node. Since neither enlarging the depth nor decreasing the min_sample_leaf parameter increased significantly the accuracy of the model (it improved only by 0.147%), the decision stump would be the best choice to avoid overfitting.

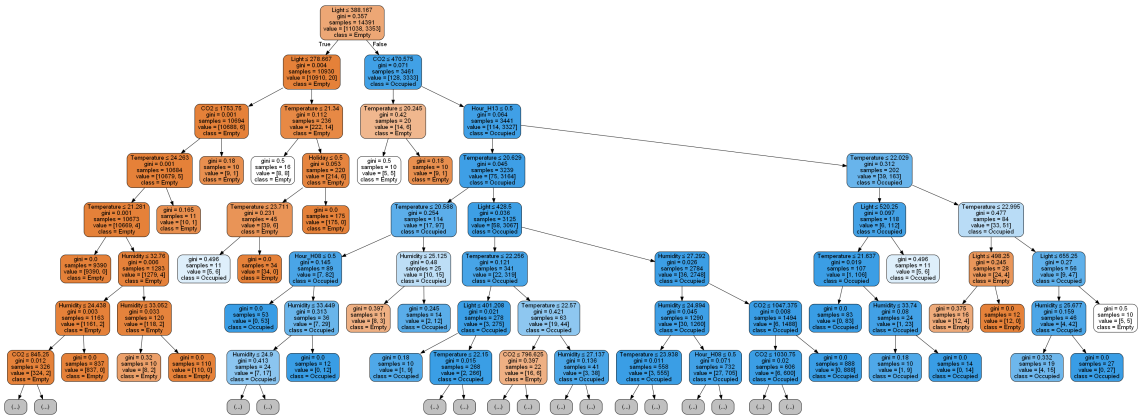


Figure 3.5 Decision tree with depth=7

3.1.4 Logistic Regression

Since the target variable **Occupancy** was binary, the **logistic regression** appeared to be an efficient method to fit a curve to the data. For this task, three different sets of attributes were defined: one including the attributes **Temperature**, **Humidity**, **Light** and **CO2**, one using the main component extracted from the **PCA** described in Section 1.2 and one composed of only the variable **Light**, which seemed to be very good at classifying in the previous paragraphs.

	4 ATTRIBUTES	MAIN COMPONENT	LIGHT
LOGISTIC	98.98%	88.5%	98.25%
PERCEPTRON	97.78%	18.37%	91.6%
RIDGE	98.65%	88.68%	96.82%
PASSIVE AGGRESSIVE	98.98%	86.8%	96.79%
SGDC	98.96%	85.67%	98.93%

Table 3.1 Accuracy comparison

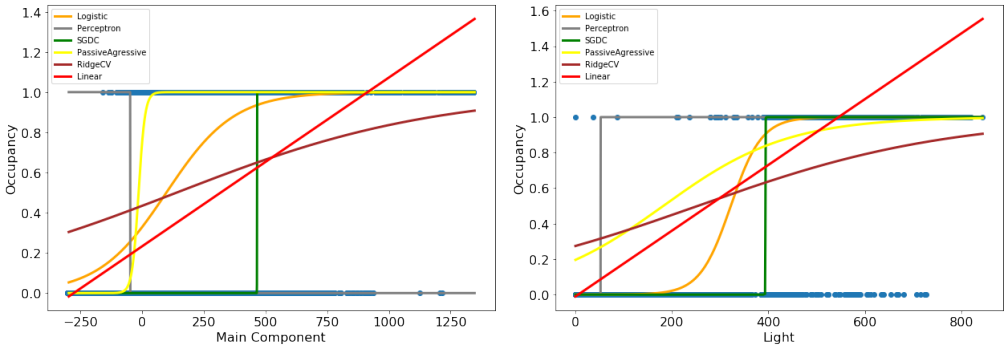
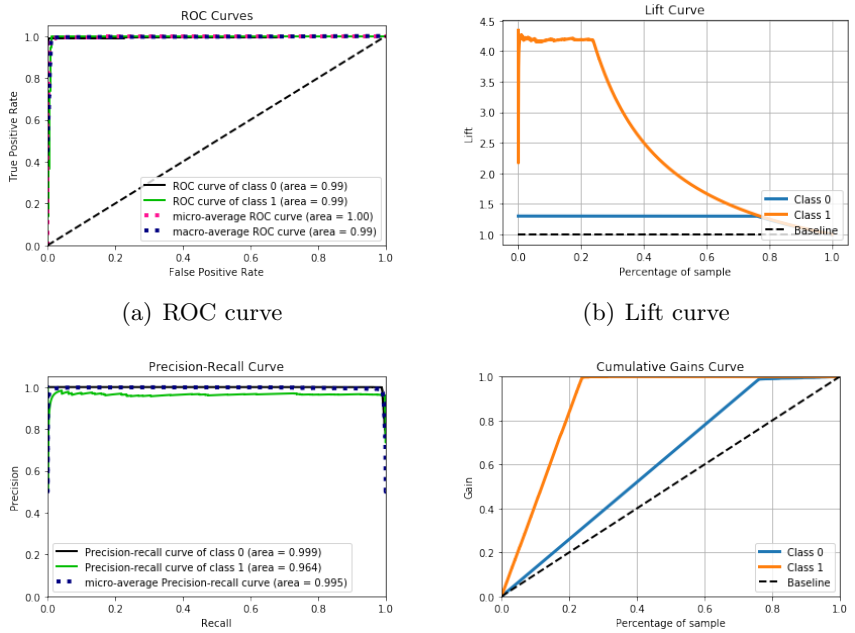


Figure 3.6 Graphical comparison

Other classifiers such as **SGD classifier**, **Perceptron**, **passive aggressive classifier** and **Ridge** were also tested and compared both in terms of accuracy (as reported in Table 3.1) and graphically (as shown in Figure 3.6). SGDC minimizes the gradient and find a clear division in the records, separating dichtomously its output, just as the Perceptron model, and was therefore considered appropriate for analyzing the data in question. Moreover, the cut value identified by such method for the variable **Light** was around 400, the same that could be observed by inspection in the scatter plot in Figure 1.1. In general, all the classifiers were able to predict very accurately when using all the attributes (accuracy higher than 98%), which was not surprising. What might be surprising is the fact that each one of them performed better when using the attribute **Light** as only independent variable instead of the main component found with the PCA. Overall, the Perceptron model was the one that performed worst (even failing to classify when dealing with the main component), but of all of the explored methods the only one that was able to perform as good as the logistic regression was SGDC. Thus the logistic regressor proved its goodness in this type of problems and its performances' evaluations are shown in Figure 3.7.



(a) ROC curve

(b) Lift curve

(c) Precision & Recall

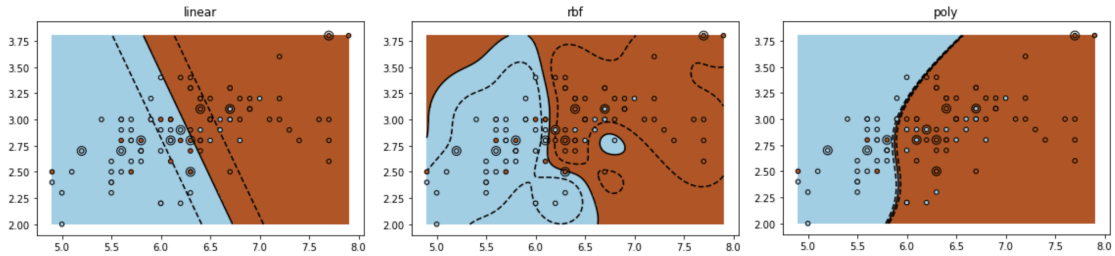
(d) Cumulative gain

Figure 3.7 Evaluation graphs for logistic regression

3.2 Advanced Classifiers

3.2.1 Support Vector Machines

Support Vector Machines are a supervised learning model that represents the decision boundary using a subset of the training examples, known as the **support vectors**. After scaling opportunely the dataset with a **min-max normalization**, **linear SVM** was implemented, which is a classifier that searches for the **hyperplane with the largest margin**. The results were almost immediately satisfying, which made not very meaningful the following parameter tuning. In fact, the worst result was obtained by using the **squared hinge loss function** and C, the parameter representing the penalty of misclassifying training instances, equal to 0.01; such configuration still gave an 98.72% accuracy and a 98% recall. The best outcome was obtained with the hinge loss function and C=100, which performed with a 98.98% accuracy and a 99% recall. The reason why parameter tuning did not change significantly the result can be identified in the fact that the data seemed to be pretty linearly separable, as can be seen in Figure 3.8. This is probably the reason why all the **nonlinear SVM** that were tuned afterwards (with various **kernel functions** such as Radial Basis Function, polynomial and sigmoid) gave worse results than the one just mentioned and thus were discarded.



(a) Linear (b) Radial Basis Function (c) Polynomial

Figure 3.8 Decision boundaries for SVM with different kernel functions

3.2.2 Neural Networks

In this paragraph we will present the task of building an Artificial Neural Network, for which a 20% of the training set was used as **validation** set. In fact it is fundamental to see after every epoch how well the NN constructed so far fits the model, comparing the loss curves obtained with the training and validation sets, and then select the best configuration based on the optimal validation error. Despite the fact that Neural Networks are usually able to deal with all kind of features, it was decided to exclude **Time** in order to avoid it to be one hot enconded, using the binary variable **Holiday** instead. All the continuous attributes were normalized in order to avoid scalability problems that could have affected the efficient working of the neural networks.

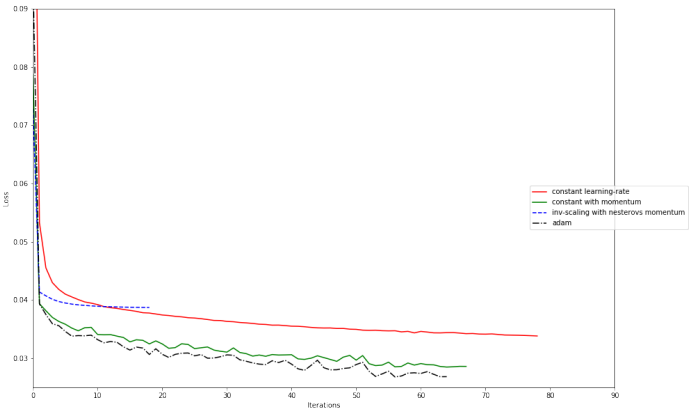


Figure 3.9 Loss curves for MLPC

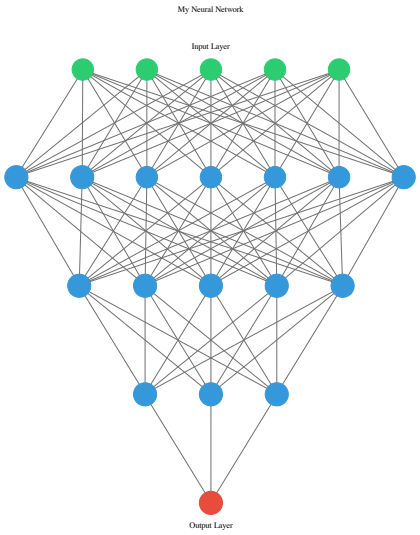


Figure 3.10 DNN structure

After having tried the single perceptron, already described in Section 3.1.4, the approach was to initially compare the loss curves provided by different configuration of **MLPC (Multiple Linear Perceptron Classifier)** before moving to the verification of the built models. The obtained MLPC loss curves are shown in Figure 3.9, from which it is possible to observe that the **Adam** adaptive learning rate optimization algorithm was the most compliant with our research for the lowest loss. Such algorithm is a stochastic gradient-based optimizer that uses the squared gradients to scale the learning rate and than takes advantage of momentum by using moving average of the gradient instead of the gradient itself. It was therefore used to build the first NN, by implementing the MLPC with a single hidden layer composed of 3 nodes, a logistic activation function and an initial learning rate of 0.2. The NN just described performed with a 98.93% accuracy, giving a loss of 0.04 on the training and of 0.048 on the validation set. It was then explored the possibility of building a more complex model: a **Deep Neural Network**. A sort of randomized search between the parameters was made and each combination was evaluated on the validation set, basing such evaluation on loss and accuracy. The goal was to select the number of hidden layers and nodes in such a way that the loss curves of the validation and training sets resulted collinear. After numerous attempts, very good results were obtained with a DNN with three hidden layers of sizes 7, 5 and 3, whose composition is represented graphically in Figure 3.10 to better understand how the information was processed and classified. The five input nodes (in green in the figure) represent **Temperature**, **Humidity**, **Light**, **CO2**, and **Holiday**, while, as already described, there are three hidden layers with different sizes which bring the process to the output node (in red) for **Occupancy**. The selected activation function for the hidden layers was the **rectified linear unit**, while the one for the output node was the **sigmoid**. The applied learning mode was the **minibatch**, updating the weights every 10 records. The performances of such model, evaluated through cross-validation, were the following: Accuracy = 0.9912 (+/- 0.004) and F1-score = 0.9878 (+/- 0.006); it was also possible to read the history of the training process, whose last three rows are reported in Table 3.2. **Early stopping**, **L2 regularization** and **dropout** were also applied, but since the described model did not seem to present an overfitting problem, they were tested on the same DNN but with a larger number of epochs (1000). None of the 3 methods lead to higher accuracy and they were therefore not needed. In fact, in Figure 3.11 is possible to appreciate the loss both on the validation and on the training set in a more accurate way, from which it can be observed that the two lines were almost parallels from a certain epoch onward, which was sufficient to conclude that the model was well fitted.

	loss	accuracy	val_loss	val_accuracy
Epoch 98/100	0.03	0.9910	0.0379	0.9875
Epoch 99/100	0.03	0.9911	0.0385	0.9875
Epoch 100/100	0.03	0.9912	0.0372	0.9878

Table 3.2 Last 3 epochs of the training process

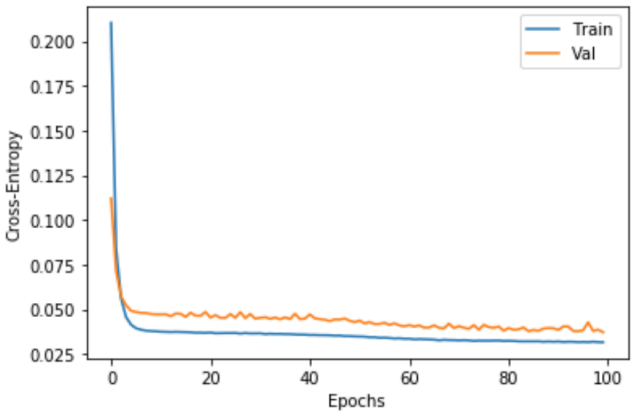


Figure 3.11 Model fitted on 100 epochs

3.2.3 Ensemble methods

The methods described in this paragraph are based on the so-called **wisdom of the crowds**, which says that collective knowledge of a diverse and independent group of people typically exceeds the knowledge of any single individual. Translating this idea to the classification task, it can be supposed that aggregating a multitude of weak classifiers forms a strong classifier. The used **base classifiers** are usually decision stumps, but in this case that was already a strong classifier, as observed in the previous paragraphs. As it could be expected, **Random Forest** gave a similar result to the one of the Decision Trees previously analysed: **Light** was still the main actor in the classification while the other features were less influencing. In spite of having an already very high accuracy with the decision stump, it could still be improved: the RF was in fact able to reach a 99.7% accuracy and the cross-validation score was 99.3%. **Boosting** and **Bagging** were also applied using the Random Forest as base classifier and the result were the same as before, reaching 99.7% of accuracy. A **stacking** technique was also tried, using the logistic regression and then the multilinear perceptron as final estimators. All this experiments were evaluated using cross-validation, where they all provided 99.3% of accuracy. F1 score was near to 100% too in all these cases. In Figure 4.5 is shown the relation between different measures used to evaluate AdaBoost and the number of estimators used. It can be observed that the algorithm SAMME.R (discrete AdaBoost that take into account the probability estimation) converged faster, but it can also be seen that for both AdaBoosts, real and discrete, it would be unnecessary to use more than 200 estimators.

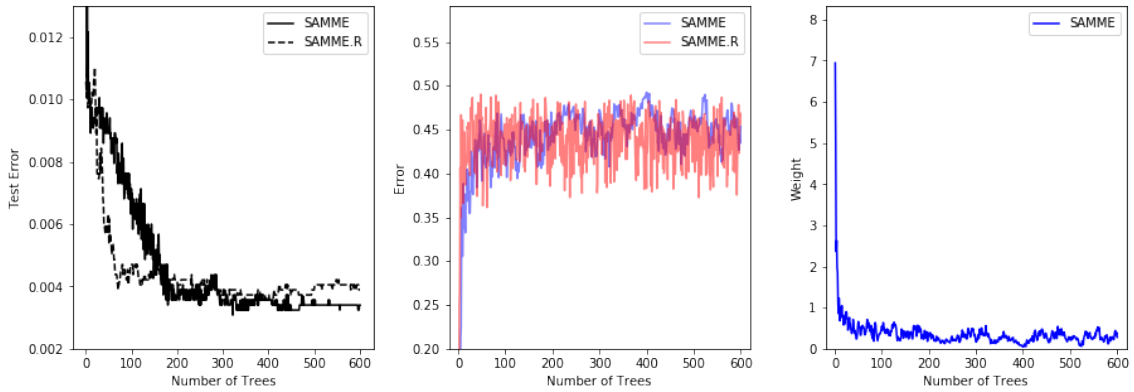
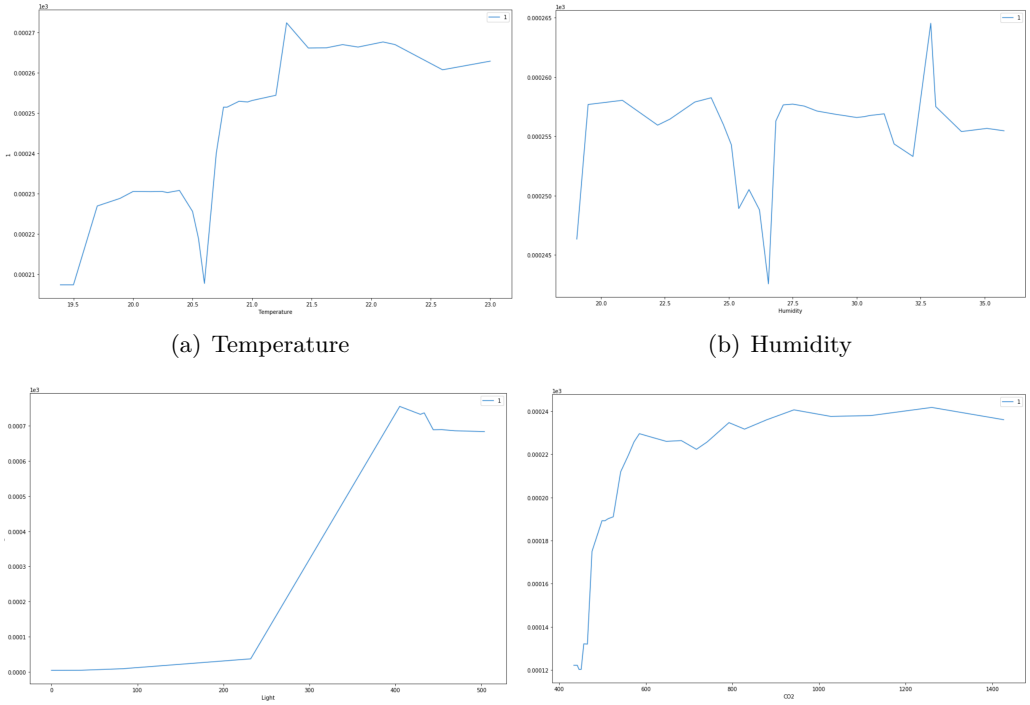


Figure 3.12 Evaluaton of the AdaBoosted Decision Trees

3.3 Explainability

In order to conclude the classification task, explainability was examined, trying to make the classification process more unterstandable and interpretable by humans. First of all a **black box** model was trained, based on the Random Forest mentioned in the previous paragraph. For what concerns global explanation, **partial dependance plots** for the dataset's features are presented in Figure 3.13. A random record was then selected in



(a) Temperature (b) Humidity

(c) Light (d) CO2

Figure 3.13 Partial dependence plots

order to obtain some local explanation: **LIME** and **LORE** methods were applied. They both confirmed what had already been discovered previously in this section, which is the importance of the variable **Light** for classification: in Figure 3.14 are shown the feature importances obtained with Lime Tabular Explainer. LORE explainer was then applied to the same randomly selected record with a neighborhood sample of 1000. The rule resulting from this approach was that ***Light>350.00*** and ***CO2>480.50*** would imply **Occupancy=1**, while the obtained counterfactual rule was ***Light<=290.36***. In fact, by lowering the value of the attribute **Light** in the analysed record from the original 496 to 290, the black box does not predict it as *occupied* anymore and labeled it with a 0 instead.

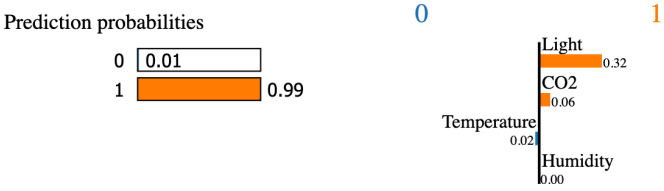


Figure 3.14 Features importances

4 Imbalanced Learning

Another very interesting topic is to learn how to deal with unbalanced datasets, but since the one analysed in this report was not unbalanced enough to consider it a problem (77-23 distribution in the target variable), some transformations needed to be applied first. One feasible solution in order to transform the available dataset into an unbalanced set was removing all the records that could be considered easily classifiable. In fact, it seemed straightforward that during nights and weekends the rooms were always empty and therefore the target variable value for instances recorded in such periods was always 0. It should also be kept in mind what was discovered thanks to the Decision Trees: low luminosity implicates **Occupancy=0**. It was thus established to apply the following transformations both to the training and test sets in order to obtain an unbalanced distribution:

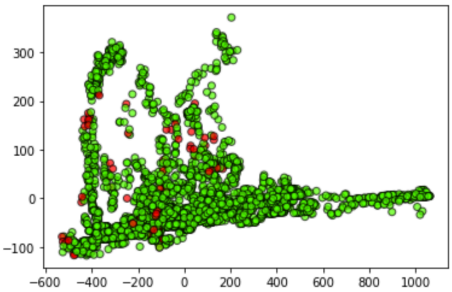


Figure 4.1 Imbalanced training

- removing data recorded on Saturdays and Sundays;
- removing data recorded before 7:35AM or after 6:35PM;
- removing records with a **Light** value lower than 388;

obtaining a dataset of 4900 records with the following distribution for the target variable:

OCCUPANCY	%	DATASET	TRAINING	TEST
1	96.27	4717	3302	1415
0	3.73	183	128	55

At this point it would have been easy to reach 96% accuracy just by classifying all the records as *occupied*, so it would have been very difficult to perceive how the classification would have improved with other classifiers. In order to solve this problem it was necessary to rebalance the datasets using different undersampling and oversampling techniques.

4.1 Oversampling

The first presented technique will be oversampling, which could be preferred over undersampling since the training set was already pretty small and so it would be better to enlarge it instead of shrinking it, to avoid underfitting. The methods used were **random oversampling**, **SMOTE**, **ADASYN** and **SVSMOTE**. Plotting the first and second eigenvectors of the PCA can help appreciating how different techniques affected the distribution of the dataset's principal component; such plots are shown in Figure 4.2.

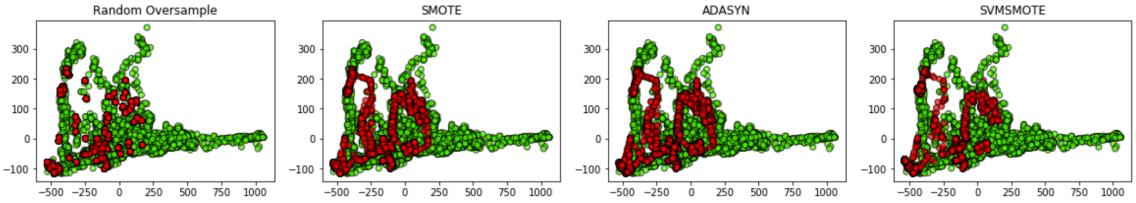


Figure 4.2 Oversamplings' scatter plots

These methods were evaluated by repeating the classification with Decision Tree and Random Forest classifiers. Surprisingly, random oversampling was the technique that performed better, leading to a 96.8% accuracy for DT and 97.28% for RF (up to 97.62% with the stacking technique); the obtained ROC curve is presented in Figure 4.3, from which it is possible to appreciate how little the area under the curve increased. SVSMOTE, SMOTE and ADASYN also gave good results with the RF, reaching respectively an accuracy of 97.55%, 96.87% and 96.67%, while none of them performed as good with the DT. In fact the DT on the newly balanced dataset always performed with an accuracy lower than the 96% threshold imposed by the *'dumb'* classifier applied to the unbalanced dataset.

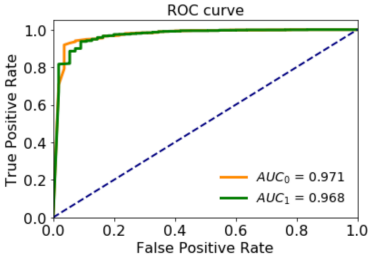


Figure 4.3 Random oversampling

4.2 Undersampling

For the undersampling techniques a different training-test split was applied: 90-10. The implemented methods were **random undersampling**, **Tomek Link**, **Condensed Nearest Neighbour** and **Near Miss**. As for the oversampling approaches, they were evaluated by repeating classification with Decision Tree and Random Forest classifiers. Tomek Link was the one that performed better, reaching a 97.62% accuracy with the RF. CNN also gave good results: 96.94% accuracy with the RF and even a 97.21% when implementing also the stacking technique. From the results obtained it could be asserted that it was still possible to classify records using an advanced classifier like the Random Forest, but a Decision Tree proved to be inefficient in predicting the target variable for the 1470 records in the test set with a training of only 250/300 instances. In fact, with the DT classifier none of the implemented methods allowed to reach an accuracy higher than the 96% threshold of the unbalanced distribution, always being even lower than 85%.

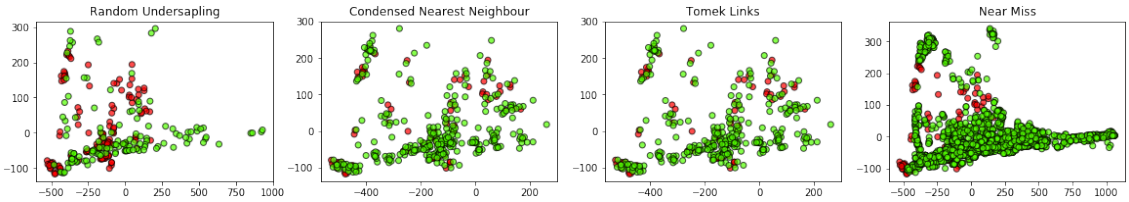
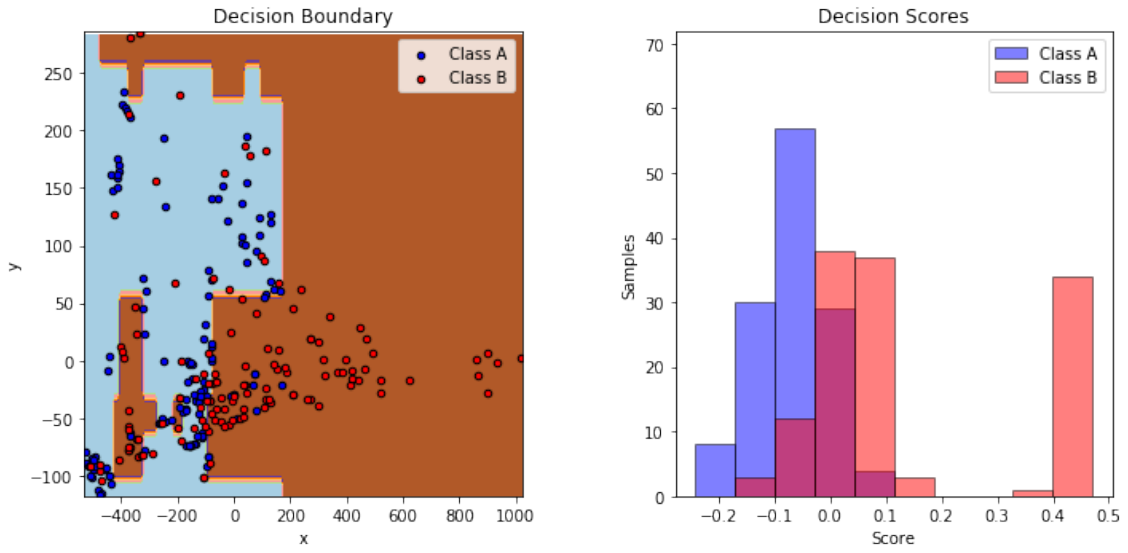


Figure 4.4 Undersamplings' scatter plots

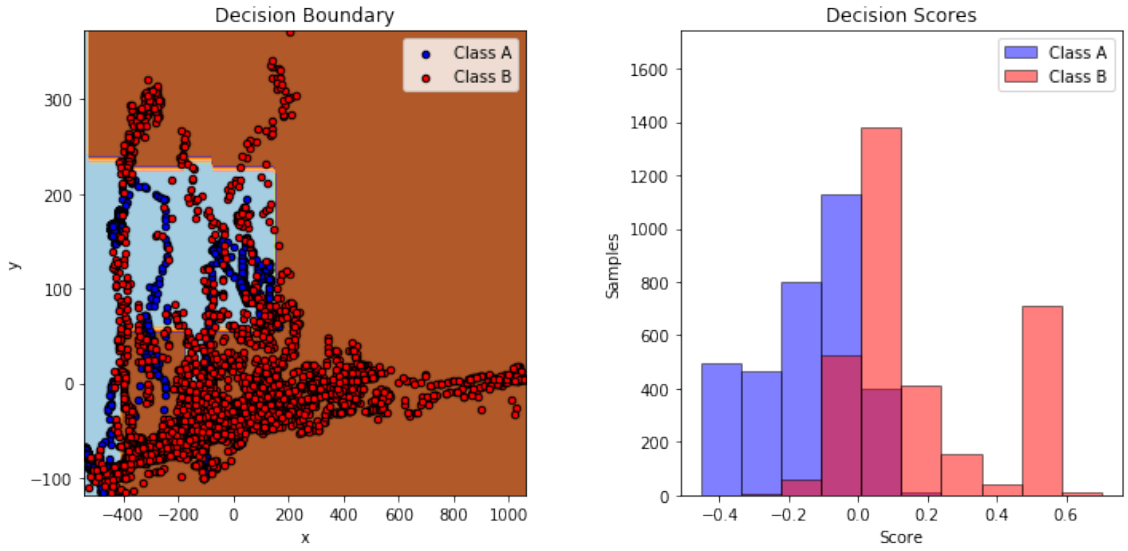
The last explored approach was to evaluate the performances of a RF on the original dataset after adjusting the **Class Weights**, giving 97% of weight to the class 0 and 3% to the class 1. The results of cross-validation are reported in Table 4.1. In Figure 4.5 are also shown the decision maps of an **AdaBoost** algorithm applied with a decision stump as base estimator. The two plotted scatters are from random oversampling and undersampling, so that it is possible to compare their results.

	F1	AUC	AVG PRECISION	ACCURACY	PRECISION	RECALL
Randomized CV	0.676363	0.734749	0.582429	0.974993	0.952381	0.546366
Stratified CV	0.702564	0.752885	0.581961	0.972991	0.87963	0.555556

Table 4.1 Random Forest with Class Weights cross-validation



(a) Random undersampling



(b) Random oversampling

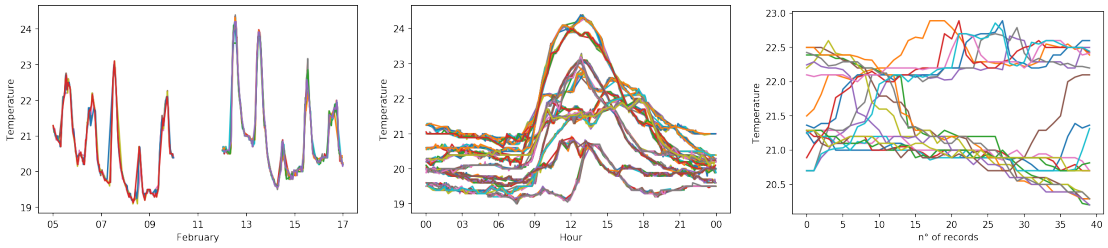
Figure 4.5 AdaBoost decision maps

5 Time Series

Given the presence of a temporal feature, **date**, in the dataset, it was possible to analyse the data as a time series and such analysis will be presented in this section. In order to inspect the data examined so far as a time series, they were manipulated in order to highlight their temporal dimension, using the attribute **date** as the new index of the dataset. The first step was trying to extract two time series datasets from the original ones, *datatraining* and *datatest*. After some exploration, the selected attribute to be extracted was **Temperature** because of its excellent temporal trend, but also other features' time series will be considered in the following paragraphs. Since the data in question represented daily surveys, the datasets were structured so that they would be composed by records from contiguous days. Both the obtained sets contained data referring to one week: the training from 3/2 to 10/2, while the test from 11/2 to 18/2. Data from the 'marginal' days of such periods were deleted, since it was noticed that they were incomplete (there were only records from the evening hours). By doing so, five days were selected to be kept, from Thursday to Monday, ensuring that both time series covered the same amount of time. At this point various time series could be extracted using different criteria, i.e. using three distinct time intervals, determining three new time series datasets:

- **Week**: the extracted time series were composed of the five days span just described, from Thursday to Monday.
- **Day**: the extracted time series were composed of a daily span.
- **Daysplit**: the extracted time series were composed of a daily span which was in turn splitted into three equal parts in order to obtain time series corresponding to different moments of the day: *night*, *morning* and *evening*.

The time series just described are plotted in Figure 5.1.

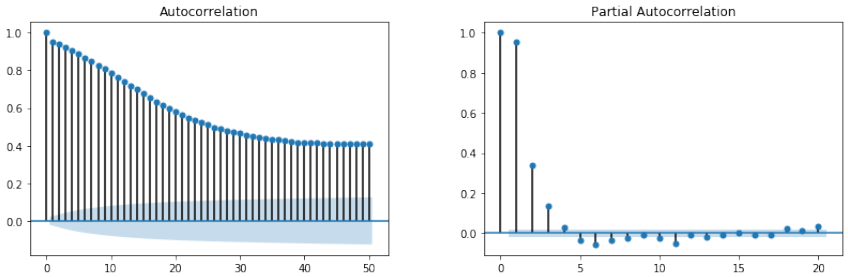


(a) Week (b) Day (c) Daysplit

Figure 5.1 Time series datasets

5.1 Forecasting

To provide a forecasting analysis, the data from *datatest* was used since it was the largest set available. Such records had been collected in the week from Tuesday 11/2 to Tuesday 18/2, but, as already mentioned, the first and last days were eliminated to obtain a five-days span. In Figure 5.2 are shown the **ACF** and **PACF** plots for the extracted series from which it is possible to retrieve some information about the time series in question. The former indicates that there is a strong autocorrelation component since, even if exponentially decaying, the values remain high even for huge lags, whereas the PACF plot shows that the correlation for a time series with a lag k>2 is mainly captured by the already explained variations with smaller lags and for k>4 is totally explained by shorter lags.



(a) ACF (b) PACF

Figure 5.2 Autocorrelation and partial autocorrelation plots

Before proceeding with forecasting, data needed to be normalized and therefore logarithm and rolling mean transformations were applied. This process was necessary to exclude some components that could prevent us from making an accurate prediction: **trend** and **seasonality**. Such components were exposed thanks to a deseasonality process, that permitted to identify the elements that composed the time series, which are plotted separately in Figure 5.3. With the help of the statistical **Dickey-Fuller test** it was proven how stationarity, which is essential for good forecasting, increased after these transformations were applied. The comparison between the results of the test on original time series and on the transformed ones is reported in Table 5.1.

	Dickey Fuller Test		
	ts	ts_log	ts_log_mov_diff
Test Statistic	-2.804057	-2.747723	-7.883220
p-value	0.057697	0.066138	4.642763e-12
#Lags Used	31.000000	31.000000	29.000
Observations Used	8608.000	8608.000	8.581000e+03
Critical Value (1%)	-3.431110	-3.431110	-3.431112
Critical Value (5%)	-2.861876	-2.861876	-2.861877
Critical Value (10%)	-2.566949	-2.566949	-2.566949

Table 5.1 Dickey-Fuller test results

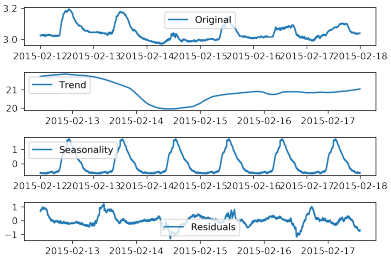
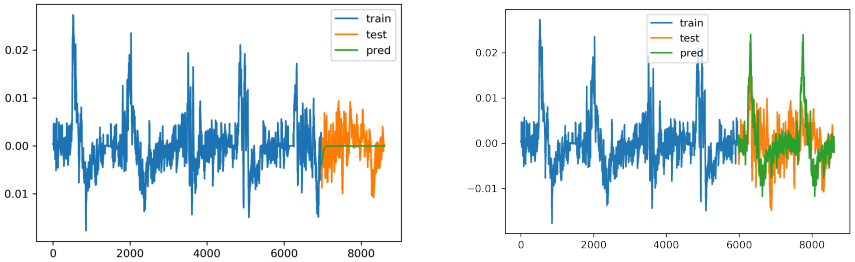


Figure 5.3 TS components

At this point, **Exponential Smoothing**, **ARIMA**, and **SARIMAX** models were applied. By looking at the PACF plot, some configurations for ARIMA were implemented, i.e. (2,1,0) and (4,1,0), but, as shown in Figure 5.4, they did not fit well the time series, probably because of the difficulty in managing the seasonality. Such problem was instead dealt with by applying Exponential Smoothing with the Holt-Winter's method, using additive seasonality and ignoring the trend (hard to capture on short periods). Such method led to very satisfying results, obtaining very low values for error metrics (MAE and RMSE smaller than 0.01, MAPE around 1.5) and reaching an R2 score of 1.



(a) ARIMA (b) Exponential Smoothing

Figure 5.4 Forecasting on *datatest*

In the light of the optimal results obtained with Exponential Smoothing, it was decided to adopt such technique for a challenging goal: given the three original datasets, predict the values of the extracted variable (**Temperature**) in order to fill the gaps that separated them, creating a unique time series that would cover the whole two weeks span. Predicting values between *datatraining* and *datatest2* was quite easy: the algorithm was trained on the former, which was sufficiently longer than the gap to be filled and therefore fitted it well, as can be seen in Figure 5.5. On the other hand, the latter was the shortest set of the three available and thus its prediction did not seem to connect it well to the training. For this reason it was also used *datatraining* to predict 'backward', which was also not accurate, but it opened to the possibility of interpolating those two predictions, obtaining pretty satisfying results, as shown in Figure 5.6.

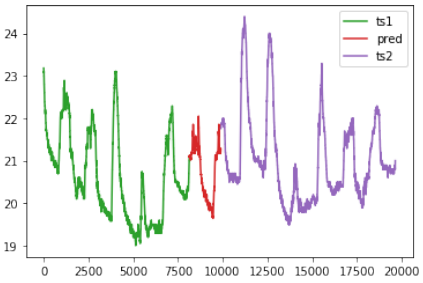
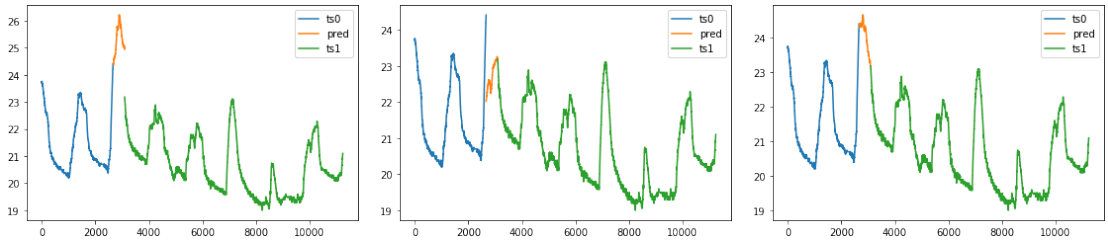


Figure 5.5 Gap between *datatraining* and *datatest2* filled



(a) Forward prediction (b) Backward prediction (c) Interpolation

Figure 5.6 Gap between *datatraining* and *datatest* filled

Once caluclated the values for filling the gaps with the described method, the five time series (three original datasets and two forecastings) were merged together, forming a unique one, as shown in Figure 5.7. Such time series was composed of 22741 records, which were then pruned to 21600 so that it would cover a period of exactly fifteen days (from 00:00 of the 3/2 to 23:59 of the 18/2). The complete time series was then used for further analysis such as motifs discovery and sequential pattern mining.

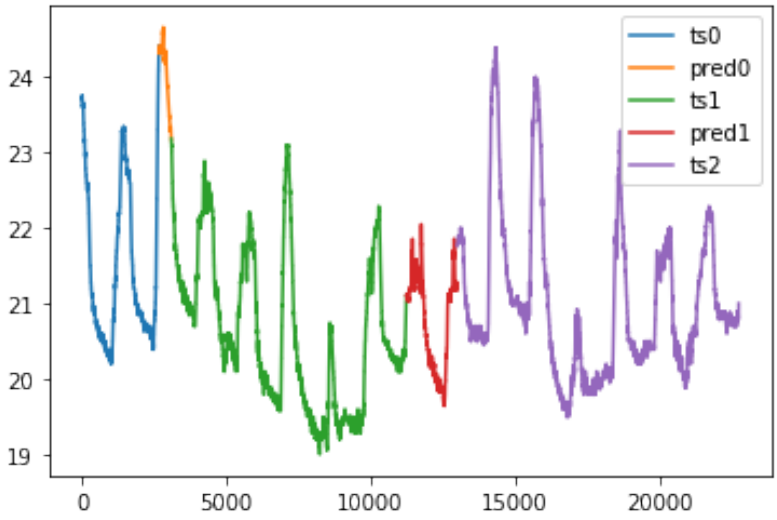
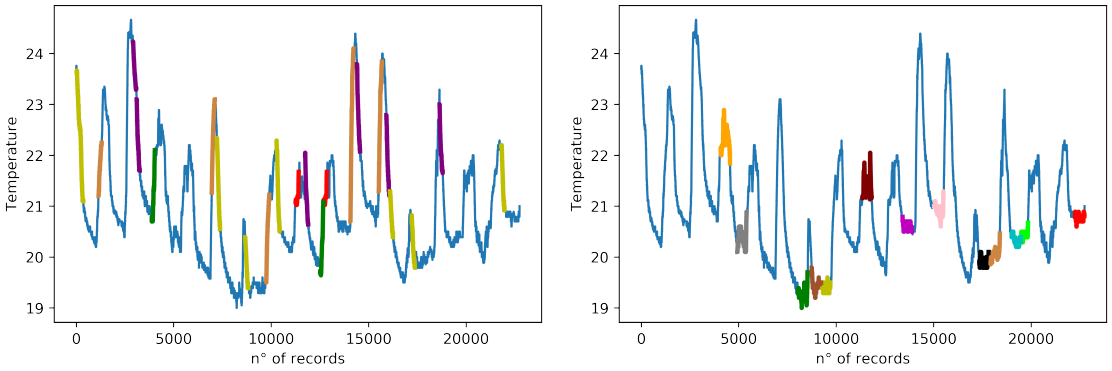


Figure 5.7 Temperature time series with filled gaps

5.2 Motifs and anomalies

Once obtained the complete time series, an interesting task was to discover motifs, which are repeated subsequences in the analysed time series. These patterns appeared to be mainly depending on the **Temperature** negative slope in the data, specifically where sudden changes took place, which represented the daily thermal excursion. When detecting anomalies, it was made sure that they were distinct, imposing at least a few hours to pass between them. Furthermore, for the purpose of finding such subsequences it was required to use a larger window (480) on the **matrix profile** than the one previously used for motifs (160). After a specific research a small number of anomalies was detected, which emerged mostly in the morning in correspondence of multiple peaks. It could be assumed that such peaks were due to the entrance of people into the detected rooms, while the drops took place when they leave the office in the late hours of the afternoon.



(a) Motifs

(b) Anomalies

Figure 5.8 Motifs and anomalies in the Temperature time series

5.3 Clustering

For the clustering task, the datasets used were the ones mentioned at the beginning of this section: **Week**, **Day** and **Daysplit**. For each one of them, one or more clustering approaches were tested and the results will be presented in the following paragraphs. Please note that for this task were used not only the series derived from the extraction of the attribute **Temperature**, but also the ones representing other features.

5.3.1 Shape-based

The shape-based approach was applied to the **Daysplit** dataset, which was selected for this kind of algorithm because of its small size. The time series in such set were pre-processed through an appropriate scaling and then different algorithms were tried, such as **timeseries kmeans** and **k-shapes**. Both algorithms were implemented with **k=3**, since the idea was to identify the three different moments of the day, *night*, *morning* and *evening*. Neither of them permitted to obtain well-separated clusters, as can be observed in Figure 5.9. However, the results of timeseries kmeans on the **Light** attribute using **dynamic time warping** as metric are reported. It is in fact the combination that returned the most interesting division: significant correspondence was found between the obtained clusters and the ones that could be defined by relying on the day natural subdivision into *night*, *morning* and *evening*. When using the euclidean distance instead, the results were not as good, as can be observed in Table 5.2.

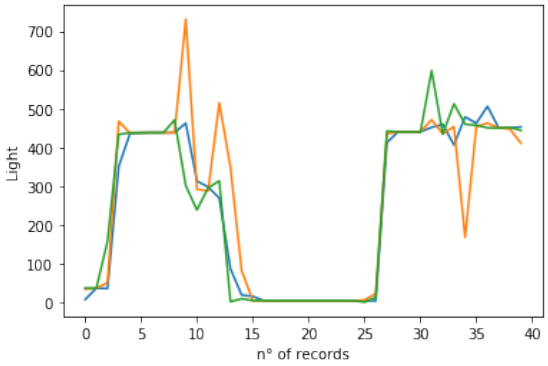


Figure 5.9 Time series K-means

		Class		
		0	**1**	**2**
Day part	evening	21	83	16
	morning	21	18	81
	night	102	17	1

Table 5.2 Cross table

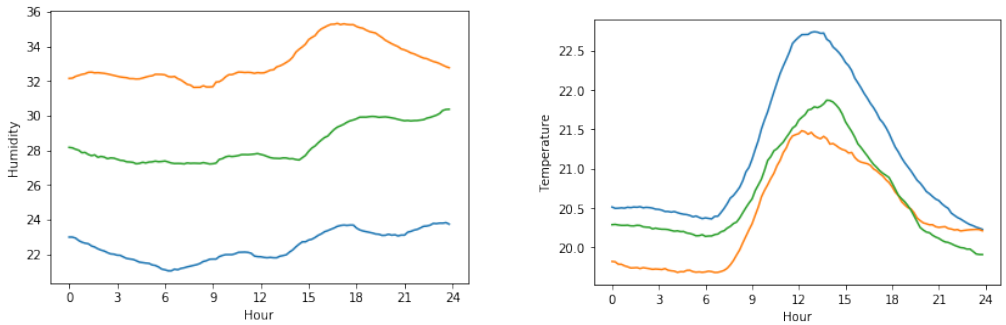
5.3.2 Features-based

Moving to the **Day** dataset, it was preferred to search for clusters using a **features-based approach**, since this series were longer than the previous ones and a shape-based approach would have been less effective. Moreover a time span long as a day lends itself well to be expressed in terms of global features, such as the minimal and maximal temperatures (similarly to what is done in weather forecasting). The main idea was to extract some global features for each time series, thus creating a dataframe suitable for applying **k-means** clustering.

	maxT	**minT**	**avgT**	**stdT**	**maxH**	**minH**	**avgH**	**stdH**
0	22.8900	20.290000	21.507264	0.697488	28.39	19.290	24.119451	2.375155
1	22.8900	20.290000	21.542931	0.719533	28.50	19.245	24.345210	2.556419
...	...	...	...	...	...	...	...	...
118	22.0000	20.133333	20.875812	0.607890	30.60	24.290	28.034342	1.893668
119	22.0000	20.150000	20.930544	0.633451	30.60	24.290	28.256846	1.942604

Table 5.3 Extracted global features

The extracted variables were **Temperature** and **Humidity** in order to identify clusters less dependent from the human activity, which should affect more other attributes (i.e. **CO2** production increase incredibly with human presence). Increasing the number of clusters k, the cluster composition worsened, as the formed clusters were not well-separated, but for k=3, as shown in Figure 5.10, the results were quite satisfying. It can be noticed how the gaps between the centroids were greater for **Humidity** than for **Temperature**.



(a) Humidity

(b) Temperature

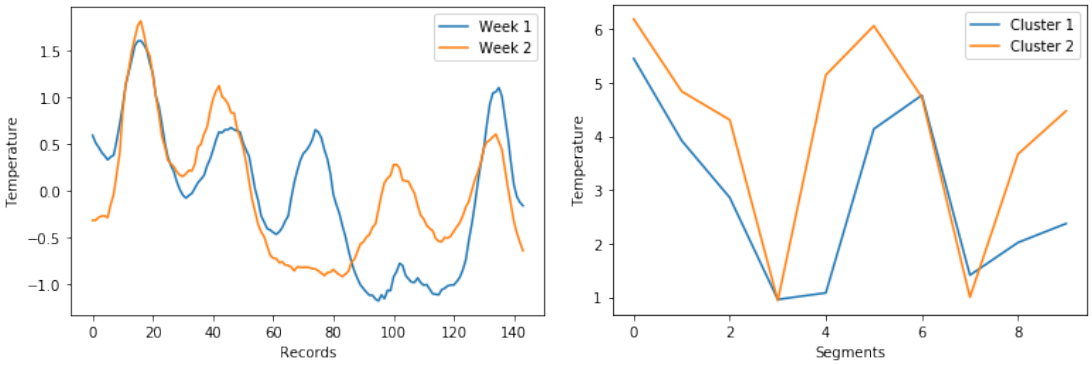
Figure 5.10 Features-based clusters' centers

5.3.3 Approximation-based

The optimal approach for the remaining dataset **Week** was considered the one based on approximation, since the extracted time series was the longest used so far for clustering. The first step was preprocessing the time series with an **amplitude scaling**, afterwards a **SAX (Symbolic Aggregate Approximation)** was performed on the scaled data and then clusters were developed using **Time Series Kmeans** with k=2, trying to reconduct the obtained clusters to the week to which the time series belonged. Table 5.4 reveals how the records were perfectly divided into the two classes, without making any error. This result should therefore be kept in mind for a possible classification. In Figure 5.11 are represented the two weekly time series (a) and the two derived clusters (b).

	Weeks	
	1	**2**
1	49	1
2	0	50

Table 5.4 Cross table



(a) SAX Centers

(b) SAX Labels

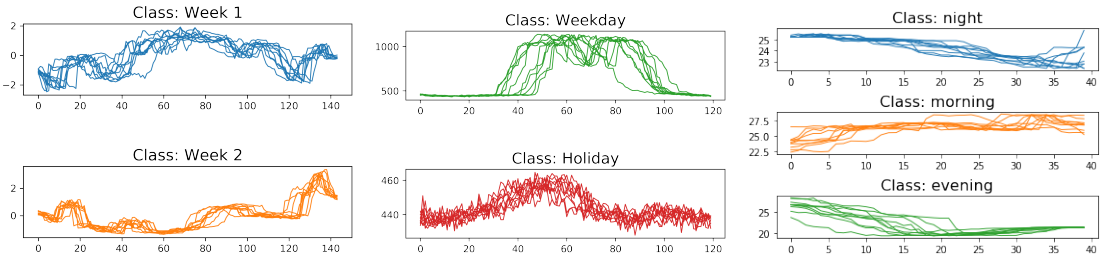
Figure 5.11 Clusters vs original weeks

5.4 Classification

For this task the same three datasets employed in the previous paragraphs were used. For each one of them, the classes in which the records should be classified were defined differently, trying to exploit the various information present in them:

- in the **Week** set it was tried to predict whether a time series had been extracted from the *first week* or from the *second week*. The utilized time series was extracted from the feature **Humidity**.
- In the **Day** set, classes could have been a specific date or the day of the week, but a simpler approach, suitable for our data, was to try to determine if a time series represented a *workday* or a *holiday*. Having days from two different weeks could have been misleading, but the attribute selected for this classification, **CO2**, seemed to have really distinct behaviours in the two identified classes since, as already observed, it is highly influenced by the presence of people, who are absent during holidays.
- In the **Daysplit** set, once selected a day-splitting criteria, the idea was to predict to which part of the day the analysed record belonged. In order to make the task more challenging, it was decided to use at least three classes, i.e. *night*, *morning* and *evening*. In this case the extracted feature was **Temperature**.

In Figure 5.12 are shown the plot of the various classes identified with the different approaches just listed.

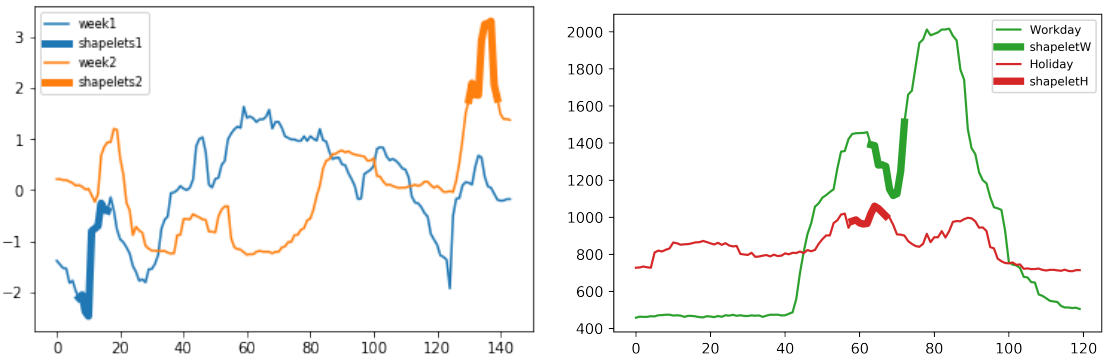


(a) Week in Humidity (b) Day in CO2 (c) Daysplit in Temperature

Figure 5.12 Defined classes

5.4.1 Shape-based

Another step that ought to have been taken before starting with the classification was the **shapelets** extraction, which are subsequences that are extremely representative of class membership for each identified class. It was not possible to find such subsequences in the the day splits, because the series were too short, while both for *week1/week2* and for *workday/holiday*, longer and with more characteristic shapes, shapelets were found. The most representative ones are shown in Figure 5.13 and they could be used for a shapelet-based classification, which lead to a very satisfying 100% accuracy for both datasets.



(a) *Week1/ Week2* (b) *Workday/Holiday*

Figure 5.13 Shapelets

5.4.2 Structural-based and distance-based

As already mentioned in Section 5.3.3, in the **Week** dataset two groups corresponding to the first and the second week could be easily identified by clustering. Such approach allowed to classify each time series without errors with any classifier (i.e. KNN or Decision Tree) both in feature and distance based approaches, resulting in a 100% accuracy. The **Day** dataset also reached 100% accuracy in classifying each day either as a weekday or holiday both with Decision tree and KNN. Classification of **Daysplit** records into night, evening and morning was not as perfect, but still gave good results. The KNN algorithm proved to be the best approach in this case, especially if used with **Dynamic Time Warping** and the **Sakeochiba** constraints, which allowed to reach a 94% accuracy. It can be affirmed that the most suitable methods would be the ones based on distances, in particular KNN has been elected as the best one since, in contrast to other classifiers, it was able, with the necessary precautions, to classify also the **Daysplit** dataset excellently.

5.4.3 Deep Neural Networks

More advanced machine learning techniques such as **convolutional** and **recurrent** neural networks were also implemented. Such approaches were explored even if they seemed unnecessarily complicated since 100% accuracy had already been reached with simpler methods. It was not easy to build a model to classify properly **Daysplit**, while 100% accuracy was reached again for the other two datasets. The accuracy reached using **LSMT (long short-term memory)** was disappointigly lower on the validation set, which was a clear signal of overfitting and was the reason why the CNN was preferred.

5.4.4 Multivariate time series

Since there were many influencing attributes when classifying **Daysplit**, a ***multivariate*** time series approach was explored, using all the recorded features. LSTM and CNN models were built, trying to exploit all the records' characteristics to classify them adequately into the parts of the day. Unfortunately neither of them was helpful because they both classified all the records as belonging to the same class; the resulting accuracy was therefore equal to the frequency of a class (0.33), which did not add any further information. A feature-based approach was hence implemented too, since it had already performed well on a single attribute. For each of the selected features, statistical measures such as mean, standard deviation, median, variance, interquartile range and skewness were calculated. These measures, taken for every attribute, formed a new dataset containing 24 features. While KNN in this scenario proved inefficient, Decision Tree allowed to reach the prefixed goal, predicting which part of the day a record came from with an accuracy of 99%. As can be seen in Figure 5.14, after the first split the afternoon class could already be identified. Similarly to what was shown in Section 3.1.3, a decision stump identifies a class by the value of **Light**, this time using its median, which again showed how much the occupation of the room and the moment of the day are correlated. Obviously this time the Decision Tree needed to be deeper because there were three classes and after a few more splits a good classification was reached. It can be noticed that only some of the attributes created took part in the decision-making process, while the others were redundant: in Table 5.5 are reported the importances of the most relevant attributes for this classification.

Feature	Importance
medL	0.500000
stdT	0.268277
varH	0.069177
avgH	0.042438
iqrL	0.030380
skwH	0.026244
skwT	0.023469
iqrC	0.017857
iqrT	0.010714
medH	0.008929
iqrH	0.002515

Table 5.5 Scores

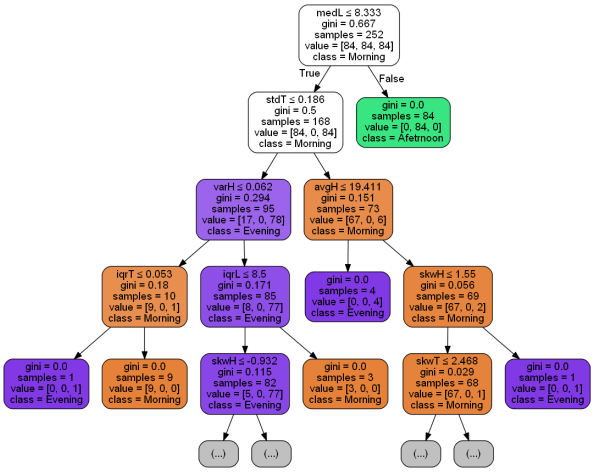


Figure 5.14 Structural-based Decision Tree for multivariate time series classification

6 Sequential pattern mining

For this final task, the data considered was the time series obtained at the end of Section 5.1. In fact, since the original data did not present any sequences, the idea was to exploit the time series and transform it adequately. **SAX** was applied to reduce the size of the series, dividing it into 720 segments of equal length, whose values were then replaced with the mean of the segment they belonged to; finally they were divided into 15 equiprobable regions and renamed according to a specified alphabet. Once completed this process, the search for recurrent patterns started: the idea was to consider each day as a sequence of transactions and apply the ***Prefix Span*** algorithm to find them. Dividing the time series into 15 equal parts (one for each day), the database of sequences of transactions was obtained, which are shown in Figure 6.1. Each of them was compose by 48 transactions, each one made by a *singleton*. By applying Prefix Span, the search was made between closed frequent pattern with various values for support, which are presented in Table 6.1 together with the number of patterns with a specified minimum length for each of those supports.

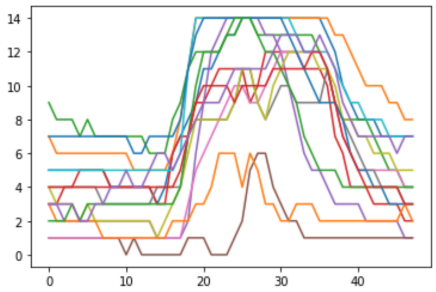
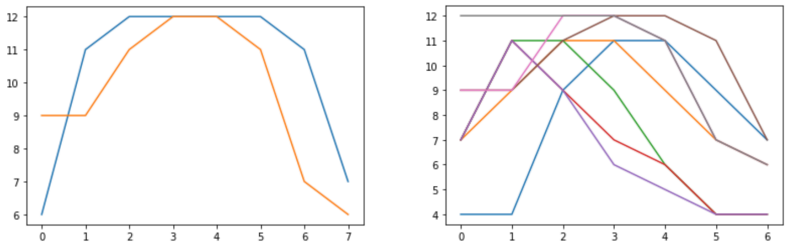


Figure 6.1 Daily sequences

MinSup\MinLen	>=1	>=2	>=3	>=4
3	2252	2241	2174	2001
4	1418	1407	1340	1170
5	816	805	739	582
6	451	441	378	250
7	235	225	167	77
8	118	108	60	16
9	64	54	20	3
10	33	23	3	0
11	18	9	1	0

Table 6.1 N° of frequent patterns

Considering a minimum support equal to 6, the longest patterns found were two of length 7, as shown in Figure 6.2a, and other 7 with length 6 (Figure 6.2b). In both cases the represented patterns seemed to follow a little increase in the values reaching a local max and than a decreasing zone. Clearly this observations are not the real goal of sequential pattern mining, which assumes that items do not have an order, but in this case it seemed natural to reason like this since the analysed data was derived from a time series. By constructing similar plots for other supports it was noticed that the vast majority of the segments were in the [4,12] class interval, so values in the 3 lowest and 2 highest regions were not frequent. The three most interesting patterns were the ones with at least 4 elements and a minimum support of 9, which were the following: [6, 11, 11, 9], [7, 11, 11, 9] and [11, 11, 9, 7]. It was curious to observe how they all included a double 11 followed by a 9, which is in fact the only frequent pattern of length 3 with a minimum support of 11. Unfortunately the datset was not meant to apply sequential pattern mining and therefore the obtained results can be visually observed, but they are hard to interpret. An interesting topic could be to explore the possible correlation between the individuated frequent patterns and the motifs extracted in Section 5.2.



(a) MinLen=7 (b) MinLen=6

Figure 6.2 Frequent Patterns with MinSup=6

7 Conclusion

The provided dataset presented only numerical features apart from the target variable **Occupancy**. Only one of them, **HumidityRatio**, was considered not helpful for the conducted analysis, while all the other were somehow exploited. There were not many outliers in the data, as already mentioned. The individuated regression problems gave quite interesting results, even though the data appeared to be too sparse to be synthetized either with single or multiple linear regressions. Overall the most satisfying results were obtained in the classification task, where lots of different classifiers performed almost perfectly, even though this suggested that such results were obtained because of the easiness to classifiy the records and not the actual adequacy of a method rather than another. In Table 7.1 is presented a summary of the classification methods performed already explored through the report. The mentioned easiness in classifying might indicate that there were some peculiar characteristics that allowed to distinguish very clearly whether the room was occupied by someone or not. The task concerning unbalanced learning was performed even though the dataset did not present an unbalanced structure and thus needed an appropriate transformation first. This lead to results that were actually not very interesting, even if the applied methods worked in most of the cases. The second most challenging task has been the time series analysis. In fact, by exploiting the temporal dimension of the data lots of analysis were possible: the forecasting not only performed well, but also allowed to fill the gaps between the original datasets, creating a unique time series of the overall period. Furthermore, different temporal splits were explored, all performing pretty badly in the clustering task, but quite well in the classification one. The same observation made for the unbalanced learning is valid for sequential pattern mining: the application of the Prefix Span algorithm in fact brought to the results exposed, but they were not very useful and hard to be interpreted, since the sequences and transactions were not actual sequences and transactions, but were derived from the time series with the described transformation.

Classifier	Accuracy
KNN	99.01%
Naïve Bayes	97.5%
Decision Tree	99%
Logistic regression	99%
SVM	98.98%
Neural Network	99.12%
Random Forest	99.7%

Table 7.1 Classifiers summary