# Progetto DM2

Gianmarco Pepi, Monia Bennici

June 2020

## Contents

# 1 Introduction

This text is based on the study of the dataset "Occupancy Detection", available in UCI (machine learning repository) site, composed by three dataset ("datatest.txt", "datatest2.txt" and "datatraining.txt"), using the sci-kit learn library, pandas, Numpy and Keras. The project objective is that of apply the main tool we know, in order to extract information from the dataset, to classify new records, make predictions based on time and detect the pattern rules to understand the occupational situation of a room. To do so, we have to adapt the dataset to the task, sometimes modifying it or creating a new one from the original. The study could have multiple application, like to avoid waste of energy or to get better some environmental condition. The work starts with the analysis of the data, passing through the main classification techniques we know. Also, a time series analysis was made. The code we implemented to perform the study is available in our GitHub channel (https://github.com/DM2-project/Occupancy-detection).

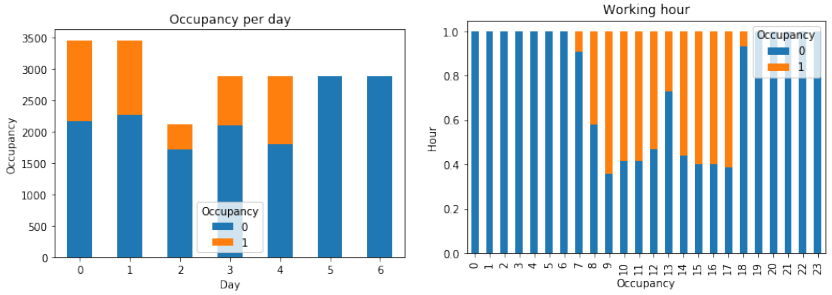
# 2 Data Understanding

The dataset offers information about the occupancy of a room, given some parameters referring to the "Temperature" in Celsius, "Humidity" in absolute terms, the "Humidity Ratio", the level of "CO2", the "date" in which the observation were made (format year, month, day, hour, minute, second) and of curse, the "Occupancy". The complete original dataset is composed by 20560 rows and 7 attributes. In particular, all the attributes are integers or floats, except for the "date" that is an object. "Occupancy" is a binary value that report 0 if the room was not busy, 1 if it was. The date goes from the 2ND of February 2015 to 18Th. We didn't detect any missing value.
The first thing we made, was to transform the attribute date into a datatime format, creating the new "Date". To have a general view on data, we split the attribute "Date" in Day, Month, Year and hour and we saw the main statistic values, noticing that all the observation were made on February 2015, so for month and year the variance is null. The values are shown in the table in Figure 1.

Analysing the dataset, it can be seen that the room is occupied from 7 a.m. until 18 p.m., that can make us think we are talking about an office. Moreover, we noticed that there are some days in which the room is never occupied, no matter the time. These days correspond to the weekend. That's why we decided to consider the days based on the week and not on the month, so we transformed the format giving a value to days: starting from Monday which correspond to 0, until Sunday that correspond to 6. So created two new attributes namely "Hour" and "Day", deleting the old "Date" (Figure 2).
Then we deleted the attribute "Humidity Ratio" since it is redundant for the information about Humidity. Below the final dataset (Figure 3).

| | Temperature | Humidity | Light | CO2 | HumidityRatio | Occupancy | Day | Hour | Month | Year |
|---|---|---|---|---|---|---|---|---|---|---|
| **count** | 20560.000000 | 20560.000000 | 20560.000000 | 20560.000000 | 20560.000000 | 20560.000000 | 20560.000000 | 20560.000000 | 20560.0 | 20560.0 |
| **mean** | 20.906212 | 27.655925 | 130.756622 | 690.553276 | 0.004228 | 0.231031 | 10.073249 | 11.446887 | 2.0 | 2015.0 |
| **std** | 1.055315 | 4.982154 | 210.430875 | 311.201281 | 0.000768 | 0.421503 | 4.729430 | 7.075840 | 0.0 | 0.0 |
| **min** | 19.000000 | 16.745000 | 0.000000 | 412.750000 | 0.002674 | 0.000000 | 2.000000 | 0.000000 | 2.0 | 2015.0 |
| **25%** | 20.200000 | 24.500000 | 0.000000 | 460.000000 | 0.003719 | 0.000000 | 6.000000 | 5.000000 | 2.0 | 2015.0 |
| **50%** | 20.700000 | 27.290000 | 0.000000 | 565.416667 | 0.004292 | 0.000000 | 10.000000 | 11.000000 | 2.0 | 2015.0 |
| **75%** | 21.525000 | 31.290000 | 301.000000 | 804.666667 | 0.004832 | 0.000000 | 14.000000 | 18.000000 | 2.0 | 2015.0 |
| **max** | 24.408333 | 39.500000 | 1697.250000 | 2076.500000 | 0.006476 | 1.000000 | 18.000000 | 23.000000 | 2.0 | 2015.0 |

Figure 1: Statistic values




(a) *Working day* (b) *Working hours*

Figure 2: Barplots showing when the room is empty

| | Temperature | Humidity | Light | CO2 | Day | Hour | Occupancy |
|---|---|---|---|---|---|---|---|
| **0** | 23.18 | 27.2720 | 426.0 | 721.25 | 2 | 17 | 1 |
| **1** | 23.15 | 27.2675 | 429.5 | 714.00 | 2 | 17 | 1 |
| **2** | 23.15 | 27.2450 | 426.0 | 713.50 | 2 | 17 | 1 |
| **3** | 23.15 | 27.2000 | 426.0 | 708.25 | 2 | 17 | 1 |
| **4** | 23.10 | 27.2000 | 426.0 | 704.50 | 2 | 17 | 1 |

Figure 3: Final dataset

# 3 Basic classifiers

To solve the classification task, we used all the data we had, so we merge all the datatset and then split it random in training set(70%) and test set(30%), where y_train and y_test represent the class variable ("Occupancy"), while X_test and

X_train represent the other attributes. We implemented and evaluated some basic classifiers. The performance of all them is good. Below are reported, after a brief explanation, the weighted average of precision, the recall and F1-score.

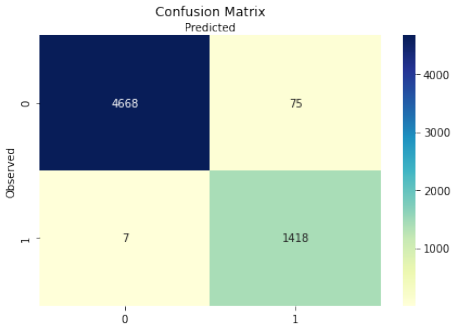
- We implemented the K-Nearest Neighbour trying between different number of k and finding the best with the grid search and according to the cross validation method: we used K=10,disagreeing the general rule for which $K = \sqrt{N}$, obtaining really accurate results. An other parameter that Gridsearch helped us to find is the weights function, with value "distance", that is the closer neighbour have more influence than the far ones. To calculate the distance between point we used the default metric, that is minkowski, but since the power parameter p is 2, it is equivalent to euclidean distance;

- We obtained a lower result, but still good, with Naive Bayes, for which we just used the Gaussian approach, since the data set doesn't present any categorical feature; it was not considered necessary to change the default parameters.

- We implemented the logistic regression, being the class "Occupancy" binary. Using the grid search method, the parameters are the same as the default parameters, that is C=1, penalty=l2 and randomstate=0; The confusion matrix (in Figure a) shows the consistency of the classifier.

- For decision tree classifier we found the parameters through the grid search method. The parameters for which the accuracy was the highest in training datatset were depth=9, minimum sample split=2 and random state=0. The results are confirmed by Cross Validation, as shown in Figure 4. The most important feature in this case is Light, infact if it is less than 375 the Occupancy value is 0.

Below are shown the results of the classifiers:




(a) *Confusion Matrix*

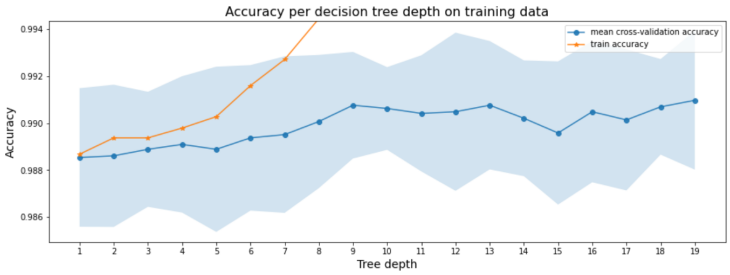



Figure 4

| Classifier | Accuracy | Precision | Recall | F1-score | AUC |
|---|---|---|---|---|---|
| K-NN (K=10) | 0.99 | 0.99 | 0.99 | 0.99 | 0.99 |
| NAIVE-BAYES | 0.97 | 0.97 | 0.97 | 0.97 | 0.98 |
| LOGISTIC REGRESSION | 0.99 | 0.99 | 0.99 | 0.99 | 0.99 |
| DECISION TREE | 0.99 | 0.99 | 0.99 | 0.99 | 0.99 |

The plots of Roc-curve and Lift chart of every classifier is available in the Github channel.

### 3.1 Dimensionality reduction

Trying to obtain the set of principal variables, we used the two existing approaches: feature selection and feature projection. For this section we used all the 11 original attributes, preserving the change in "Date".
With Variance Threshold method, we noticed a little improvement in classifiers' results selecting a threshold equal to 1.2, since the accuracy is already really high for all of them. In this way, beyond Day and Month (which have variance=0) also Humidity Ratio and Temperature are deleted. This confirmed our theory that Humidity Ratio is a redundant value. The Naive Bayes' accuracy improves from 0,97 to 0.98. All the others classifiers don't change their results.
On the other hand, based on Univariate Feature selection, all the attributes are considered important for the analysis, in fact no attribute is removed.
On the contrary, the Recursive Feature Elimination leaves just one attribute, selecting features by recursively considering smaller and smaller sets of features. Nevertheless, computing the accuracy of the previous classifiers, the accuracy doesn't change. Applying the Principal Component Analysis, we find out that the 90% of the variance is explained by two new attributes (Figure 6 and 7). Solving the classification task in two dimensions, only the in Naive Bayes can be seen a fluctuation of accuracy, that become 0,95 instead of 0.97. All the other classifiers rest the same.
The results are showed in the table of the previous section.

```
array([7.76179191e-01, 2.23291081e-01, 3.50991914e-04, 1.52499396e-04,
       2.29074871e-05, 3.32930424e-06, 0.00000000e+00, 0.00000000e+00])
```

Figure 5: Matrix Variance

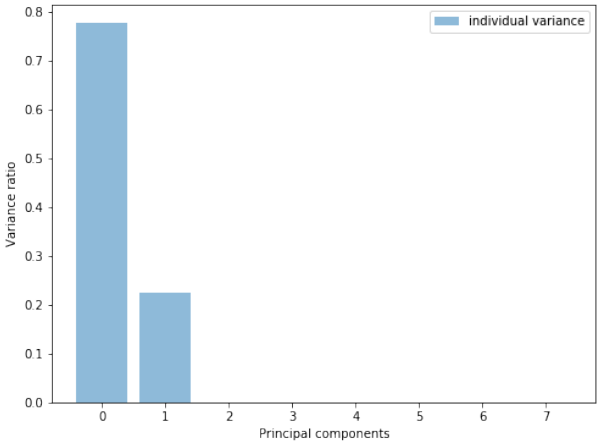



Figure 6

## 3.2 Imbalanced data

Analysing the data set, it can be seen that it is imbalanced: upon 20560 rows, 77% of values represent the class "0" and just 23% represent the value "1". This must make us wonder if the accuracy of the classifiers is actually good.
Nevertheless the imbalancing condition, our estimators are consistent, because in all cases, the accuracy is higher than the trivial classifier's accuracy, that should be at least 0.77.
To test our classifiers, we further imbalanced our data set, leaving just 1% of the values for "Occupancy= 1". We reported the results of classifiers, tested on imbalanced dataset and applying the algorithms for imbalanced dataset. Below are shown the Accuracy (Table 2) and the AUC values (Table 3).
As first try we adjusted the decision threshold and in this case the only improvement is on the accuracy of Naive Bayes, that increase of 0,9%. The random undersampling algorithm make the accuracy of all the classifiers decrease but there is a net improvement of decision tree AUC, that improve of 3.5%. Condensed Nearest Neighbour is not effective for this Logistic regression, since its accuracy decrease of 25,6% and its AUC decrease of 15.2%. The random oversampling algorithm modify the Naive Bayes' accuracy of -1.5%. SMOTE leaves almost unchanged the performance of the classifiers. Finally we implemented the Class Weight algorithm just for Logistic Regression and Decision Tree, since K-NN and NAIVE BAYES' properties doesn't let us to use it on them. We didn't

detect any important change.

| Algorithm | K-NN (K=10) | NAIVE BAYES | LOG. REGR. | DECISION TREE |
|---|---|---|---|---|
| Imbalanced dataset | 0.993 | 0.982 | 0.993 | 0.997 |
| Adjusted Decision Threshold | 0.992 | 0.991 | 0.992 | 0.997 |
| Random Undersampling | 0.967 | 0.954 | 0.984 | 0.985 |
| Condensed Nearest Neighbour | 0.964 | 0.990 | 0.737 | 0.927 |
| Random Oversampling | 0.994 | 0.967 | 0.990 | 0.994 |
| SMOTE | 0.993 | 0.974 | 0.989 | 0.994 |
| Weighted Class | - | - | 0.990 | 0.996 |

Table 2: Accuracy after applying techniques for imbalanced dataset

| Algorithm | K-NN (K=10) | NAIVE BAYES | LOG. REGR. | DECISION TREE |
|---|---|---|---|---|
| Imbalanced dataset | 0.997 | 0.996 | 0.997 | 0.957 |
| Adjusted Decision Threshold | 0.987 | 0.996 | 0.997 | 0.957 |
| Random Undersampling | 0.991 | 0.996 | 0.995 | 0.992 |
| Condensed Nearest Neighbour | 0.979 | 0.989 | 0.845 | 0.929 |
| Random Oversampling | 0.988 | 0.996 | 0.996 | 0.957 |
| SMOTE | 0.987 | 0.996 | 0.996 | 0.988 |
| Weighted Class | - | - | 0.996 | 0.936 |

Table 3: AUC after applying techniques for imbalanced dataset

### 3.3 Multiple Linear Regression

We selected two continuous attributes to apply the Linear Regression. In particular we selected the attributes having the highest correlation "Light" and "Temperature". We evaluated the linear regression performance, looking at the Errors values: we have the coefficient of determination R2 that has a value of 0,456 that means it explains the values quite good, since this value can reach a maximum of 1; the Mean Squared Error(MSE) is quite high, 24634.252; the Mean Absolute Error(MAE) has value 124.202. This result was predictable, given the distribution of the data (Figure 7) We implemented the Linear regression also selecting different attributes (e.g. Co2 and Light), but we reached worst results (lower R2, higher MSE and MAE).
We tried to fit the model using regularization methods such as Lasso and Ridge Regression. In both cases the results didn't change.

### 3.4 Discussion

Looking at the results, the classifiers made a great job. The performances didn't suffer big changes despite the alteration of the datatset. About this we noticed that the classifier who resented of the changes more is the Naive Bayes, that

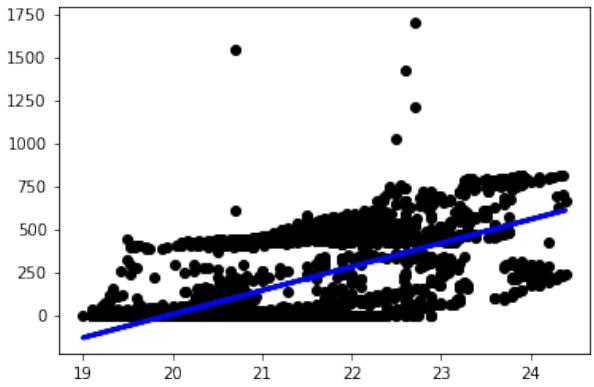



Figure 7: Distribution of data and Regression line

become inconsistent when the datatset is imbalanced and we do not apply any method.

# 4 Advanced Classifiers

- Support Vector Machine, using both linear SVM and non-linear SVM: the linear support vector machine function asks for a parameter, namely C that is the regularization parameter. Being inversely proportional to the strength of regularization we can see how the lower C is, the more the margin improve ( Figure 8), since our aim is the maximization of margin of the hyper-planes. However the plot doesn't seem really good because the model is trained on the complete training set but the plot represent just two principal components, but it explain the variations. For the non-linear support vector machine, we tested the model trying between different Kernel function: linear, polynomial and rfb Kernel. We found out that the best result is reached by the linear function, because our points are linearly separable. In fact, between linear SVM and non-linear SVM with linear kernel function, our dataset better fit in the linear SVM.

- Neural Network: to perform this task, it was necessary to standardize the data. We implemented both the single layer and the multiple layer. The single layer network gives an elevate performance. In this case we can't test it with ROC curve and neither with lift chart, because perceptrons do not output class probability. They just make prediction based on a hard threshold. On the other hand, the multiple layer network, gives a higher accuracy and plotting the loss curve we can see in Figure 9 that there is not overfitting or underfitting because the plot of training loss decreases to a point of stability. In particular, we obtained this result using the default

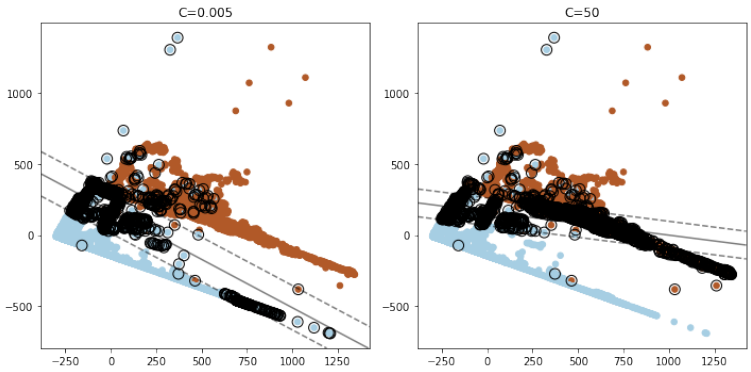



Figure 8: Difference of the hyper-plane with different C

parameter, but we tested other 9 strategies (costant learning-rate, constant with momentum, constant with Nesterov's momentum, inv-scaling learning rate, inv-scaling with momentum, inv-scaling with Nesterov's momentum and Adam) seeing that they have overfitting issues, because the curves continue decrease with experience.

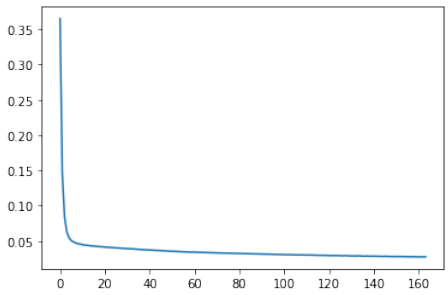



Figure 9: Loss curve Multiple layer network

- Deep Neural Network: in this case we are using Keras library with Tensor flow back-end. We build a Sequential model, with one input Dense layer with 6 neurons, three hidden Dense layers with 16 neurons and one output Dense layer with 1 neuron. We fit the model modifying the parameter batch-size. In particular, we evaluated the loss curve with batch size=30 and batch size=100. The performances are good in both cases but the second model is better because has higher accuracy. So we chose as parameter batch size=100. To understand if our model produces overfitting, we split the training set in training and validation set, in order to evaluate the model on the validation set. Looking at the Figure 10 we can

say there is overfitting, because while the loss in the training stabilize, on the validation start increasing with the number of epochs. To avoid this, we used the Early stopping with Noreg.h5, Regularization and Dropout methods, to find the best epoch to stop, that, according to the first one, results to be number of epoch equal to 37. Finally, throught the RandomizedSearchCv function we found the best model to obtain the higher results: it is composed by a single hidden layers with 32 neurons and uses as optimizer 'Adagrad' and 'Relu' as activation function. However the model we build has the same accuracy of this one (0.77) but the difference is that this one doesn't present overfitting, infact the loss curve of train and validation decrease in the same way.

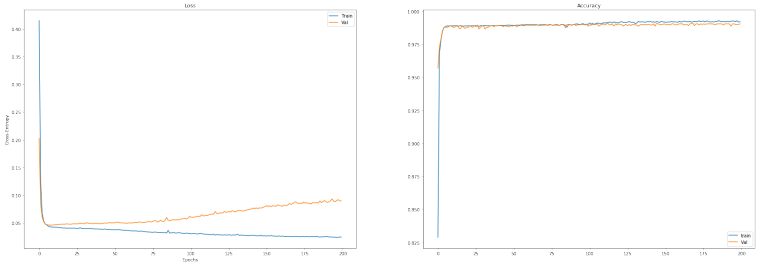


Figure 10: Loss curve and Accuracy on validation set for DNN

- Random Forest: we selected the parameters through the grid search method, finding that the best depth of the tree is 8, the minimum sample leaf is 20 and the minimum sample split is 2. The most important feature emerge to be Light, that is greatly more important than the others attributes(0,59%). Other parameters that affect the classification are Temperature and Co2.
- Bagging: the algorithm split the dataset and train the predictor on different random subset of the training set. In our case we use the out-of-bag evaluation so, as usual, only the 63% of the training instances are sampled for every predictor. So we used the remaining 37% as validation set as a cross-validation and finally we computed the accuracy testing on the test set.
- Boosting: we implemented the algorithm on the training set using as base estimator the Decision tree with max_depth= 1, chosen with Grid Search. In the table are shown the results:

| Classifier | Accuracy | Precision | Recall | F1-score |
|---|---|---|---|---|
| Linear S.V.M. | 0.99 | 0.99 | 0.99 | 0.99 |
| Non Linear S.V.M. | 0.99 | 0.99 | 0.99 | 0.99 |
| Single Layer Network | 0.98 | 0.99 | 0.99 | 0.99 |
| Multiple Layer Network | 0.99 | 0.99 | 0.99 | 0.99 |
| Deep Neural Network | 0.77 | 0.82 | 0.77 | 0.67 |
| Random Forest | 0.99 | 0.99 | 0.99 | 0.99 |
| Bagging | 0.99 | 0.99 | 0.99 | 0.99 |
| Boosting | 0.99 | 0.99 | 0.99 | 0.99 |

### 4.1 Discussion

The advanced classifiers go obviously deeper and are more precise the the basic ones. Nevertheless, in this case it seem useless to applique these classifiers, since the basic ones were already almost perfect and considering that in some cases, like the application on NN, the classifiers are more time consuming, it doesn't worth to use them. However, they confirmed the work done and we can affirm we have high precision in classify our data.

## 5 Time series analysis

We made an analysis of the dataset based on time: we created time series by transforming the attribute "Date" into our index. From the distribution of the observation during time, we noticed that some of them are repetitive: the attribute 'Light' for example, increase and decrease every day with similar shape; the same happens for the class 'Occupancy' for which the repetition has some anomaly after 5 days for 2 days (Figure 11). In the other attributes, as 'Temperature' and 'CO2', we can't observe repetitions, as we could suppose. To make this analysis, we used three different datasets: the original one and two datasets obtained by selecting random rows from the original one. The latters have same dimensions (10280 rows).

### 5.1 Shape based similarity

To compare the time series, we used the two dataset created from the original one, by selecting random rows. So now we have two different time series for same attributes of size equal to half original dataset and we can compare them to see some similarity and calculate distances. We did so for the attributes "Temperature" and "CO2" and we tried to calculate for them the euclidean and Manhattan distance. For example, the attribute 'CO2', we applied some of the transformation methods to avoid the distortions. Computing the distance with the amplitude scaling (Figure 12), we can notice a decrease in distance (from about 12039,7 to 32,5). We couldn't apply the trend removal since there is not a trend and try to remove noise only get worst the situation.

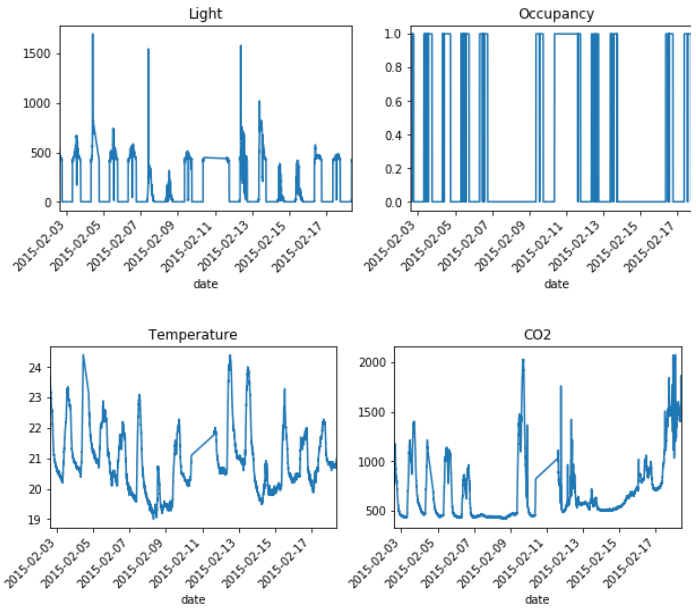
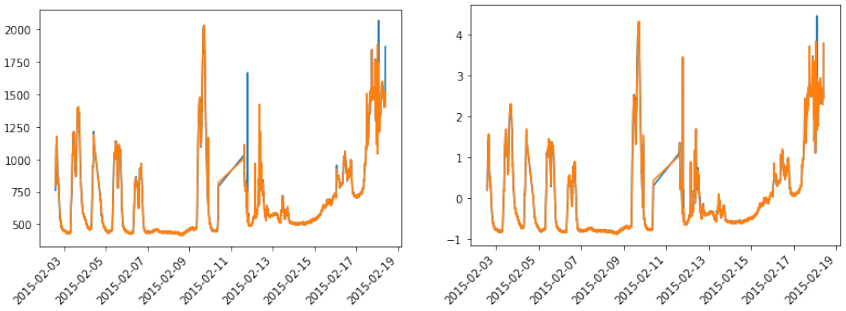


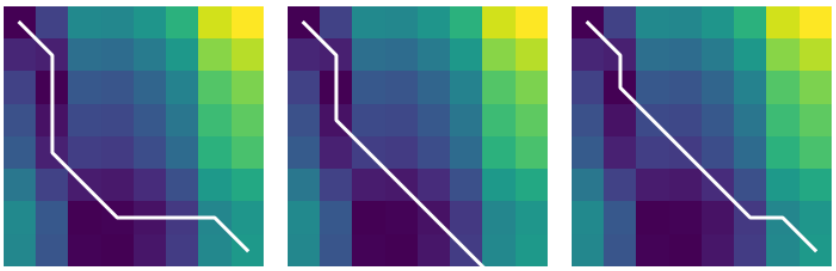

Figure 11: Distribution

On the other hand, 'Temperature', for which time series already had small distances, don't really change since the distance pass from a maximum of 34.8 to a minimum of 32.8.
We decided to try to calculate the dynamic time warping distance between "CO2" of the original dataset and "CO2" of the halved one. Computing the distance on the first 8 rows, it came out to be equal to 37.4. Then we computed cross similarity matrix and accumulated cost matrix to obtain the optimal path, using different methods. In the figure 13 we can see the optimal paths obtained with different global constraints: we have different distance for every of them, in fact we have a distance 46.8 for Itakura parallelogram and 74.8 with Sakoe-Chiba band.
Then, we tried again to compute the distance in the same 8 first rows of the datatset but applying a different library (Pyts) with different values of the parameter distance and we tested the fast approximation method. This analysis brought an increasing of distance.
We noticed that it is not a good idea to compute the distance between time series of different attributes, because also using DTW the distance is very high, and computing the distance matrix and cost matrix, the algorithm can not find an optimal path, in fact it's indicated as a diagonal line, so it doesn't make any alignment.



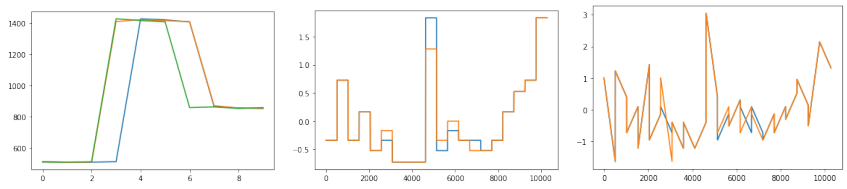

(a) *Original Time Series CO2* (b) *Transformed Time Series CO2*

Figure 12: Change in distances



(a) *Optimal Path* (b) *Path Sakoe Chiba* (c) *Path Itakura*

Figure 13: Different method to compute the path



(a) *Piecewise Linear Approximation* (b) *Symbolic Aggregate Approximation* (c) *1d-Symbolic Aggregate Approximation*

Figure 14: Compression Based Dissimilarity

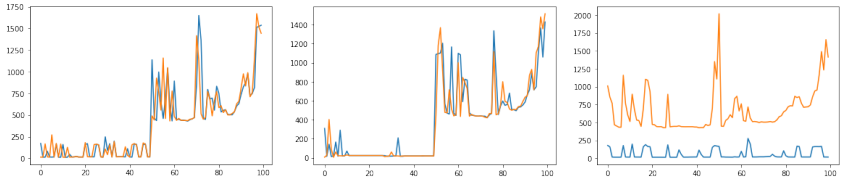
## 5.2 Structural based similarity

We also computed the similarity based on structure, extracting the main features for the attributes and creating a vector to measure the similarity. So, we computed the distance for 'CO2' again and we obtained 5353.6, so it get better than the original situation, but worst than the transformation. This may be because we don't have a really long time series and in this case it's better to compute similarity based on shape.
However we also tried the compression based dissimilarity. Before, we computed the compression based dissimilarity measure and then we applied the transformations we know: Piecewise Linear Approximation, Symbolic Aggregate Approximation and 1d-Symbolic Aggregate Approximation. In this case we computed the compressed distance measure that was 0.86 for the row data, 0.95 transforming the distribution with PLA, 0.67 with SAX and 0.62 with 1-D Sax. In figure 14 some example is shown.

## 5.3 Clustering

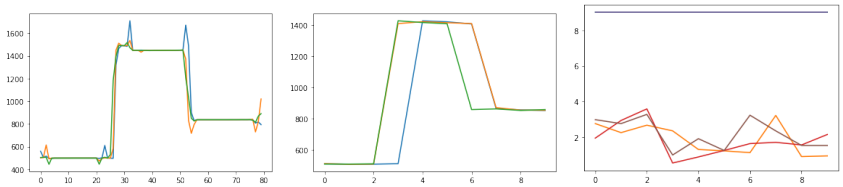
Time series can also be clusterized. We know different types of clustering, so we applied them. To do so, we tried on two different dataset.

- The first one is composed by all the attributes, except for the class 'Occupancy' and 'Humidity Ratio' that was eliminated. We modified the dataset, leading the analysis on a reduced dataset, composed by 100 row and 4 attributes. So, we considered the four attributes as time series, which have same length and that are based on the same time window. Then, we created a transposed array (100×4), reducing the dataset to simplify the representation and calculus. So we applied K-means to this dataset, choosing as parameter k=2, which gave SSE=996938.47. We also computed K-means changing the distance method with dynamic time warping. The parameter k kept the same and in this case the SSE was 701905.38.
  Then, we computed K-means using the dataset's main features, obtaining SSE equal to 395802145.35. Compressing the dataset, we applied DBSCAN, but we obtained really bad results, since we only found one



(a) *K-means with euclidean distance* (b) *K-means with dtw distance* (c) *Feature based k-means*

Figure 15: K-means centroids for multivariate clustering



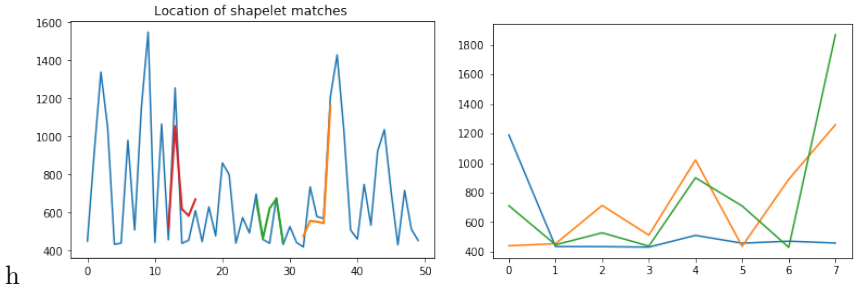
(a) *K-means with DTW distance* (b) *Piecewise K-means* (c) *1D- Symbolic aggregate approximation*

Figure 16: K-means centroids for univariate clustering

cluster, trying between different values of epsilon and minimum samples, and choosing the one suggested by the knee method, that is 0.9, and 8 minimum samples. In figure 15 the distances between centroids are shown.

- For the second try, we used the original dataset and we created 257 consecutive time series of the same attribute (CO2), of length 80. Again, we computed K-means with euclidean distance and dtw distance. Also, we tried between different number of clusters and we obtained the lower inertia with dynamic time warping distance and 3 clusters. Also in this case the SSE is still high, equal to 601590.63. Feature based clustering is really bad, since the SSE is 1150646073.06. Compressing the data, we applied DBSCAN, choosing as parameter epsilon equal to 0.875 and minimum samples equal to 3. In this case we obtained five clusters, but the silhouette is 0.02. Finally, we used two approximation: Piecewise aggregate approximation and 1D- Symbolic aggregate approximation. The latter brought an excellent result, giving SSE equal to 31.84, in contrast to the other technique, for which the inertia was infinite or, however, too high. The main plots are shown in figure 16. The labels of this clustering will be used as classes in the time series classification task (see subsection 10.9)

So, from the results we can say that the clustering didn't reach really good results and this can be because in the first case we have a problem of curse of dimensionality, since we used all the attributes that are different from each




h

(a) *Shapelets first model* (b) *Shapelets third model*

Figure 17: Shapelets

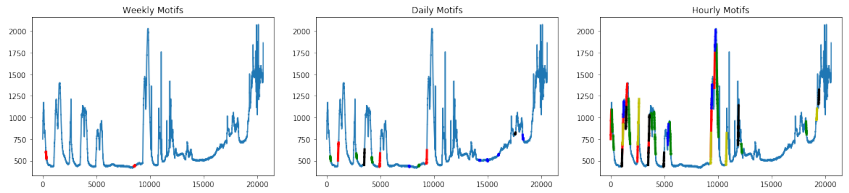
other; in the second case, we used time series belonging to different temporal split. However, we reach a good result approximating the data.

## 5.4 Shapelets

To find shapelets, we had to consider the set of timeseries obtained in the clustering task. So we are considering 20 timeseries, from different time split, but of same length and the class value obtained from the clustering with lowest inertia (with transformation 1-D Sax). We split the set of time series in training and test. We used three methods. The first one compute the distance between the time series and the shapelets candidate: we found three shapelets, shown in the following figure, with accuracy equal to 0.5. The second one, that sees the model as a layered graph, didn't bring any good result: even though the parameter changes, there is not any improvement in accuracy. Finally the third method, gives a score to the shapelets, that is, respectively for each class, [0.38, 0.40, 0.41]. In figure 17 it can be observed the fit of the shapelets on the time series (fig a) and the shapelets found (fig b).

## 5.5 Motifs

To discover any motifs, we used the complete attribute 'CO2' of length 20560 . First of all, we computed the matrix profile considering 3 different subsequences length, to discover weekly, daily and hourly motifs, since the observation were made every minute. For the weekly motifs, in fact, we computed a time window as $60 \times 24 \times 7$, observing some repetitions. For the hourly motifs we attributed 60 to the time window, and finally we computed the daily window as $60 \times 24$. In particular, for the latter, we also made a logarithmic transformation in the time series. We found the motifs shown in the figure 18.




(a) *Weekly motifs* (b) *Daily motifs* (c) *Hourly motifs*

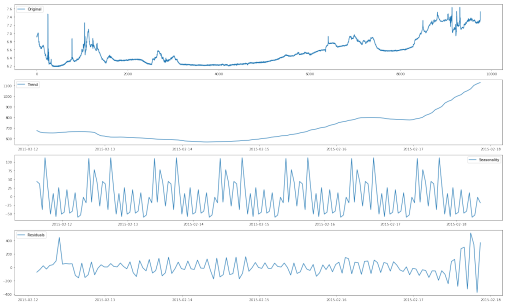
Figure 18: Motifs

## 5.6 Stationarity, Autocorrelation and Partial Autocorrelation

In order to introduce the forecasting task, we have to check some behaviour of the time series we are analysing. That's why we focus our attention on the stationarity. We know that a time series is said to be stationary if its statistical properties such as mean, variance remain constant over time. We know that if a time series is not stationary, the predictive model won't probably be good , that's why sometimes we have to transform the time series to bring stationarity. To check if a time series is or not stationary, we used just the file 'datatest2.txt', using the time series 'CO2'. The first method we used is the Dickey Fuller method, which let me compute the statistic test of the ts and some critical value at different levels (5%,10%, 15%). We can state that a ts is stationary if the statistic test is less than at least one critical level. We computed the DF method to the original time series, finding out that it is not stationary. So we transformed the time series with log transformation, but it didn't help. So we tried with the differencing, that consist in taking the difference of the observation at a particular instant with that at the previous instant; it works well and infact we obtain a test statistics that is lower than 1% critical values, so it is really close to stationarity. We also applied this method to the log differenciated with the mean time series that also works well.
Another way to identify whether time series are stationary, is by decomposition, so showing the trend, the seasonality and the residual. Applying the Dickey Fuller test on the residual (so deleting trend and seasonality), we can observe the values representing stationary time series. Another important information we want to know about time series, is the dependence between lagged ts. To do so, we have to compute the autocorrelation function and the partial autocorrelation function. Again, we computed the two values for all the transformations of the time series. So, as it can be seen from the graph, the original time series of CO2 contain lags that are highly correlated from lag 1 to 50, and this can be due to the fact that we're observing the trend of CO2, that usually is not random for short periods of time, like that we are observing (minutes). This behaviour suggest that the ts is not statioary.The relation is weaker considering the partial autocorrelation, so eliminating any linear dependence (Figure 21). In particular, in this case there is a significant correlation for the first 8 lags. The same thing

```
Results of Dickey-Fuller Test:
Test Statistic                  -0.485794
p-value                          0.894759
#Lags Used                      34.000000
Number of Observations Used    9717.000000
Critical Value (1%)             -3.431023
Critical Value (5%)             -2.861837
Critical Value (10%)            -2.566928
dtype: float64
Results of Dickey-Fuller Test:
Test Statistic                  -0.769233
p-value                          0.827993
#Lags Used                      24.000000
Number of Observations Used    9727.000000
Critical Value (1%)             -3.431022
Critical Value (5%)             -2.861837
Critical Value (10%)            -2.566928
dtype: float64
Results of Dickey-Fuller Test:
Test Statistic                 -19.629795
p-value                          0.000000
#Lags Used                      38.000000
Number of Observations Used    9701.000000
Critical Value (1%)             -3.431024
Critical Value (5%)             -2.861838
Critical Value (10%)            -2.566929
Results of Dickey-Fuller Test:
Test Statistic                 -29.755061
p-value                          0.000000
#Lags Used                       0.000000
Number of Observations Used    9740.000000
Critical Value (1%)             -3.431022
Critical Value (5%)             -2.861837
Critical Value (10%)            -2.566928
dtype: float64
```

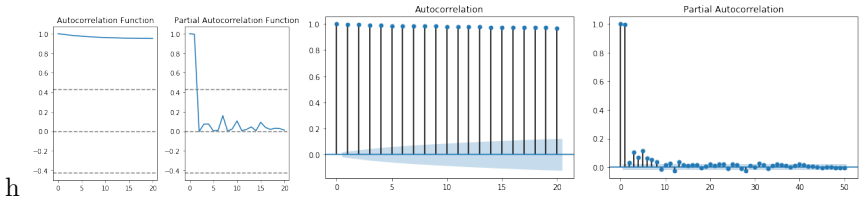
Figure 19: Dickey Fuller method for ts, ts_log,ts_diff, ts_log_diff



(a) *Decomposition of CO2*

```
Results of Dickey-Fuller Test:
Test Statistic                  -4.531807
p-value                          0.000172
#Lags Used                       5.000000
Number of Observations Used    133.000000
Critical Value (1%)             -3.480500
Critical Value (5%)             -2.883528
Critical Value (10%)            -2.578496
```

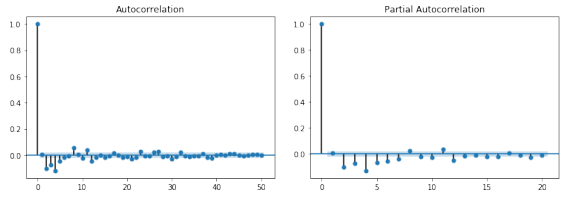
(b) *Dickey Fuller on residuals*




h

(a) *AC and PAC graph* (b) *AC plot* (c) *PAC plot*

Figure 20: AC and PAC




h

Figure 21: AC and PAC in transformed ts

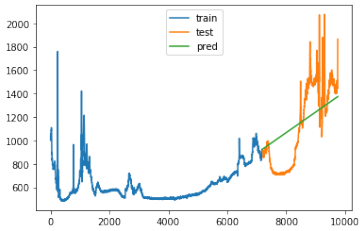
can be observed in the plot. We can see the difference with the differenced timeseries, for which the correlation is significant just for the lag 2 until 6.

## 5.7 Forecasting

To try to forecast the behaviour of our time series, we can apply various methods. For this task we used just one of the three provided datasets, using as time series CO2. First of all, our time series represent observation made every minute, but the index frequency results as 'None'. So, we set up the frequency as minutes and we formatted the ts[1]. Everytime we do this step, the algorithm creates Nan, that we need to fill: we did it with the method backfill. After the split of the dataset into train and test, the train has length $60 \times 24 \times 5$ in order to training the model in 5 days. The rest will be the test set. We first try the forecasting on the original time series, but since it is not stationary, in general it didn't lead to good results. The only exception is with Holt-winter's season method (with smoothing level and smoothing slope of 0.1), that gives good predictions and some error are quite low (Figure 23). Then we tried using the seasonality, which according to Dickey-Fuller is stationary, and results were good: the absolute error are between 0 and 1, MAPE is 1.885, MAXAPE 3.510, but the predictability is less than 0. Similar results are reached with Arima and Sarima, where the order is chosen looking at the pacf and acf plots. Then, we used the forecasting methods on the residuals computed decomposing the ts_log, obtaining the best results. Even though Simple exponential smoothing and Holt-winter's seasonality don't

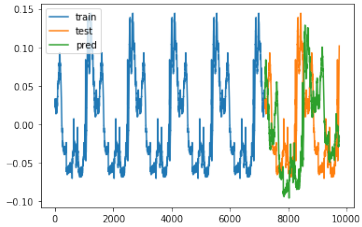
[1]This process is necessary every time we transform the ts or we decompose it or when we split it in train and test.

Figure 22: Seasonal Exponential smoothing on ts




MAE 202.637
RMSE 237.913
MAD 222.359
R2 0.763
MAPE 0.180
MAXAPE 0.638
TAPE 458.494

Figure 23: Simple Exponential smoothing on ts_log




MAE 0.055
RMSE 0.072
MAD 0.042
R2 -0.823
MAPE 3.323
MAXAPE 1142.657
TAPE 8479.131

bring any good result: the exponential smoothing, with seasonal period=2500 reach a predictability (R2) of 1 and low level of error. Arima model, to which give it an order (2,0,1) also reached a good predictability and low error (Figure 24).
We also computed forecasting for the transformed time series (differentiated and applying the moving average), but in every case the predictability was less or equal to 0 and we didn't reach nothing interesting. More information can be found in the GitHub channel.

## 5.8 Univariate Classification

To classify our time series we used the set of ts created in the clustering task (that we will call X), composed by 257 consecutive ts of the same attribute (CO2), of lenght 80. We exploited its labels as classes of the ts (named y). The set is homogeneously distributed on the three classes. To train the model, we split the dataset into the train and test as usual.
We first tried to classify time series based on shapelet model, but it didn't work, because the accuracy is lower that the naive classifier and reach a maximum of 0.48; plus, it doesn't classifie well all the classes, but just two of them (in fact for one class the f1-score is almost 0). Between the other different classifiers we tried, not all of them made a good job: in the table above are shown the results, reporting the accuracy of classifiers and the F1-score for each class.

| Approach | Classifier | Accuracy | F1-score |
|---|---|---|---|
| Shapelets | Shapelet Model | 0.487 | [0.58, 0.5, 0.009] |

| | | | |
|---|---|---|---|
| Distance between shapelets | K-NN | 0.53 | [0.61 0.52 0.42] |
| | Decision Tree | 0.53 | [0.49 0.49 0.58] |
| Features based | K-NN | 0.41 | [0.46 0.41 0.29] |
| | Decision Tree | 0.423 | [0.26 0.56 0.30] |
| Time series distance | K-NN | 0.538 | [0.62, 0.58, 0.23] |
| | Decision Tree | 0.538 | [0.54, 0.625, 0.4] |
| NN | CNN | 0.36 | [0.53, 0, 0] |
| | LSTM | 0.28 | [0, 0, 0.44] |

About NN, the first model is composed by 4 Dense layers, with 16, 64, 64 and 3 neuron each; the second model is composed by three LSTM layers with 256, 64 and 3 neurons each.

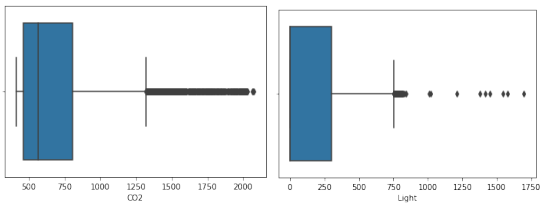
### 5.9 Multivariate Classification

We also tried to classify multidimensional time series. To do so we created an array with 257 element of length 80 and 6 dimensions. It was necessary to apply clustering in order to obtain the labels of the classes.
After split the the array in train and test, it is shaped as follow: 80 timesteps, 3 classes and 6 features. To classify it, we chose to apply the CNN, building a sequential model, composed by 3 LSTM and 1 Dense layers, containing respectively 6, 143, 143 and 3 neurons, where the first is chosen based on the number of features, the last based on the number of classes. The activation function chosen was Leacky Relu and we also add the dropout regularization term, in order to avoid overfitting. After training the model for 50 epochs, we didn't reach a really good result, since CNN only classify two classes and assign one class F1-score equal to 0.
We then built another sequential model, using 3 convolution layers and one dense layer: this had the best results, in fact has accuracy equal to 0.91, and F1-score, respectively for the three classes, [0.90, 0.96, 0.79]

## 6 Sequential pattern mining

We exploited the temporal distribution of our data to discover if it exists some relevant sequential pattern between our data.
As first try, we used the time series CO2. This is composed by continuous values, so it was necessary to let it discrete and we transformed it with SAX transformation, choosing 22 segments and 10 symbols. So now, every range of continuous values has an assigned value:

- -1.0364333894937898: 0,
- -0.6744897501960817: 1,
- -0.38532046640756773: 2,
- -0.12566134685507402: 3,
- 0.0: 4,
- 0.12566134685507416: 5,
- 0.38532046640756773: 6,
- 0.6744897501960817: 7,
- 1.0364333894937898: 8,
- 1.6448536269514722: 9




(a) *CO2* (b) *Light*

Figure 24: Boxplots

These symbols will be used to see if there are values repeating between data. To make the analysis we split the dataset of 8200 rows in time series of length 100, that, when transformed became 150. Applying the PrefixSpan algorithm we detected the first 30 patterns, finding that the maximum support is 82. Analysing the frequent subsequences we noticed that every time there is a change, then it go back to the medium value (4). This suggest that the CO2 is always kept at a certain level. Then, since the most frequent subsequence is 2, we can say that the level of the room is in general medium-low. Here there is an example:
Support 82: [2, 2, 2, 2, 2, 2, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4];
Support 78: [6, 6, 6, 6, 6, 6, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4];
Support 73: [8, 8, 8, 8, 8, 8, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4];

# 7 Outliers Detection

Some of the attributes of our dataset contain observations which are significantly different from the others. To easily detect outliers, we can visualize the distribution of the attributes through the boxplots or plots, as shown in the Figure 24. It seems to be any in Humidity and Temperature.
The boxplots show that in all cases, the outliers are in the top. Another way to check if in our dataset in one dimension there are outliers is with Grabbs' Test, that let us refuse the null hypothesis. We were interested in identifying the 1% top outliers in the dataset. To do so, we used one of the most common approach, that is the interquartile rate. In particular we computed the 99th percentile, so that all the values that are bigger than that are considered outliers.
To use other approaches, we used the library Pyod. Using a density based approach, we applied the LOF. Fitting the model, we found out there are two groups labeled -1 and 1, in which there are respectively 501 and 7642 data. We computed a decision score, in particular we computed the negative outlier factor, that implies that outliers are those data which have a factor less than -1.499. In our case this factor reach a maximum of -0.949 and a minimum of -13.846. A distance based approach is DBSCAN, for which we used epsilon= 90 and minimum samples=20. Obtaining a silhouette of 0.623, we found two clustering of 29 and 8114 points. In Figure 25, the violet points are the outliers.

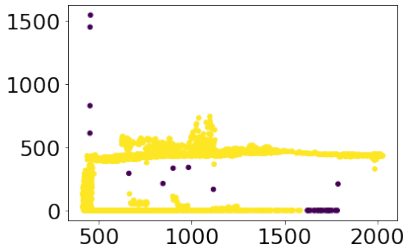



Figure 25: DBSCAN clustering

| cluster | 0 | 1 | 2 | 3 | 4 | score |
|---|---|---|---|---|---|---|
| 0 | -0.045627 | -0.074684 | -0.044469 | -0.130558 | -0.093676 | 1.924507 |
| 1 | 1.297338 | 2.123553 | 1.264414 | 3.712241 | 2.663567 | 5.431560 |

Figure 26: Outliers decision score

Classifying with KNN with a level of contamination equal to 0.01, we can find two clusters. The prediction function says that our classyfier found 523 errors with accuracy equal to 0.79. Finally, we built a NN with 4 layers and [5,1,1,5] neurons, in order to use the Autoencoder. We can detect the anomaly on those records which have the decision score greater than a certain threshold, that, looking at the histogram, we chose to be 4. So we found two clusters and looking at the distances (Figure 26), we can see that points in cluster 1 (anomaly cluster) have a greater distance than the others.

# 8 Explainability

Finally, we tried to give a more practical meaning to the result of our classifiers. As an example, we run the Random Forest classifier and used the Lime method to explain it. Explaining the 10th row of the test set, we can see the importance of the feature and how these determine if a record is part of the group "Occupancy=0" or "Occupancy=1". As we can see the most important feature is Light.

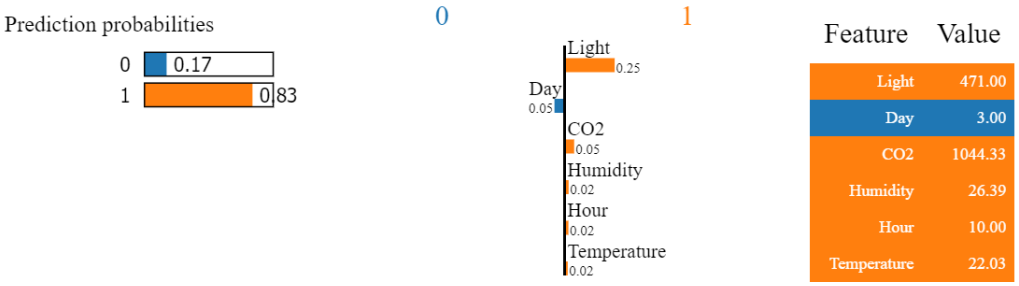



Figure 27: Explainability