Università degli Studi di Pisa

PROJECT REPORT

Asia Trinci, 550342
Lorenzo Maggio, 537366
Riccardo Figliozzi 609652

A.A. 2019/2020

Index

Task-1

- Data understanding and preparation
- Classification tasks
- Dimensionality reduction
- Imbalance learning
- Linear Regression

Task-2

- Neural Network
- SVM
- Ensemble classifiers

Task-3

- Time series analysis and Classification
- Time series Clustering
- Time series Forecasting

Task-4

- Sequential Pattern Mining

Task-5

- Outlier Detection

TASK 1 - Basic Classifiers and Evaluation

1.1 Data Understanding And Preparation

This report examines the Occupancy Detection Dataset, provided by UCI Machine Learning Repository.

The data are available in three different files, that are supposed to be a training dataset, a validation dataset and a test dataset. To start the understanding process, we need a first description of them:

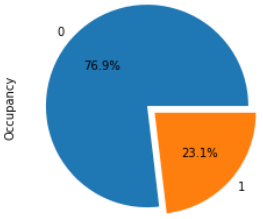
DATASET	DEFINITION	REGISTRATION	RECORD
Data Training	Recorded with the door closed	From 2015-02-04, 17:51:00, to 2015-02-10, 09:33:00	8143
Data Test	Recorded with the door closed	From 2015-02-02, 14:19:00, to 2015-02-04, 10:43:00	2665
Data Test 2	Recorded with the door open	From 2015-02-11, 14:48:00, to 2015-02-18, 09:19:00	9752

We can notice that the three datasets are recorded in sequence, but temporally separated by some hours of missing data.
The dataset is composed by seven attributes, described in the table below:

ATTRIBUTES	TYPE	DESCRIPTION	MISSING VALUES
dates	Object datetime	date: year-month-day time: hour:minute:second	0
Temperature	Numerical Continuous	Temperature in Celsius	0
Humidity	Numerical Continuous	Humidity express by %	0
Light	Numerical Continuous	Light in Lux	0
CO2	Numerical Continuous	CO2 in ppm	0
Humidity Ratio	Numerical Continuous	Derived from temperature and relative humidity, in kg-water-vapor/kg-air	0
Occupancy	Numerical Binary	Express the status of the room: 0 if the room is not occupied, 1 if the room is occupied	0

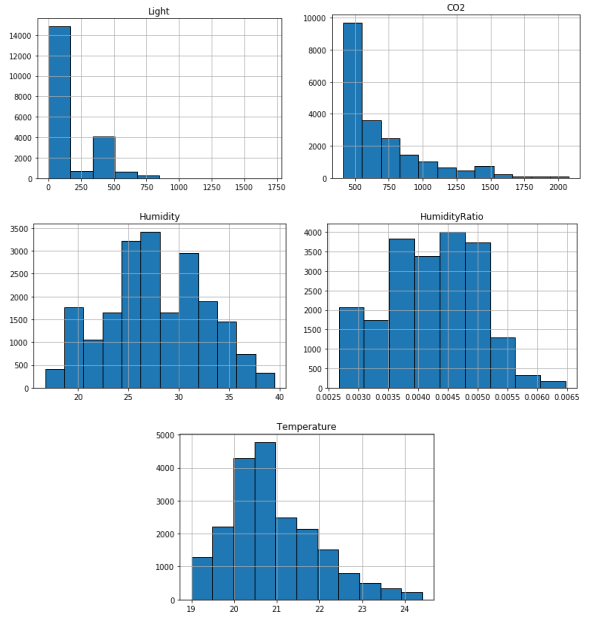
To develop a more complete analysis we decided to merge the records in one dataset, following the chronological order of data registration. Thus, we have obtained a full dataset of 20560 observations.

The feature "Occupancy" is the attribute that we want to predict through the classification algorithms.



Its distribution in the dataset is 76.9% of records (15810) classified as 0 and 23.1% (4750) as 1. So, the dataset is not heavily unbalanced.

Diving into the dataset we discovered the distribution of the numerical attributes, shown in the graphs below.



We can observe that the data of Light and CO2 are concentrated in the first columns of the distribution, while Humidity and Humidity Ratio share a very similar distribution. Also, Temperature has a more left-skewed distribution.

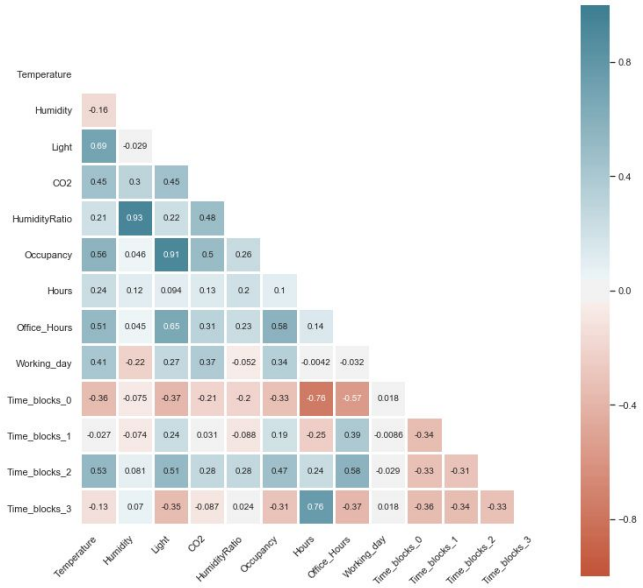
Before starting the classification tasks, we need to do some data preparation.

From “dates” 5 new features were extracted, 2 categorical, “Dates” and “Time”, and 3 numerical, “Hours”, “Minutes” and “Seconds”, which were used to separate the data into different intervals of time.

Indeed, we created other 6 binary features:

- **Time_blocks**: 4 variables, that divide the daytime in 4 different time blocks (time_block_1: 0.00 h - 5.59 h, time_block_2: 6.00 h -11.59 h, time_block_3: 12.00 h -17.59 h, time_block_4: 18.00 h - 23.59 h).
- **Working_day**: variable that gives 1 when it takes into consideration a day between Monday and Friday, while 0 if the day is Saturday or Sunday. This variable helps us on dividing days in working days and free days where the percentage of occupancy should decrease.
- **Office_Hours**: equal to 1 from 8.00 h to 18.59 h, so in the possible office hours, where the occupancy should be higher, and equal to 0 otherwise.

Then, we analyzed the correlation matrix between the attributes, original and created:



Here it is possible to notice how values like “Light” and “Occupancy” are highly related, probably because in the room there is a sensor that turns the light on or off when someone enters or leaves.

Another strong correlation is between “Humidity” and “Humidity Ratio” since the second one takes values from the first one.

Moreover, through the new created variables, it is possible to see how much attributes like “Light” and “Occupancy” are correlated to the different time blocks. For example, we can see that during the time blocks of the early morning and deep night the feature “Occupancy” is negatively correlated with the blocks. This also happens for “Light” and the other features.

Even better is the correlation with Office_hours, which is able to capture a more specific aspect of the daytimes and summarizes in one variable what has been observed through the Time_blocks attributes. Less than expected is instead the correlation with “Working day”.

1.2 Classification Tasks

In the following section, we trained different algorithms to perform the classification. In our case the main goal is to predict the variable “Occupancy”, chosen as class label, that indicates if the room was occupied or not.

The algorithms that we applied are K-NN, Logistic Regression, Decision Tree and Naive Bayes.

To begin, since the classification algorithms require binary or numerical attributes, we selected these 10 features from the prepared dataset:

"Light", "Temperature", "Humidity Ratio", "CO2", "time_blocks_1", "time_blocks_2", "time_blocks_3", "time_blocks_4", "Office_Hours" and "Working_day".

Also, since "Humidity" and "Humidity Ratio" are highly correlated we decide to keep only one of them, that is "HumidityRatio", because it has a higher correlation with the class label "Occupancy".

To increase the performance of the classification algorithms we normalized the dataset, by using MinMaxScaler, which allows us to preserve the shape of the original distribution of the attributes, scaling them within the range [0,1].
Then we randomly split the dataset in two different parts: 70% as a training set, to build the model, and 30% as a test set.

K-Nearest Neighbour

The first classifier implemented is K-NN, for which we have evaluated the right parameters executing a grid-search and a 10-cross validation over the following values:

- **n_neighbors**: range (1, 50)
- **weights**: 'uniform', 'distance'
- **metric**:'minkowski','euclidean', 'manhattan'

The suggestion for the best parameters was:

- **k** = 9
- **weight** = 'uniform'
- **metric** = 'minkowski'

The value of k is the most important parameter of this algorithm, since it decides how many neighbours will be considered to classify a record. We can notice that the selected is different from the one suggested by the general rule $k = \sqrt{N}$.
The weight set to uniform means that all points in each neighborhood are weighted equally, so they do not reduce the impact of the chosen k.
Minkowski is instead the metric used to calculate the distance between points. It is a generalization of the Euclidean and Manhattan metric, but since the parameter p = 2 by default, it is exactly equal to the Euclidean metric.

Naïve Bayes Classifier

The second classifier we applied is the Naive Bayes classifier with a Gaussian approach. To deal with continuous attributes the main assumption made by this approach is that they are Gaussian distributed.

To discover the best model configuration, we have again executed a grid search and a 10 cross-validation over the following two parameters:

- **Priors,** the prior probabilities of the classes: any combination of decimal numbers whose sum is 1 ([0.1, 0.9], [0.2, 0.8], ...)
- **Var smoothing**, portion of the largest variance of all features that is added to variances for calculation stability: [1e-10,1e-9, 1e-8, 1e-7, 1e-6, 1e-5, 1e-4, 1e-3, 1e-2, 1e-1]

After that, the best model was found setting these parameters:

- **Priors** = [0.9, 0.1]
- **Var smoothing** = 1e-2

Anyway, the classifier's performance mainly depends on the truth of the assumption made.

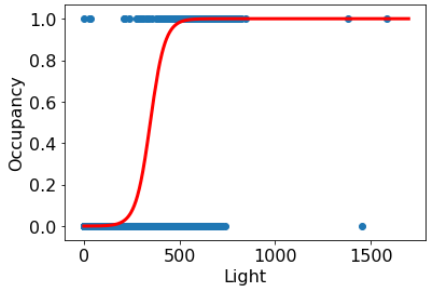
Logistic Regression

Logistic Regression is a binary classification algorithm, so, since the dependent variable Occupancy is binary, it should fit well the data.

We tried different combinations of attributes (ex. Temperature + Light, Temperature + Light + Humidity, etc.), and it resulted that those combinations including the feature 'Light', were the ones with the best performance.

Thus, we decided to use it as a unique variable to build a single model and predict "Occupancy". At the same time, we also set different C values. It express the inverse of regularization strength, then smaller C values specify stronger regularization. The best C parameter found was C = 1000.

This plot shows how the Single Logistic Regression based on Light classifies the values.



Only the "Light" variable is required to achieve 98% accuracy on this dataset.

It is very likely that the lab rooms, in which the environmental variables were recorded, had a light sensor that turned internal lights on, when the room was occupied.

Instead, for the Multiple Regression problem, we selected the same attributes of the other classifications.
This combination, gave us an even better result than the Single logistic regression, selecting as C parameter 1.5 (through a grid search).

Decision Tree Classifier

The Decision Tree is a supervised machine learning algorithm where the data is continuously split according to certain parameters.

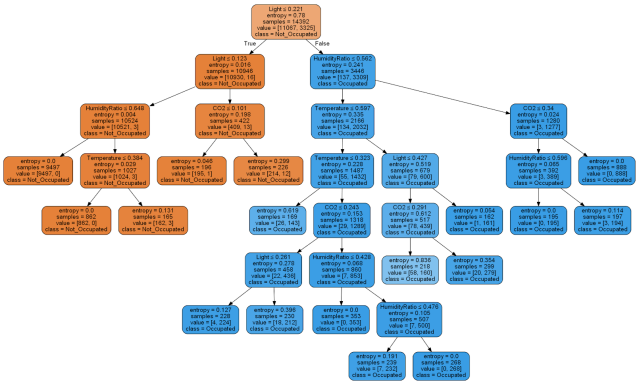
Before building it, for the purpose of determining the best model this time we executed a random search (because the search space was too wide, and computationally expensive, to perform a grid search) over the following four parameters:

- **max_depth,** the maximum depth of the tree: range (2, 50)
- **min_sample_split,** the minimum number of samples required to split an internal node: [2, 5, 10, 15, 20, 30, 50, 100, 150]
- **min_sample_leaf,** the minimum number of samples required to be at a leaf node: [1, 5, 10, 15, 20, 30, 50, 100, 150]
- **criterion,** the function to measure the quality of a split: 'gini', 'entropy'

Note that the first three parameters are the ones in control of the stopping criteria of the tree growing, which may help to control the overfitting of the model.

For each combination of parameters, we also executed a 10 cross-validation, finally finding as best model the one with parameters:

- **Criterion** = 'entropy'
- **max_depth** = 10
- **min_samples_split** = 150
- **min_samples_leaf** = 150

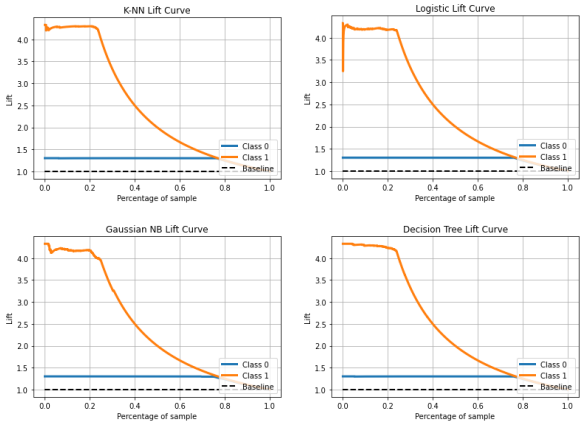


Light is the attribute that lead the first split, and it influences all the result. If the Light is less than 0.221 (normalized) the record will be classified as 0, not occupied, otherwise it will be labeled as 1, as can be seen from the tree on the previous page.

Classifier's Performance Result

Applying the previous models to the test set we obtained the following performance result:

Classifier	Accuracy	Precision	Recall	F1	ROC
K-NN	0.9920	1.00 (0) 0.98 (1)	0.99 (0) 0.99 (1)	0.99 (0) 0.98 (1)	0.9899
Naive Bayes	0.9586	1.00 (0) 0.86 (1)	0.95 (0) 0.99 (1)	0.97 (0) 0.92 (1)	0.9679
Multiple Logistic	0.9904	1.00 (0) 0.96 (1)	0.99 (0) 1.00 (1)	0.99 (0) 0.98 (1)	0.9930
Decision Tree	0.9889	1.00 (0) 0.96 (1)	0.99 (0) 0.99 (1)	0.99 (0) 0.98 (1)	0.9906



Firstly, looking at the ROC we can state that the Naive Bayes is the worst model.

While comparing the other three models precision and recall for the class 1 (occupied), the one we are looking for, we can notice that they are all very high and almost equal to each other. It means that all these methods misclassified very few records.

However, the difference between the Decision Tree and the other two models is more evident considering the accuracy. The K-NN and the Logistic have the best performance from this point of view, but the Multiple Logistic Regression can be seen as our best model for his slightly better ROC.

1.3 Dimensionality Reduction

Dimensionality reduction helps to reduce the dataset using different methods to select the principal variables. The methods we chose to apply are Variance Threshold, Univariate Feature selection, Recursive Feature Elimination and PCA.

Since our original dataset does not show a high dimensionality, we decided to increase it, only for this task, by adding 16 columns, created by making the relationship between the various attributes (i.e. Light/Temperature, Humidity/CO2, etc.).

Then, the new dataset has been standardized with StandardScaler and the classification algorithms shown previously (Decision Tree, K-NN, Naive Bayes) have been re-applied to the dataset so that a comparison between the classifications on a resized dataset could be made.

MODELS	Accuracy	Precision	Recall	F1	ROC
Decision Tree	0.9893	1.00 (0) 0.96 (1)	0.99 (0) 1.00 (1)	0.99 (0) 0.98(1)	0.9913
Naive Bayes	0.9750	1.00 (0) 0.90 (1)	0.97 (0) 1.00 (1)	0.98 (0) 0.94 (1)	0.9809
K-NN	0.9909	1.00 (0) 0.97 (1)	0.99 (0) 0.99 (1)	0.99 (0) 0.98(1)	0.9906

Variance Threshold

The first feature selection method used is the Variance Threshold. This technique allows us to eliminate those attributes that do not exceed a certain threshold of variance, and which add little information to the dataset, since they remain almost constant.

The threshold chosen in our case is 0.5, which selects 21 attributes among the 27 available. Considering instead the extreme thresholds, 0 and 1, the selected attributes were respectively 27 (therefore no attribute was discarded) and 8, but the results obtained on the classification did not improve compared to the threshold of 0.5, so we considered only the latter.

The performance results with it were:

VARIANCE THRESHOLD	Accuracy	Precision	Recall	F1	ROC
Decision Tree	0.9893	1.00 (0) 0.96 (1)	0.99 (0) 1.00 (1)	0.99 (0) 0.98(1)	0.9913
Naive Bayes	0.9640	1.00 (0) 0.87 (1)	0.95 (0) 1.00 (1)	0.98 (0) 0.93 (1)	0.9758
K-NN	0.9909	1.00 (0) 0.97 (1)	0.99 (0) 0.99 (1)	0.99 (0) 0.98(1)	0.9909

Univariate Feature Selection

About the Univariate Feature Selection, which selects the attributes based on statistical tests, we have chosen to apply two of these tests, the f-test and the mutual information test. Between these two, the best was the mutual information test, because it was able to better capture the non-linear relationships between the attributes.

Univariate Feature Selection (Mutual)	Accuracy	Precision	Recall	F1	ROC
Decision Tree	0.9893	1.00 (0) 0.96 (1)	0.99 (0) 1.00 (1)	0.99 (0) 0.98(1)	0.9913
Naive Bayes	0.9606	1.00 (0) 0.86 (1)	0.95 (0) 1.00 (1)	0.97 (0) 0.92 (1)	0.9739
K-NN	0.9924	1.00 (0) 0.97 (1)	0.99 (0) 1.00 (1)	1.00 (0) 0.98(1)	0.9936

Recursive Feature Selection

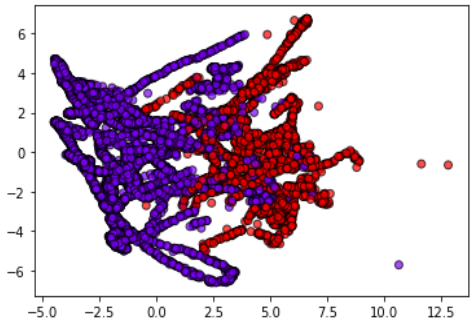
We set the selection method for the model on the feature importance of the decision tree. This method selected only one attribute, “Light/Temperature” which is the one guiding the building of the tree, with almost 98% of importance.

Recursive Feature Selection	Accuracy	Precision	Recall	F1	ROC
Decision Tree	0.9893	1.00 (0) 0.96 (1)	0.99 (0) 1.00 (1)	0.99 (0) 0.98(1)	0.9913
Naive Bayes	0.9760	1.00 (0) 0.91 (1)	0.97 (0) 1.00 (1)	0.98 (0) 0.95 (1)	0.9837
K-NN	0.9889	1.00 (0) 0.96 (1)	0.99 (0) 0.99 (1)	0.99 (0) 0.98(1)	0.9904

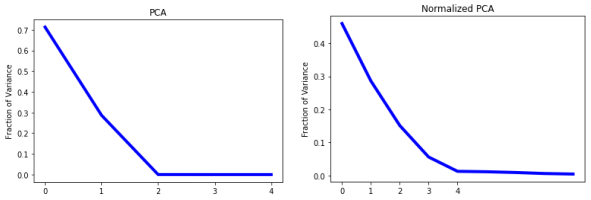
Comparing the different performances before and after the reduction, we can see that the decision tree remains always the same. This is probably linked to the fact that it is mainly built over one attribute, Light/Temperature. On the other hand, while Naive Bayes gets slightly worse, the K-NN has a small increase in his performance.

Principal Component Analysis

Another implemented technique of dimensionality reduction was PCA, where we wanted to extract two main components, which are shown in the following plot:



We notice that without normalizing the data the variance is captured only by the first two attributes, while with the standard scaler normalization, the explained variance ratio of the two attributes selected is respectively 0.456 for the first attribute and 0.284 for the second. This means that the other attributes (mainly the third and the fourth) capture the remaining 26% of the variance.



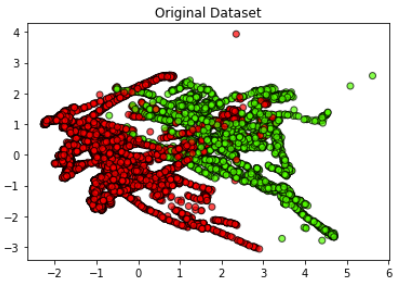
The result of PCA's application on the classifier algorithm is the following:

PRINCIPAL COMPONENT ANALYSIS	Accuracy	Precision	Recall	F1	ROC
Decision Tree	0.9680	0.98 (0) 0.93 (1)	0.98 (0) 0.93 (1)	0.98 (0) 0.93 (1)	0.9556
Naive Bayes	0.9142	0.96 (0) 0.77 (1)	0.92 (0) 0.89 (1)	0.94 (0) 0.83 (1)	0.9049
K-NN	0.9779	0.99 (0) 0.95 (1)	0.98 (0) 0.96 (1)	0.99 (0) 0.95(1)	0.9704

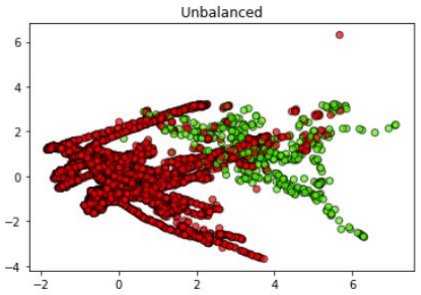
With respect to the other dimensionality reduction methods, the PCA is the worst one. It doesn't improve the performance of any of the classifier and moreover it also reduce the Decision Tree overall results. An interpretation could be that this method does not select the attribute Light/Temperature.

1.4 Imbalanced Learning

The objective of this task is to understand how to deal with an unbalanced dataset applying different algorithms.

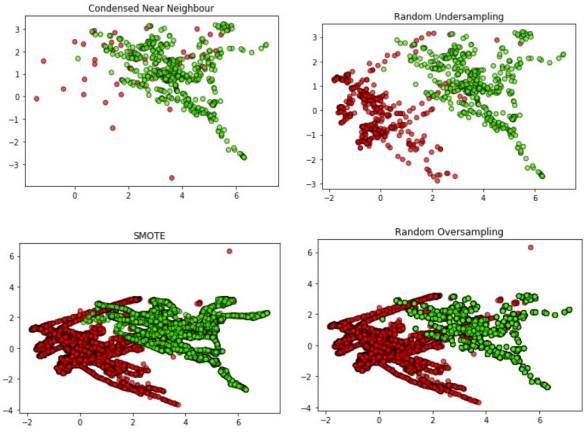


Firstly, we unbalanced the dataset removing 3091 rows with "Occupancy" value equal to 1, in order to reach a strong majority-minority distribution, so getting 96% (0) and 4% (1).



Once we have obtained the unbalanced dataset we applied our three classifiers to this new dataset and we analyzed how the performances change.

Then we re-balanced the dataset with four different methods (Random Undersampling, Condensed Near Neighbor, random Oversampling, Smote) and we runned again all the classifiers. These are the balanced dataset obtained.



In the following tables is shown how each classifier's performance is changed thanks to their best balancing method applied:

DECISION TREE	Accuracy	Precision	Recall	F1	ROC
Unbalanced	0.986	0.99 (0) 0.86 (1)	0.99 (0) 0.81 (1)	0.99 (0) 0.83 (1)	0.90
Random UNDER	0.991	1.00 (0) 0.97 (1)	0.99 (0) 0.99 (1)	0.99 (0) 0.98 (1)	0.994

NAIVE BAYES	Accuracy	Precision	Recall	F1	ROC
Unbalanced and Balanced	0.952	1.00 (0) 0.46 (1)	0.95 (0) 0.98 (1)	0.97 (0) 0.62 (1)	0.99

K-NN	Accuracy	Precision	Recall	F1	ROC
Unbalanced	0.984	0.99 (0) 0.92 (1)	1.00 (0) 0.80 (1)	0.99 (0) 0.86 (1)	0.984
SMOTE	0.986	1.00 (0) 0.77 (1)	0.99 (0) 0.96 (1)	0.99 (0) 0.85 (1)	0.974

As can be seen in the tables above, for the Decision Tree classifier the performance is subject to a decrease when the dataset is unbalanced. However, after the application of the Random Undersampling technique the classifier obtained an increase of its performance even surpassing the original. We can suggest this happen because the new configuration of the dataset is more suitable than the original for this classifier.

For the Naive Bayes classifier after applying all these four techniques there was no improvement of the scores. Probably because the Naive Bayes classifier, among all, is the classifier that after the unbalancing has a less decrease in its performance. So when the dataset was rebalanced applying the techniques the performance couldn't increase significantly.

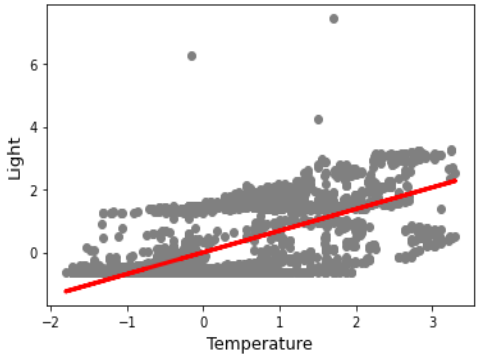
For the K-NN classifier, the best rebalancing technique is SMOTE. However, after rebalancing the dataset we only obtained a slight increase of the accuracy in spite of the precision on class 1 and then on the ROC value.

1.5 Linear Regression

In Regression problems we try to predict a continuous dependent variable (Y) through other independent variables (X).

We first defined a **Single linear regression** problem, with two attributes taken from the dataset. "Temperature" as independent variable, used to predict the dependent variable "Light". These two were chosen because they are the continuous attributes with the highest correlation (apart from Humidity and Humidity Ratio whose relationship seems obvious).

The Linear Regression model has been applied to allow us finding the values for the intercept and coefficient of the line that best fits the data.



- **Coefficients**: 0.68882
- **Intercept**: -0.00197

This means that for each variation of one unit of temperature, the light varies by about 0.69%.

The evaluation of the model is made calculating the R^2 score, Mean Squared Error and Mean Absolute Error:

- **R^2**: 0.475
- **MSE**: 0.526
- **MAE**: 0.576

Next, we built a **Multiple Linear Regression** problem, using again "Light" as the dependent variable and all the remaining continuous attributes of the dataset as independents. We firstly applied the Linear

Regression, then the Lasso and eventually the Ridge Regression.

These last two are regularization methods, particularly efficient when are considered more attributes, that allow to regularize the coefficients and reduce complexity by preventing model overfitting.
For both, the alpha parameter controls the regularization strength and its balancing with the squared loss function.
To tune it a 10 cross-validation was carried out, using in one case RidgeCV and in the other LassoCV, resulting in:

- α **RIDGE** = 3.068
- α **LASSO** = 0.005

Obtaining:
LINEAR
- **Coefficients**: [0.482, -0.369, 0.150, 0.396]
- **Intercept**: -0.00253

LASSO
- **Coefficients**: [0.609, 0.0, 0.158, 0.0169]
- **Intercept**: -0.00263

RIDGE
- **Coefficients**: [0.496, 0.329, 0.151, 0.356]
- **Intercept**: -0.00254

From the values of the coefficients, we can notice that the Lasso makes a kind of feature selection, resetting the coefficient of the "Humidity" attribute to zero, and raising the coefficient of “Temperature”. All the other values are similar between the three methods.

This behavior is reflected in the performance result, that appear to be the same for all the methods applied, as reported below:

- **R^2**: 0.501
- **MSE**: 0.500
- **MAE**: 0.554

Therefore, by increasing the number of variables, the performance of the regression seems to improve, with an increase in the determination coefficient, which detects an increase in the total variability of the dependent variable explained by the independents. A small reduction of the error, absolute and square, is also noticed.

TASK 2 - Advanced Classifiers

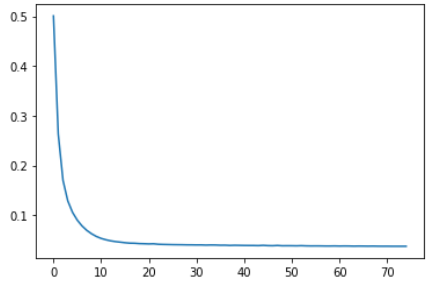
To perform this task, we applied advanced classifier techniques such as Neural Network, Deep Neural Network, Support Vector Machine and Ensemble classifiers.

2.1 Neural Networks

A Neural Network (NN) is an interconnected group of nodes, known as a perceptron, which connect the input with the output layer, between these two there are also other layers, called hidden, which allow to compute more complex decisions.
To find the best configuration of a NN we need to split the dataset in three parts: Training set, Validation set and Test set.
The validation set is used to decide when to stop the training, by monitoring his loss through the epochs to select the best configuration of the model.

Firstly, we tried different Single and Multi-layer Perceptron (MLP) classifiers to understand which parameters optimize our model.

We got really good results using the default parameter, correspondent to a model with a single hidden layer composed by 100 units, activation = 'relu', learning_rate = 'constant', solver = 'adam', early_stopping = False.
The loss curve obtained is really stable and the accuracy reached is equal to 0.9892.

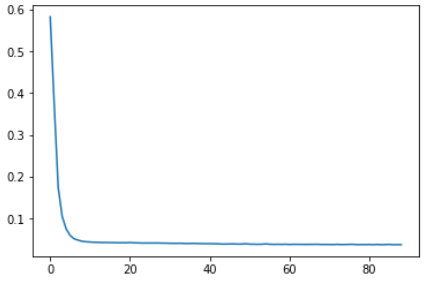


But, after some trail, an even better model appeared to be a multilayer perceptron with parameters:

- **hidden_layer_sizes** = 20, 15, 5
- **solver** = 'adam'
- **shuffle** = True
- **activation** = 'relu'
- **early_stopping** = False
- **learning_rate** = ‘adaptive’

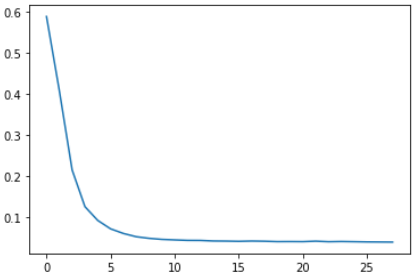
Therefore, it has three hidden layers, with respectively 20, 15 and 5 units (it’s indeed a deep network), with relu as activation function. The solver ‘adam’ is a stochastic gradient-based optimizer that helps to converge faster. It uses an adaptive learning rate, reduced using a moving average of the squared gradients to which is added an exponentially decaying average of the past gradients. The early stopping is usually used to prevent overfitting, but in this case wasn’t activated.

By applying it, we observed that the model was stable, with a loss curve that reaches its minimum faster with respect to the single-hidden layer model. Moreover, it had an higher accuracy value (0.9893).



If early stopping is instead set to True, 10% of training data is automatically set aside as validation and the training is terminated when the validation score stops improving for a certain number of consecutive epochs.

Switching this parameter for our model to True the loss curve is the following.



The decrease of the loss is slower, but eventually it reaches almost the same constant level of the previous.

Deep Neural Networks

Using the Keras library, we defined our model for DNN with the same parameters used for the multilayer perceptron classifier, but we added to it another dense layer, of 10 units, between the 15 and the 5 units layer. Finally, there was the output layer, which had the sigmoid as activation function.

A confirmation that the best performance was generated by the previously selected parameters was obtained, comparing the loss curve and accuracy with several random deep network created.
The loss curve tends to stabilize faster when the previous selected parameters are used, on the contrary, using different ones, the loss curve delays its stabilization and is constantly unstable during the process.

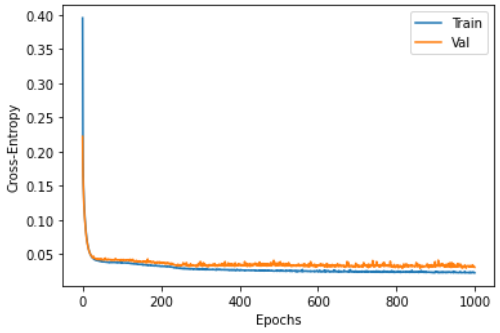
The sequential structure of the model was therefore the following:

- **Dense input layer** (20 units, activation function = 'relu')

- **Dense hidden layer** (15 units, activation function = 'relu')
- **Dense hidden layer** (10 units, activation function = 'relu')
- **Dense hidden layer** (5 units, activation function = 'relu')
- **Dense output layer** (1 units, activation function = 'sigmoid')

The batch size parameter was also set equal to 100 (so, the weights were updated every 100 records). In this way the loss curve reached, at the same time, his minor value (0.041080) and the highest accuracy (0.9894) with respect to the other batch size tried (50, 10).

Anyway, since the performance seemed to be too perfect, our deepest concern was the overfitting. Thus, we check the loss on the training set and on the validation set for the model, trough 1000 epochs.



From this plot it's possible to conclude that there is no overfitting for this model because the loss lines on both set are stable and quite close.

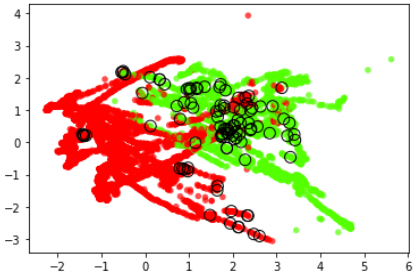
Usually when overfitting is present techniques like Early Stopping (Noreg), Regularization (L2) and Dropouts layer are applied. In this case, they are not needed but, to have a more complete view, they were also tested on the network to see which could be the effect on the performance.

Model	Accuracy	Loss
DNN	0.9894	0.04108
Noreg	0.9888	0.04252
Dropout	0.9883	0.07958
L2 Regularization	0.9866	0.18608

As expected, none of them improve the accuracy or the loss since our model was already well fitted.

2.2 Support Vector Machine

SVM it's a model that represents the decision boundary using a subset of the training examples, known as the support vectors.



In this plot are shown the support vectors in two dimensions for our dataset.

The objective of the classifier is to find the hyperplane that maximize the margin, because decision boundary with largest margin tends to have better generalization error.

Linear SVM

We first implemented the Linear Support Vector Machine, mainly testing different values only for the parameter C, that is the regularization parameter, inversely proportional to the strength of regularization.

After trying 0.01, 0.1, 1, 10, 100, and performing a 10-cross-validation, we got the best performance of the SVM linear classifier with C= 100.

These are the results on the test set:

Accuracy	Precision	Recall	F1	ROC

0.9905	1.00 (0) 0.96 (1)	0.99 (0) 1.00 (1)	0.99 (0) 0.98 (1)	0.999 2

Non-Linear SVM

Then we applied the Non-linear Support Vector Machine algorithm.
We again tried different parameter combinations. In particular, we focused on the kernel function (used to transform the dataset, choosing between radial basis function (RDF), sigmoid function, and polynomial) and on the regularization parameter C (0.01, 0.1, 1, 10, 100).

For each kernel function, we found its best C and then we evaluated it.

RDF had C=0.01:

Accuracy	Precision	Recall	F1	ROC
0.989	1.00 (0) 0.96 (1)	0.99 (0) 1.00 (1)	0.99 (0) 0.98 (1)	0.991

For the **Sigmoid** function was selected
C= 0.001:

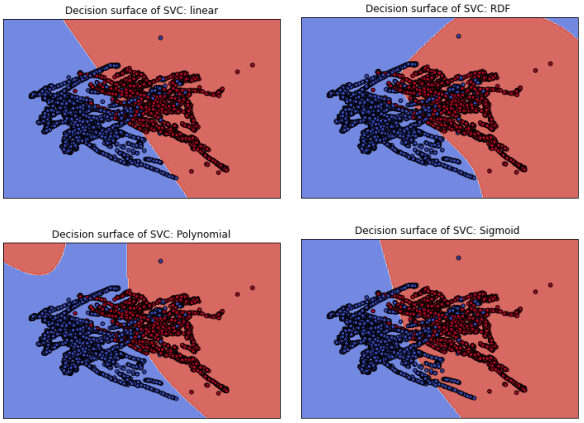
Accuracy	Precision	Recall	F1	ROC
0.958	1.00 (0) 0.85 (1)	0.95 (0) 1.00 (1)	0.97 (0) 0.92 (1)	0.973

Eventually, for the **Polynomial** kernel function
C=0.1 was set:

Accuracy	Precision	Recall	F1	ROC
0.990	1.00 (0) 0.96 (1)	0.99 (0) 1.00 (1)	0.99 (0) 0.98 (1)	0.992

From these results, we can see that the sigmoid kernel function does not give a good performance as the other kernel. The precision in class 1, the one in which we are interested in, is decreased.

This means that the algorithm has misclassified more records as 'Occupied' when they were actually "Not occupied". While the polynomial kernel function is the one that fits better the data, with the higher ROC and cross-validation accuracy/F1.



So, we can affirm that the linear SVM algorithm is more suitable than a non-linear SVM algorithm. This result is also confirmed by the following graph, in which is visible that the data are linearly separable.

2.3 Ensemble Classifier

The different ensemble classifiers we tested on our dataset are Random Forest, Bagging and Boosting. The main characteristic of these methods is that are based on the so-called "Wisdom of the crowd". The assumption is that the collective knowledge of different people can exceeds the knowledge of any single expert. In this case, the metaphor can be applied as unique models against models that use more classifiers of the same type.

The application of these techniques would probably give a better result, by taking in consideration the classification methods previously executed, that have already shown a good performance.

Random Forest

Random forest consists of a large number of individual decision trees that operate as an ensemble.

We applied the random forest algorithm on the same normalized dataset of the previous classification task.

To tune the parameter we executed a random search and a 10-cross validation over:

- **max_depth**: range(2, 20)
- **min_samples_split**: [2,5,10,20,30,50,100]
- **min_samples_leaf**: [1,5,10,20,30,50, 100]
- **criterion**: ['entropy', 'gini']
- **n_estimators**: [25 ,50, 100, 200, 500]

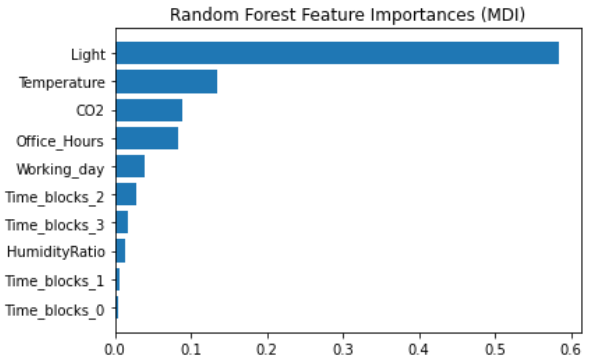
The best model is :

- **n_estimators** = 500
- **criterion** = 'gini'
- **max_depth** = 10
- **min_samples_split** = 5
- **min_samples_leaf** = 5

With the following result on the test:

Accuracy	Precision	Recall	F1	ROC
0.9915	1.00 (0) 0.97 (1)	0.99 (0) 0.99 (1)	0.99 (0) 0.98 (1)	0.9918

Compared to the decision tree, the performance results are better, in particular the precision on class 1, maybe because the model has less false positives.
On the other hand, accuracy and F1 measurements of the cross-validation has decreased.



The feature importance of the random forest shows us how, once again, that Light dominates the construction of the trees, but also that the importance of the other variables, which in the decision tree was almost 0, has increased.
Interesting is the fact that the variables we created, such as office_hours and working_day, have a greater importance than HumidityRatio, an attribute of the original dataset.

Bagging

As second ensemble classifier, we applied Bagging, it randomly select at each iteration, different subset samples of the records, upon which is built the base classifier chosen.

By comparing the different results for each classifiers, before and after the bagging application, we didn't notice big differences, except for the Random Forest, which has a performance decrease, when used as base classifier. We indeed obtained the following results:

Base Classifier	Accuracy	Precision	Recall	F1	ROC
Random Forest	0.970	0.97 (0) 0.97 (1)	0.99 (0) 0.90 (1)	0.98 (0) 0.93 (1)	0.99

Boosting

Boosting is an iterative procedure that adaptively change distribution of the training data, giving more importance to the previously misclassified records, by increasing their weight for the following iteration.
We applied Boosting to different classifiers.

In this case, on the contrary to what we obtain performing bagging, we noticed a slight performance increase using Random Forest as base classifier.
However, it's also remarkable the decrease for the application on the Naive Bayes classifier.

Base Classifier	Accuracy	Precision	Recall	F1	ROC
Naive Bayes	0.747	0.77 (0) 0.12 (1)	0.97 (0) 0.01 (1)	0.85 (0) 0.03 (1)	0.49
Random Forest	0.992	1.00 (0) 0.98 (1)	0.99 (0) 0.99 (1)	1.00 (0) 0.98 (1)	0.99

2.4 Advanced Classifier Conclusion

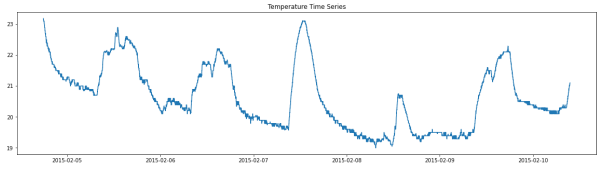
From the results of the advanced classifiers tested in this task, we can claim that these methods improve the classification (except for boosting and bagging), but their application on this dataset is probably unnecessary since the base methods were already very precise. In most of the cases, they also results to be very computationally expensive.

TASK 3 - Time Series

3.1 Time series analysis and classification

Since the dataset provides temporal information (feature "dates"), we exploit it to implement some Time series methods.

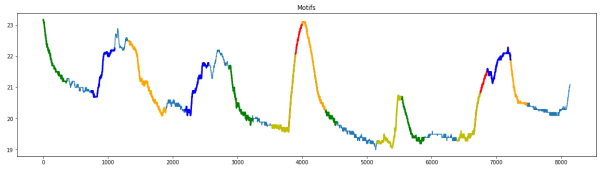
To carry out this task we selected the attribute "Temperature" as our univariate time series, since it has the better temporal shape in comparison with the other.



Also, in order to work with a continuous series, without interruptions, we chose to use only the training dataset.

Motif Discovery and Anomaly Detection

The first analysis computed is the motif discovery, therefore the detection of the repeated pattern in the series.
To find these frequent sub-sequences we used a matrix profile with a time window of 360. The motifs on Temperature are the following.

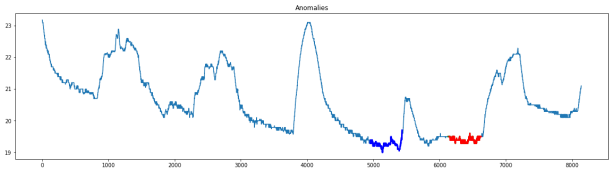


We have extracted the top-5 motif, that are in position:

- [[**3718, 6576**], at a minimum distance of 1.399 from each other
- [**0, 2871, 4185, 5524**], at a minimum distance of 1.969
- [**1317, 1531, 4004, 7111**], at a minimum distance of 2.821
- [**744, 2193, 6865**], at a minimum distance of 3.281
- [**3537, 5169, 6395**]], at a minimum distance of 3.874

These patterns seem connected with the sudden changes, increasing or decreasing, that occur in the series.

With the same methods, but inversely, are also found in position (6132, 4956, 6147, 6162, 6116) the Anomalies, so rare patterns.



These are instead related to the flatten part of the series. Is also interesting their closeness to the Sunday, a free day, in which the office should be empty.

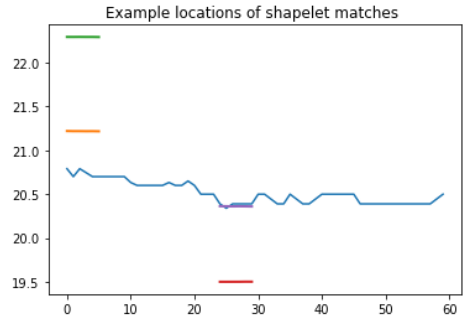
Shapelet

Another kind of patterns are the shapelet, which are indeed the most representative subsequence of a class. Thus, to find them we actually need to do a shape-based classification.

Before, proceeding with the shapelet discovery we decided to divide the original time series into one-hour series, with an offset of 20 minutes between them. So we obtained 405 series of 60 minutes.

After that, we split this series dataset into 70% training and 30% tests to perform the classification, considering Occupancy as the class once again.

Setting as parameter of our model: optimizer="sgd", weight_regularizer=.01 and max_iter=200, we detected four shapelet with a dimension of 6.
The correct classification rate on the test set is equal to 0.828.



From the shapelet found we can see how the hourly series of temperatures can be mainly divided in relation with these four levels of values.

Univariate Classification

On this same univariate time series dataset, we also applied other classifiers algorithm (structural-based, feature-based, distance-based, CNN and LSTM), aiming to label the records as 0 or 1 for the class Occupancy.
The following are the most relevant results on the test set obtained:

Classifier	Accuracy	Precision	Recall	F1
LSTM	0.803	0.80 (0) 0.00 (1)	1.00 (0) 0.00 (1)	0.89 (0) 0.00 (1)
CNN	0.844	0.98 (0) 0.56 (1)	0.83 (0) 0.92 (1)	0.90 (0) 0.70 (1)
KNN (metric DTW)	0.959	0.98 (0) 0.88 (1)	0.97 (0) 0.92 (1)	0.97 (0) 0.90 (1)
Shapelet Distance-Based Classifier (KNN)	0.819	0.92 (0) 0.53 (1)	0.85 (0) 0.71 (1)	0.88 (0) 0.61 (1)

The first result is given by the LSTM model. As we can see it is the worst since it is not able to predict any of the class 1 records. So, there is a clear overfitting in this case.

The LSTM model structure that we use is the following:

- **LSTM** layer of size 20
- **Dense** layer of size 10
- (**output layer**) Dense layer of size 2

While the **CNN** model structure (with dropout layers of 0.2) is:

- 1-dimensional convolutional layer with **20** filters of size **10**
- 1-dimensional convolutional layer with **35** filters of size **6**
- 1-dimensional convolutional layer with **50** filters of size **4**
- **global pooling 50**
- (**output layer**) Dense of size 2

The CNN has a better performance than LSTM but it's not the best for the univariate problem, the in particular precision results to be unsatisfying.

The Shapelet Distance Based classifier has a similar performance to the one of LSTM and CNN, indeed it has a poor result in class 1.

Therefore, the distance-based classifier K-NN provided by the *pyts* library especially for the time series (for which DTW has been set as distance metric) is the best model, the only that predicts well in both classes (situation that is reflected in the higher accuracy obtained).

Multivariate Classification

For the multivariate classification, we considered the entire time series dataset (not only the attribute "Temperature"), to which LSTM and CNN have been applied. The results obtained are the following:

Base Classifier	Accuracy	Precision	Recall	F1
LSTM	0.816	0.81 (0) 0.96 (1)	1.00 (0) 0.24 (1)	0.89 (0) 0.39 (1)
CNN	0.884	0.87 (0) 1.00 (1)	1.00 (0) 0.52 (1)	0.93 (0) 0.68 (1)
CNN2	0.961	0.95 (0) 0.99 (1)	1.00 (0) 0.85 (1)	0.98 (0) 0.91 (1)

From the performance results, we can see that the LSTM classifier used is better than the previous one applied on the univariate series. Even if the values of the Recall and F1 are still low for class 1, at least are not all equal to 0.

We also implement two different CNN models, which differ from each other in some parameters. The second one performs better reaching higher values for recall and F1 (with recall lower than the univariate CNN classifier), and so, a higher accuracy.
Its model structure is:

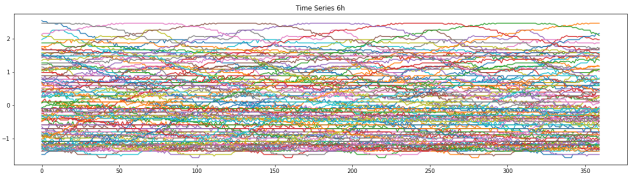
- 1-dimensional convolutional layer with **16** filters of size **4**
- 1-dimensional convolutional layer with **32** filters of size **2**
- 1-dimensional convolutional layer with **64** filters of size **1**.

As you can see from the values above, the last configuration of the model is the most suitable among the three taken into consideration.

3.2 Time Series Clustering

Before starting the clustering task, we take again the original "Temperature" time series and, in order to get better results, different transformation techniques, such as Amplitude Scaling, Trend removal and Smoothing, to normalize and remove trend and noise from the series, have been applied.

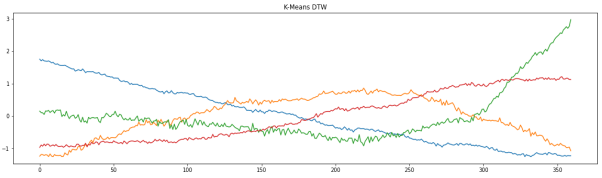
Then, we split the series into groups of 6 hours, with an offset of 60 minutes, obtaining a dataset of 130 time series. In this way the series, being longer, are more able to capture the temperature excursion, that in the previous one hour dataset is almost absent (the one hour series tend to be linear because the considered period is too short).



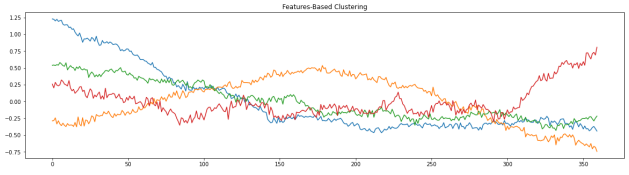
To analyze the similarity of the created time series we applied different clustering techniques:

- **shape - based** (with Euclidean and DTW distance)
- **feature - based**
- **compression - based**
- **approximated - based**

For all of these methods, the idea is to identify four clusters, which referred to the four moments of the day that were already identified with the time_blocks variable at the beginning of the task 1.
The worst performance is obtained with the compression-based method clustering, because it found one unique cluster and therefore was useless to our scope. On the other hand, all the other methods give us the expected outcome.

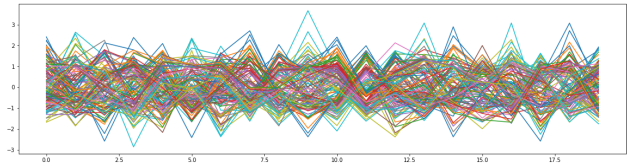


The **Shape-Based Clustering** obtained applying the K - means performs better with DTW set as distance metric. The SSE for it is equal to 22.23 (while the Euclidean has SSE 115.184, which is too high).

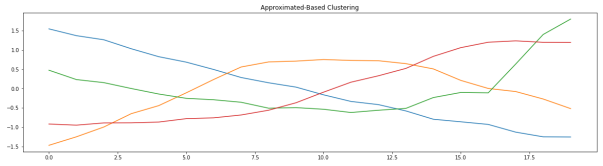


To perform the **Feature – Based Clustering** metrics like mean, variance, percentiles are extracted from the series, and the clusters found are based on them. The SSE is 40.051.

Also, the **Approximated Based Clustering** give good result. The better is found using the Piecewise Aggregate Approximation (PAA), set with 20 segments. Here in the plot is shown the approximated series.



The clusters obtained are the following, for which the SSE is 5.4724, the lowest one, so we can conclude that this method is the best.



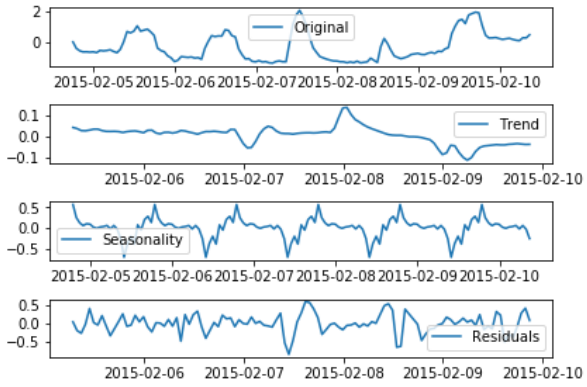
All these methods always identified four cluster as we wanted, with four different behaviour: ascending, descending, convex and concave. They probably reflect the three times of the day, morning (ascending), late-morning/afternoon (concave), late afternoon/evening (descending) and night (convex).

3.3 Time Series Forecasting

In this section, we apply several forecasting methods on our dataset by comparing them in order to find the best one. Firstly, we adjust the frequency of the index of our time series, by choosing a 1 minute frequency, because the records have been registered with this time regularity.

Consequently, we made the decomposition of the time series, that model separately trend and seasonality and returns the

residual part of the Time Series, in order to identify them.



The presence of these components can affect the forecasting. So, some transformations are needed to remove them.

We tried different transformation methods, such as Log, Diff, Log Mean Diff and Untrended series, and then a Dickey-Fuller Test has been performed for all of them, to discover which results in the most stationary series, since without a stationary time series the forecasting is not efficient.

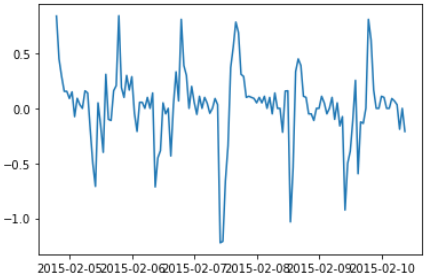
By comparing the test results on the time series before and after the transformations, we noticed that the Diff method has a really good p-value of 3.669680e-09 and a Test statistic less than all critical values:

Test Statistic	-6.711557e+00
p-value	3.669680e-09
#Lags Used	0.000000
Number of Observations	1.340000e+02
Critical Value (1%)	-3.480119e+00
Critical Value (5%)	-2.883362e+00
Critical Value (10%)	-2.578407e+00

While the results of **Dickey-Fuller Test** on the time series not transformed is:

Test Statistic	-3.543381
p-value	0.006940
#Lags Used	1.000000
Number of Observations	134.000000
Critical Value (1%)	-3.480119
Critical Value (5%)	-2.883362
Critical Value (10%)	-2.578407

So, the series with the Diff transformation, shown in the plot, has been selected among the others and used for the forecasting.

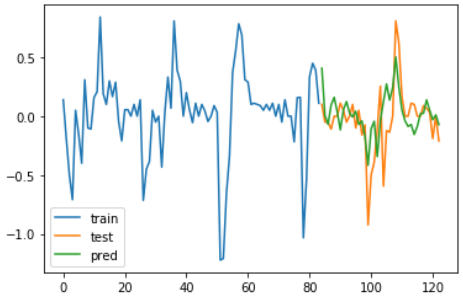


The forecasting methods implemented are Holt, ARIMA, Simple Exponential Smoothing and SARIMAX.

The series was before temporally split in training and test set in order to do the prediction on the test set.

The best technique of prediction results to be SARIMAX, setting the following parameters:

- **order**=(1,0,3),
- **seasonal_order**=(8, 0, 2, 8)).



The performance evaluation reports that the errors are near to zero and R^2 = 0.278. These metrics are not really good, but they are our best ones. Also by looking at the plot it is possible to see that the prediction doesn't perfectly match with the test set.

TASK 4 - Sequential Pattern Mining

The report continues with the pattern analysis, that differs compared to the simpler "pattern mining" because it exploits the temporal information which although would be lost.

The goal of this task is to find the most frequent sequential patterns of at least length 3 using the time series created to execute the clustering, so 130 series of 6 hours.

In order to execute the sequential pattern mining algorithm, was necessary to transform the original series and the series of 6 hours by applying the SAX algorithm.
This algorithm allowed us to map the value of the time series with symbol, by dividing it in 20 segment (18 minutes long) to which numbers from 0 to 9 have been associated. In this way, we obtained a dataset of segments.

To find the motifs of the several sequences, we used the PrefixSpan algorithm. The sequences searched have minimum length equal to 3, in order to obtain subsequences of at least about 1 hour.

Setting a threshold of 97% for the support we extracted the following patterns:

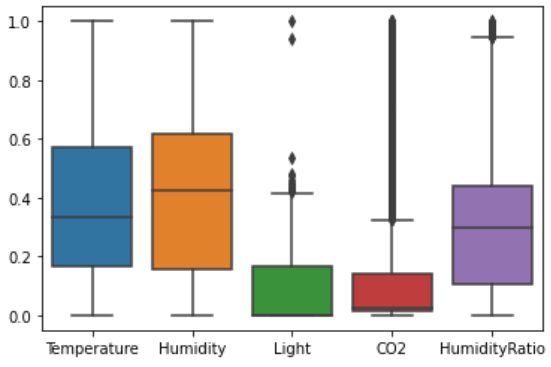
Sequential Patterns	
[9, 2, 1] [9, 2, 1, 5] [9, 2, 1, 5, 5] [9, 2, 9] [9, 2, 9, 5] [9, 2, 9, 5, 5] [9, 2, 5] [9, 2, 5, 5] [9, 1, 1] [9, 1, 1, 5] [9, 1, 1, 5, 5] [9, 1, 5] [9, 1, 5, 5] [9, 9, 1] [9, 9, 1, 1]	[9, 9, 1, 1, 5, 5] [9, 9, 1, 1, 5, 5] [9, 9, 1, 5] [9, 9, 1, 5, 5] [9, 9, 5] [9, 9, 5, 5] [9, 5, 5] [2, 1, 5] [2, 1, 5, 5] [2, 9, 5] [2, 9, 5, 5] [2, 5, 5] [1, 1, 5] [1, 1, 5, 5] [1, 5, 5]

It is interesting to notice how the sequences contain more or less the same elements. Indeed, it is possible to see that the more relevant and frequent items are the "9", "1" and "5", which are even often repeated within a same sequence.

This repetition of the same elements can be explained by supposing that the Temperature inside a room (occupied or not) it is barely subjected to big swings of values, but instead it has a consolidated mean over time.

TASK 5 - Outlier Detection

To identify the anomalies within our dataset we applied different methods.
The first one was the simplest. We plot the boxplot in order to have a visualization of the outliers for each feature.



We can see that the features with a significant number of anomalies are “Light”, “CO2” and “HumidityRatio”.

Through the boxplot we identified the top 1% outliers computing an interquartile range from 0 to 0.99 and subtracting this to 1 to find the remaining 1%.

With this method we found only 2 outlier for the “Light” attribute.

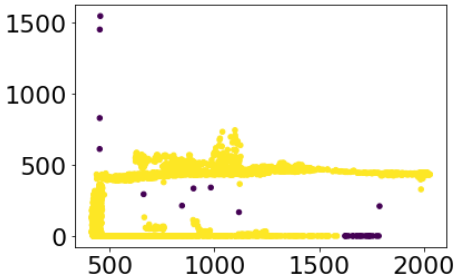
In order to continue our anomaly detection, we adopted five different methods: DBSCAN, LOF, ABOD, KNN, LOF(clustering-based)

Density-Based Approach

We applied the DBSCAN as Density-based approach. The idea is to find the anomalies comparing the different densities of the points with the density of their local neighbors. We identify an outlier if its density is really different from the one of its local neighbor, thus considering the noise points as anomalies.

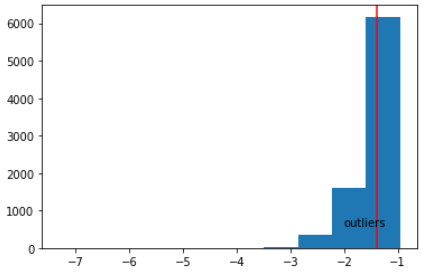
We tried different values of k selecting k = 20 as the best one since it evaluates better the difference between anomalies and normal points.

The result was the following:



As it’s possible to see from the plot, the DBSCAN identify also some normal points as anomalies. This is because of the density problem related to DBSCAN, which is not able to handle different density depending on the considered regions.

This is why we also applied another Density - based method called Local Outlier Factor (LOF). This method computes the distance between every point with its k-nearest neighbors. The value obtained by this computation is then used to estimate the local density around the point. For this reason, points with local density significantly less than the local density of their neighbors, will be classified as outliers.
In particular, points that obtain a score value near to 1 will be classified as normal, while points that obtain a score value increasingly greater than 1 will be classified as outliers.
Since we only have available the outlierness factor, the opposite of the local outlier factor, the points with a score value near to -1 are classified as normal while the points with a score value increasingly lower than -1 are classified as outliers.
By applying this method 1584 outliers have been detected. The minimum outlier score is equal to -1.5, while the greater outlier score value is equal to -67666714.



Later the LOF was also applied using the distance between records, as a simple distance-based approach. In this

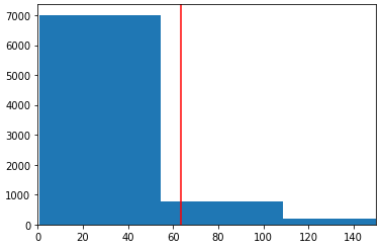
circumstance, the method identified 672 outliers.

Angle-Based Approach

Another outlier detection method that we use is ABOD (Angle Based Outlier Degree). This kind of method is used for high dimensional dataset since it uses the spectrum of pairwise angles between points, which is a more robust distance respect to traditional ones. Considering the angle between any two instances, it detect as outliers the points with a highly fluctuating spectrum. By applying this outlier detection approach we detected 781 anomalies.

Distance-Based Approach

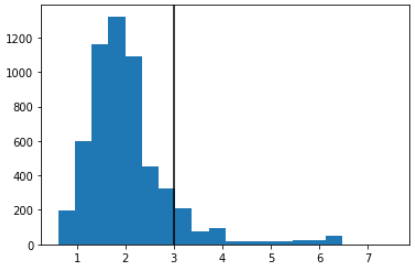
Another distance-based method implemented is K-NN, which uses as decision score the distance from its k-nearest neighbour. We set k = 100 since it gives us a better distribution. Applying this approach we detected 807 points as outliers.

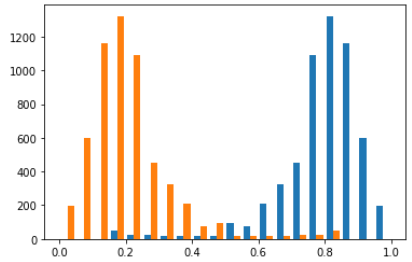


Autoencoders

The last method that we implemented is the Autoencoder, which is an approach based on the Neural Networks model. In fact, it represents each record through the compression and decompression of a Deep Neural Network.
This kind of technique is able to detect the outliers points by considering the reconstruction error of the DNN. It labels as outliers points that generate a large value for such error since they're more far away from the training points of the Autoencoder.

The Autoencoder used on this dataset is made of 4 hidden layers composed by 5, 8, 8, and 5 units. After 50 epochs, it detects 570 outliers.





Conclusion of outliers

The several approaches previously seen, has certainly helped us to identify the possible anomalies presented in our dataset, even if their results are basically the same.

Indeed, the outlier points detected are between 500 and 800, except for the LOF, used as Density-Based method, which detected 1584 outliers and the DBSCAN which detected only 2 anomalies.

The features with the most outliers detected are Light and CO2, which have similar behaviors, while the other ones do not present many anomalies as we saw so far.

The result obtained from all the precedent methods used to detect the anomalies in our dataset are:

METHOD	OUTLIERS
DBSCAN	2
LOF	1584
ABOD	781
KNN	807
LOF (clustering based)	672
Autoencoders	570