Data Mining II project

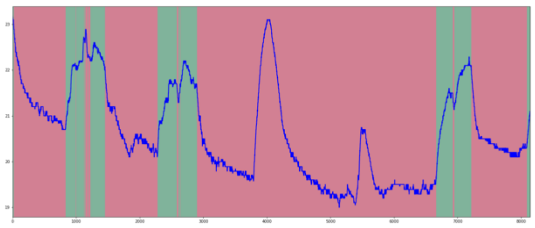
Occupancy Detection

Academic year 2019/2020

Attributes evaluation and modifications

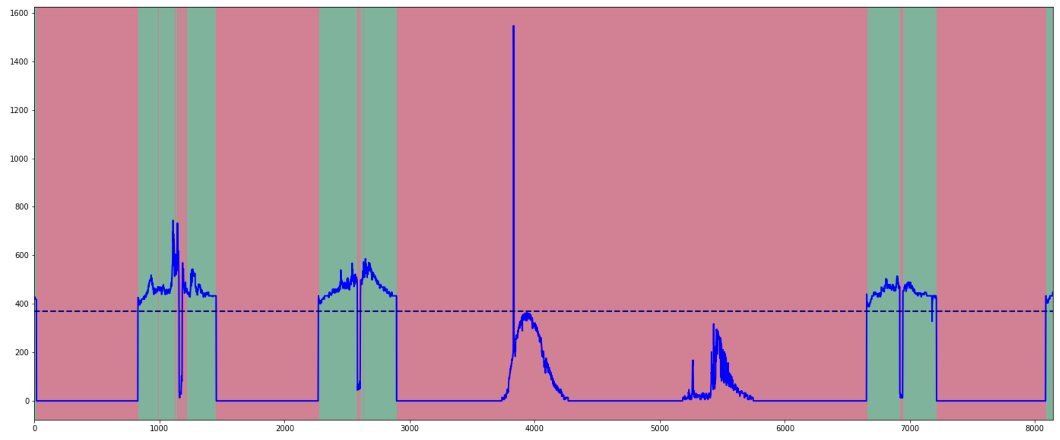
Our dataset is composed by the following attributes:

- **id**: just a numerical counter of the rows, we delete it because it is useless for our aim.
- **Date**: day/month/year/hour/minute format, this attribute shows the day, the month, the year, the hour and the minutes when the data were taken. We have done some transformations about this feature. First of all, we have divided the feature in two variables, the first contain the date in a format YYYY-MM-DD and the second contain only the hour, so we have discarded the minutes.



1-Temperature time-series plot

- **Temperature**: in Celsius grades. From the graph on the side we can see the part of time series which correspond to the office occupied colored in green while the red part corresponds to the office not occupied. We can easily see that we don't have a clear boundary between occupied and not occupied.
- **Humidity**: percentage of the water in the hair. As for temperature, don't show us a clear boundary, so we haven't done any transformation on this variable.
- **Humidity Ratio**: kgwater-vapor/kg-air, derived quantity between temperature and humidity. Since it shows the same thing of humidity, we can delete this attribute. In fact, the similarity between Humidity and Humidity Ratio is 0,955198 which is very high.
- **Light**, in Lux. It shows the quantity of light inside the office. From the dataset it seem that this light is a sum of the artificial lights in the office and the sunlight from outside, because we see that the light start at the open hour and grow up during the morning until 12, and after decrease until it fall to 0 at the closing of the office, maybe due to the turning off of the artificial lights and to the closing of the blinds so that no more sunlight can enter. We know that when the light is 0, the office is always not occupied. As we can see from the graph behind, in that case we can see an almost perfect separation between occupied and not occupied based on the value of Light. A good boundary is 370. So, we think that the amount of lux is not so important for what concern the occupancy of the office, thus we change this attribute to a Boolean attribute: it is going to take 0 as value if the light is less than 370, and 1 if the lux are greater or equal than 370.



2- Light time-series plot

- **CO2**, in ppm. It shows how much part per millions of CO2 there are in the office. As for 'Temperature' and 'Humidity' from the graph we can't see a clear boundary like we have seen for 'Light' so also in that case we have decide to leave the variable unchanged.

Creation of new variables

- Variable "**date**": a new attribute that show only the date in YYYY-MM-DD.
- Variable "**Hour**": a new attribute that show only the hour without the minutes.
- Variable "**DayName**": a new attribute which show the name of the day starting from the new variable 'date'.
- Variable "**IsWorkDay**": a new Boolean attribute which show if the day is a working day or an off day, this variable takes 1 in the first case and 0 in the last. This variable is useful for our goal by the fact that when 'IsWorkDay' is equal to zero so the office will be always empty, thus 'Occupancy' is equal to 0.
- Variable "**IsWorkHour**": a new attribute that link the occupancy whit the hours. If the hour is a workhour it takes 1, if the hour is a non-work hour it takes 0 and if the hour is a boundary hour it takes 2. The hours 7, 13

and 18 are boundary hours because the office open/close in the middle of them and so we don't know if it is occupied or not in those hours. From 18 to 7 it is a non-work hour. From 7 to 12 and from 14 to 17 it is a workhour, but only if the variable ‘IsWorkDay’ is 1.

Classification task

In this task we are going to use different classifier in order to find the better one between:

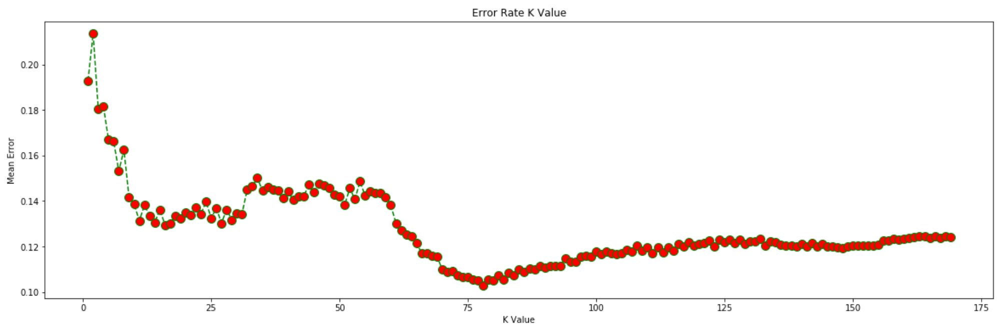
1) K-NN
2) Naive-Bayes
3) Logistic Regression
4) Decision Tree

For train the classifier we have used the dataset called “*datatraining.csv*”, then the classifier will be tested using the files called “*datatest.csv*” and “*datatest2.csv*”. For all the datasets we have applied the same preparation in order to have the same shape of the dataset. The attributes “date” and “DayName” are dropped before the application of the algorithms because for our aim in these tasks are useless.

KNN

The biggest problem with the application of KNN classification algorithm is to find the best K.
So, first of all, we have tried starting with fewer number increasing randomly up to 158 (which is the square of N, $\sqrt{N}$).
In all the tables behind are showed some different results using different value of K on the “*datatest.csv*”.

k=5	
accuracy	0,833020
k=25	
accuracy	0,867542
k=100	
accuracy	0,882176
k=158	
accuracy	0,876547



3- Error rate K value

As we see from the tables, the results of the different classifier are pretty good, but our aim is to find the best K in order to build the best classifier, so we have decided to plot the error of each KNN classifier using a value for K included between 1 and 170.

From the chart we can see that the K that minimize the mean error is 78, so, running again the KNN algorithm with K=78 we have obtained the best result.

k=78	Accuracy
accuracy	0,897185

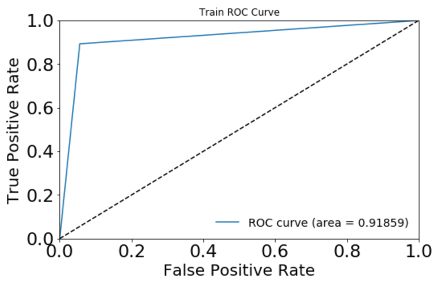
The problem come out when we go for apply the same classifier to the “*datatest2.csv*”, in that case the for example, the accuracy is equal to 0.56, so we had to change something. Using the GridSearch we went to see the new best K crossing the three different datasets. The new best value of K obtained from the grid search is 81. And these are the new classification reports for the training dataset and for the two test datasets:

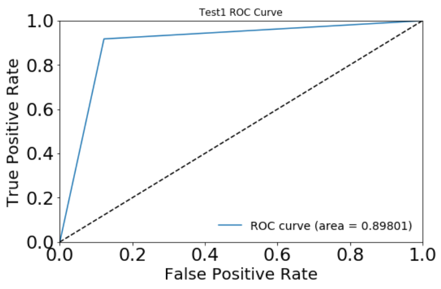
```
Train Classification Report:

Accuracy 0.9333169593515903

              precision    recall  f1-score   support

           0       0.97      0.94      0.96      6414
           1       0.81      0.89      0.85      1729

    accuracy                           0.93      8143
   macro avg       0.89      0.92      0.90      8143
weighted avg       0.94      0.93      0.93      8143


Train Confusion Matrix:
[[6056  358]
 [ 185 1544]]
```

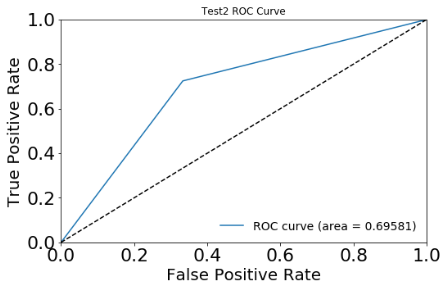
```
Test1 Classification Report:

Accuracy 0.8926829268292683

              precision    recall  f1-score   support

           0       0.95      0.88      0.91      1693
           1       0.81      0.92      0.86       972

    accuracy                           0.89      2665
   macro avg       0.88      0.90      0.89      2665
weighted avg       0.90      0.89      0.89      2665


Test1 Confusion Matrix:
[[1487  206]
 [  80  892]]
```

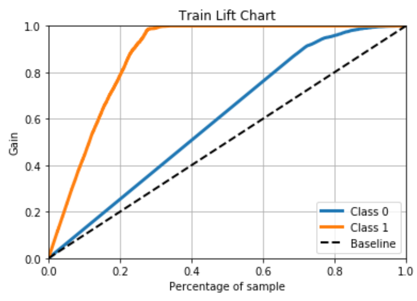
```
Test2 Classification Report:

Accuracy 0.6790401968826907

              precision    recall  f1-score   support

           0       0.90      0.67      0.77      7703
           1       0.37      0.72      0.49      2049

    accuracy                           0.68      9752
   macro avg       0.63      0.70      0.63      9752
weighted avg       0.79      0.68      0.71      9752


Test2 Confusion Matrix:
[[5137 2566]
 [ 564 1485]]
```

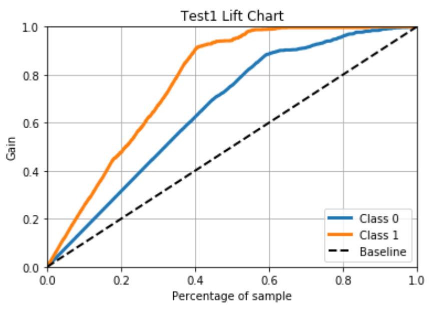
Then we can see that on the first test dataset we lost some accuracy, but now we have a better performance in overall, even if, with second testset we have a low value for accuracy and so this classifier goes in overfitting on this dataset.
And here we can see the ROC curve and Lift Chart for each classification report.

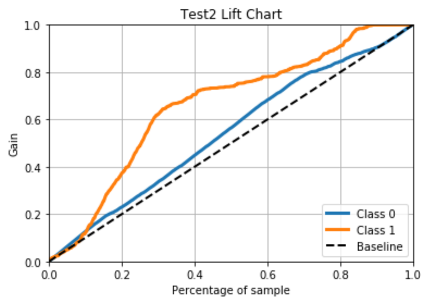












4- ROC Curve and Lift Chart for each dataset usign KNN

Naive-Bayes

Using Naive-Bayes as classifier, we have obtained a good value for both class value and for both the test dataset. Here we can see the results for "*datatest.csv*":

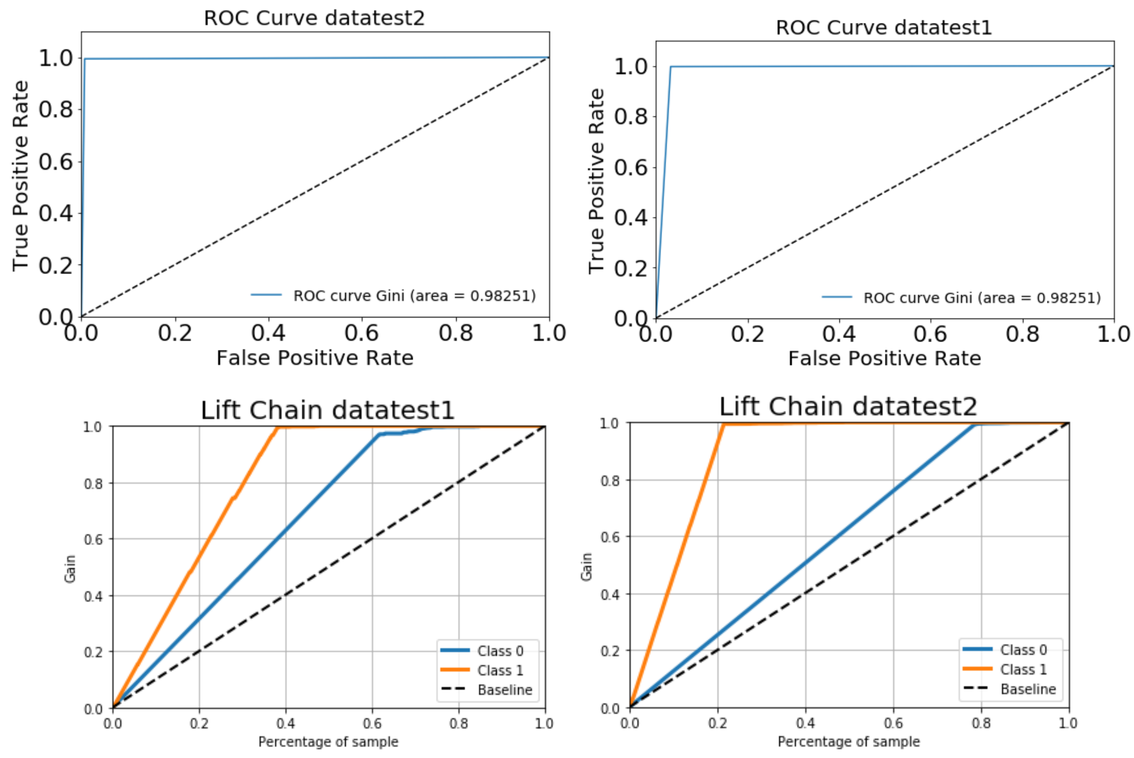
Test1	Predicted Occupancy=0	Predicted Occupancy=1
Occupancy=0	1639	54
Occupancy=1	3	969

Test1	f1-score	support	accuracy
0	0,982908	1693	**0,978611**
1	0,971428	972	

Test2	f1-score	support	accuracy
0	0,995704	1693	**0,993232**
1	0,984057	972	

Test2	Predicted Occupancy=0	Predicted Occupancy=1
Occupancy=0	7649	54
Occupancy=1	12	2037

Calculating the Roc value for this classifier we see that also this value is very good, almost perfect:



5- ROC Curve and Lift Chart Naive-Bayes

Naïve-Bayes classifier give us a good performance for both the test dataset, also the overfitting and the underfitting are avoided. In fact, as we can see from the tables the accuracy for the test datasets is very good and, trying the algorithm also on the train dataset we obtain very good values, accuracy=0,988210.

Logistic regression

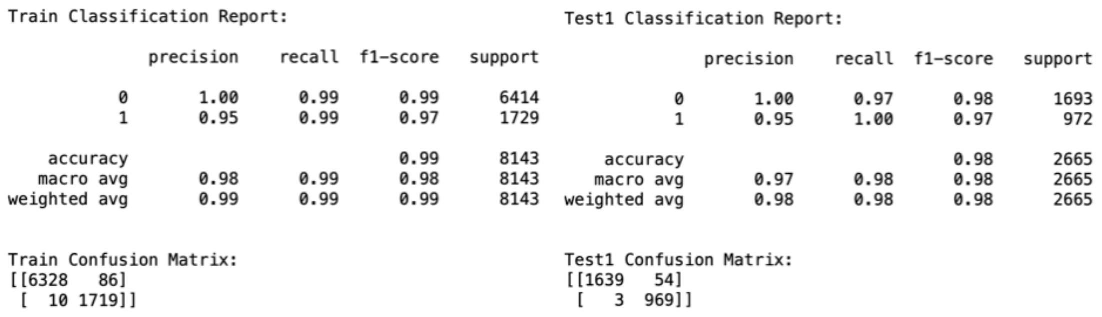
Here we went to try the logistic regression as a classifier for our dataset. First, we have tried with all the attributes in order to understand if it is a good way to classify the whole dataset. Then, choosing just one continuous attribute with the class, we have plotted the logistic regression combined with the linear regression for visualize how the regression works in two dimensions. For the logistic regression we don't show the result of the second datatest, in fact the results are very similar, and the parameters viewed in GridSearch part give us good results for both the test datasets. The values obtained with all the attributes are:

	precision	recall	f1-score	support	accuracy
0	1,00	0,97	0,982908	1693	**0,978611**
1	0,95	1,00	0,971428	972	

These values are obtained using the function LogisticRegression() without any parameters, so, in order to find the best parameters combination we have done the grid search for parameters estimations.
The parameters that we have checked are:

- C, range= [1, 1.2, 1.5]
- Penalty, range= ['l1', 'l2']
- Solver, range= ['liblinear', 'saga']
- Tol, range= [1e-3, 1e-4]

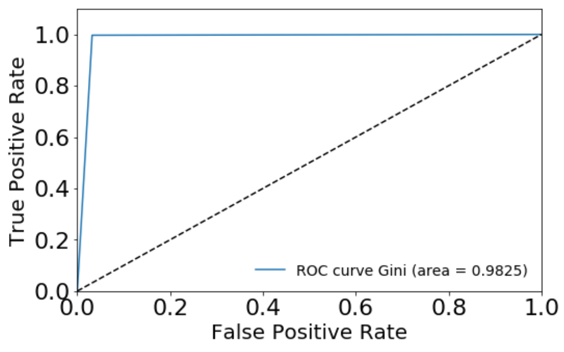
After the grid search the best parameters obtained are:



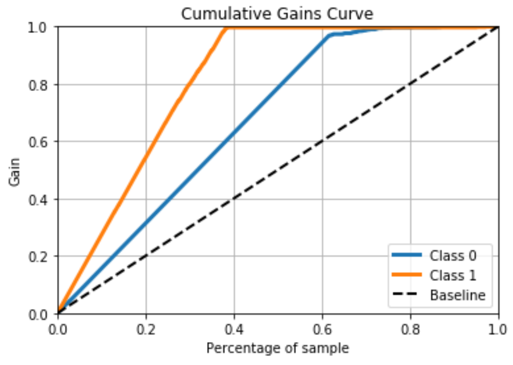
```
Train Classification Report:                       Test1 Classification Report:

              precision    recall  f1-score   support              precision    recall  f1-score   support

           0       1.00      0.99      0.99      6414           0       1.00      0.97      0.98      1693
           1       0.95      0.99      0.97      1729           1       0.95      1.00      0.97       972

    accuracy                           0.99      8143    accuracy                           0.98      2665
   macro avg       0.98      0.99      0.98      8143   macro avg       0.97      0.98      0.98      2665
weighted avg       0.99      0.99      0.99      8143 weighted avg       0.98      0.98      0.98      2665


Train Confusion Matrix:                            Test1 Confusion Matrix:
[[6328   86]                                       [[1639   54]
 [  10 1719]]                                       [   3  969]]
```

So, as we can see, the classification report from the test dataset is equal to the previous one, the one without parameters. An interesting thing discovered now is the good quality of the classifier also for the training set which means that both underfitting and overfitting are avoided in that case.

The Roc Curve and the Lift Chart for this classifier are showed here:

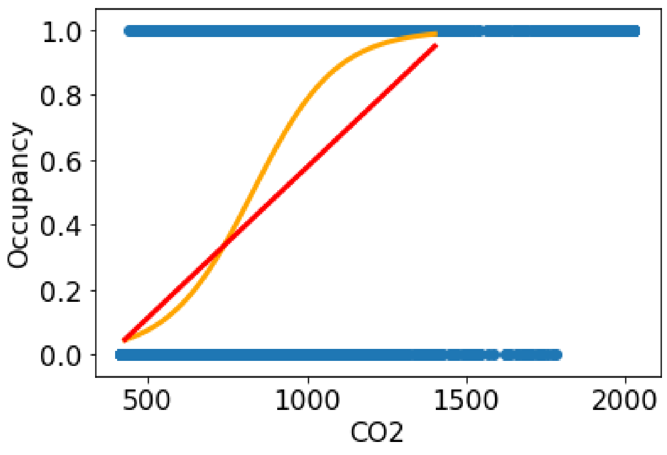


6- ROC curve Logistic Regression classifier



7-Lift Chart Logistic Regression

After that, we wanted to see how the logistic regression work, thus we have chosen the continuous attribute 'CO2' (which is the one that sowed us better the regression), and we have plotted the graph in relation with the class ('Occupancy').



From the graph behind we can see the different regressions. The red one show us the linear regression and yellow one the logistic regression. The logistic clearly approximate better the shape of the plot, and, as showed in the graph *8- CO2 time-series plot* represent also a good approximation.

8- CO2 Logistic and Linear Regression

Decision Tree

In this part about the decision tree we have run the algorithm using all the attributes, the attributes reduction will be subject of analyzes of the next point. First of all, using the grid search crossing the three datasets, we have checked the best value for different parameters. After the grid search the optimal parameters are:

- max_depth = 3
- min_samples_split = 2
- criterion, range = entropy

Test1	f1-score	support	accuracy
0	0,969911	1693	**0,961726**
1	0,947422	972	

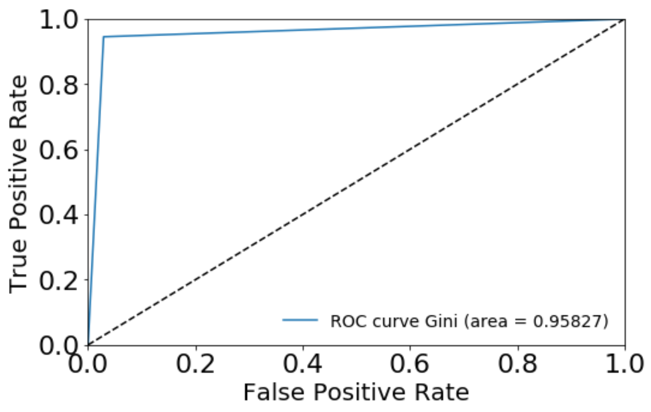
Test1	Predicted Occupancy=0	Predicted Occupancy=1
Occupancy=0	1644	49
Occupancy=1	53	919

Test2	f1-score	support	accuracy
0	0,991436	7703	0,990976

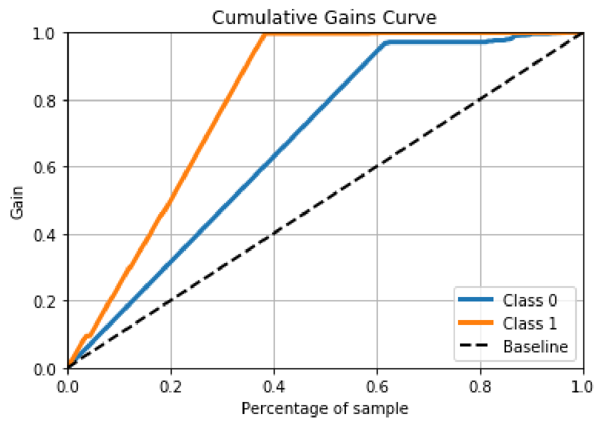
1	0,982096	2049	

Test2	Predicted Occupancy=0	Predicted Occupancy=1
Occupancy=0	7659	44
Occupancy=1	44	2005

In the reality, the best max_depth would be 1, but, since the all classification in this way would be based on the value of 'Light' this could be a little problematic in trivial case that are a little different from the one that we have, so we have decided to start the value from 3. The Roc Curve and the Lift Chart for the Decision Tree classifier are:



9- *ROC Curve Decision Tree for datatest1*



10-Lift Chart Decision Tree for datatest1

The decision tree is a very good classifier in our case, the values of 'Light' do the majority of the work because of the big similarity between this variable and the class 'Occupancy'. Overfitting and underfitting are avoided, the classifier works well also for training set (accuracy=0,99). Also, the ROC value is very high in all the cases, in fact we have 0,95 for the first testset, 0,98 for the second and 0,99 for the training set.

Attributes Reduction

In this part our goal is to try to reduce the number of attributes of the datasets in order to have a smaller datasets which will be lighter to classify. Another thing analyzed in this task is PCA algorithm in order to plot the dataset in two dimensions. First of all, we know that our dataset is made by 8143 rows and 7 columns (the number of columns is already calculated without the attribute 'date' and 'DayName').

Variance Threshold

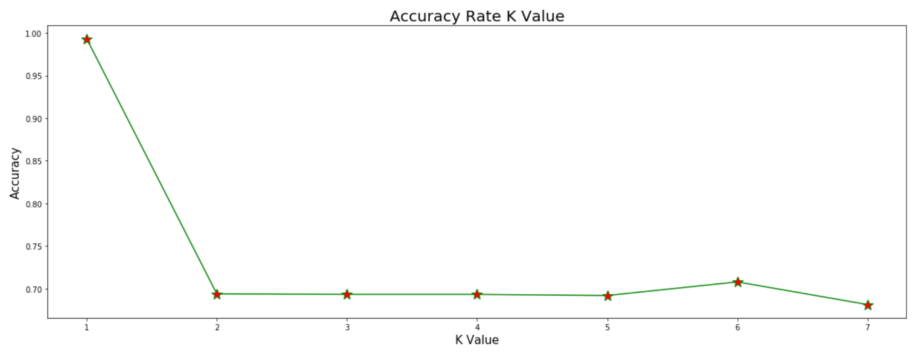
The first algorithm used for reducing the number of attributes in the dataset, this method is based on the elimination of the features whose variance doesn't meet a threshold that we set. For example, for the Decision Tree and the Naïve-Bayes we don't have any variance threshold that increase our accuracy instead for these classifiers we loss accuracy. Here are some instances with comparison between the accuracy and the threshold:

Threshold	Attributes	Test 1	Test 2
0	7	Naïve-Bayes: 0,978611 Decision Tree: 0,961726	Naïve-Bayes: 0,993232 Decision Tree: 0,990976
$\frac{0,3}{1-0,3}$	6	Naïve-Bayes: 0,918574 Decision Tree: 0,960225	Naïve-Bayes: 0,724056 Decision Tree: 0,603560
$\frac{0,5}{1-0,5}$	5	Naïve-Bayes: 0,904690 Decision Tree: 0 960225	Naïve-Bayes: 0,960225 Decision Tree: 0,643560

Thus, in our case the variance threshold is useless for these two classifiers. We can't say the same for KNN classifier. If we remember, the accuracy for the second datatest using KNN wasn't a good value (0,679040), but reducing the dimension of the dataset to 5 columns, so using a threshold equal to $\frac{0,5}{1-0,5}$ we can see a little increment in the accuracy for the second test dataset obtaining an accuracy equal to 0,680578. However, in the overall, the variance threshold features selection doesn't give us a good performance, the increment in the KNN is very small and for the other classifiers tested the effect gave form this reduction decreases the accuracy.

Univariate feature selection

The univariate feature selection is based on the selection of the best K feature of the dataset, so we have to find the best K in order to maximize the accuracy of our classifiers. As we remember, the worst classifier viewed is the KNN applied on second test dataset, so first of all, we have searched a good value for K in such a way to increase the accuracy of this classifier for this dataset. So, we have plotted the accuracy at the variation of the K.



11- *Accuracy rate K value*

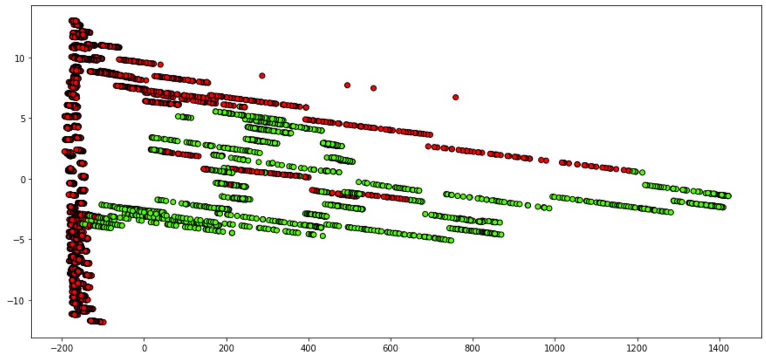
As we can from the graph above, for this case the best value of K is 1, so in order to see if this value give us good result also for the other classifiers, we have tried this value for each algorithm. The results with K=1 are very good for every classifier, for simplicity we show behind only the value of the accuracy:

Algorithm	**Previous accuracy**	**Actual accuracy**
KNN test 1	0,892682	**0,978611**
KNN test 2	0,679040	**0,993232**
Naïve-Bayes test 1	0,978611	**0,978611**
Naïve-Bayes test 2	0,993232	**0,993232**
Decision Tree test 1	0,961726	**0,986587**
Decision Tree test 2	0,990976	**0,998014**

With K=1 the accuracy increases in every case, especially for the case of KNN on datatest2 that was the worst before. The problem could be that with K equal to 1 we don't have many attributes in our dataset, just one, so in special cases these classifiers could do a wrong prediction.

PCA

We have used PCA in such a way to reduce the dataset in 2 dimensions (using the parameters of the PCA function n_components=2) and plot it for see the distribution. Then, on this reduced dataset we have tried the decision tree classifiers for see the results and the decision boundary. So, first of all, reducing the dataset in two dimension we have obtained this shape:

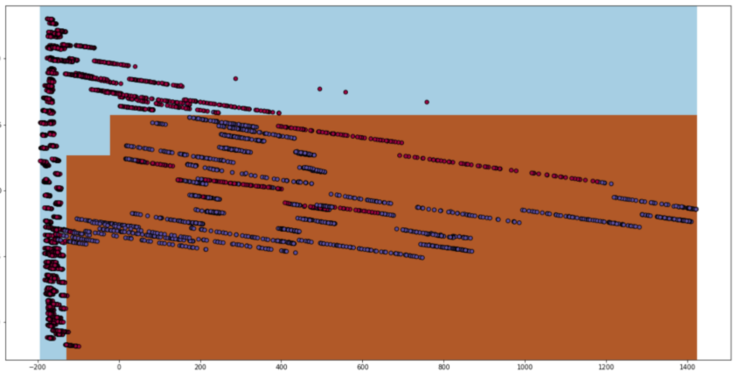


12- *Two dimensions dataset reduced with PCA algorithm*

Then we tried the decision tree algorithm for see how this classifier works on this type of reduced dataset. For simplicity, the test dataset used in this case is the *"datatest.csv",* so after reducing also the test dataset in two dimensions using PCA for fit that with the classifier we ran the decision tree algorithm obtaining these results:

	precision	recall	f1-score	support	accuracy
0	0,99	0,79	0,881100	1693	**0,863789**
1	0,73	0,98	0,840579	972	

In that case with the PCA reduced dataset we can see that we lose some accuracy form the original dataset used (that was 0,961726) and also the other classification report doesn't improve. Thus, it is useless to use this type of reducing algorithm on this dataset. The interesting thing with the PCA is the possibility to plot the dataset in two dimensions as we have seen in the picture [12]. In this way it is also possible to plot the dataset and the decision boundary of the classifier chosen. So, in the next graph we can see the decision boundary of the decision tree on the two dimensions dataset. The blue part is dedicated to the 'Occupancy=0', in this part it is clear that the prediction is pretty good, only 15 false negatives founded by the algorithm out of 1693 instances, and this is proved that the value of the precision for 0 which is 0,99. The problem come out in the orange part. There we can observe that we have a lot of misclassified instances, 348 false positive founded out of 972 instances. These instances that have been misclassified are easily identified in the graph (the red circle in the orange part). So, plotting the dataset in this way is easy to see where we can have some problem in the classification.



13- *Decision boundary of the Decision Tree*

Imbalanced data

This task is about the study of our classifier in case where the dataset is imbalanced, namely where the distribution of the class inside it is strongly unequal. All of these steps will be documented with graph of the dataset plotted in two dimensions using PCA in such a way to see how the shape of the dataset change from balanced and imbalanced and how the different algorithm work during the resample of the minority class. The first thing to do for start the task is to transform the dataset in to an imbalanced one. So, in order to obtain it we decided to reduce the number of instances where the class was equal to 1 (Occupancy=1) to the 4% of the total dataset. We have done this procedure both for the training set and for the first test set, letting away the second test set for avoiding some redundant tables and graphs in the relation. The shape of our training set before the transformation was: {Occupancy=0: 6414, Occupancy=1: 1729}, after the reduction the new shape is: {Occupancy=0: 6414, Occupancy=1: 326. Applying the same procedure to the testset, before we had this shape: {Occupancy=0: 1693, Occupancy=1: 972}, after the reduction the new shape is: {Occupancy=0: 1693, Occupancy=1: 71}. After the transformation of the datasets we have tried the 'Decision Tree' algorithm for see how our classifier works with a different test set where the discrepancy between the two class is very high. These are the results:

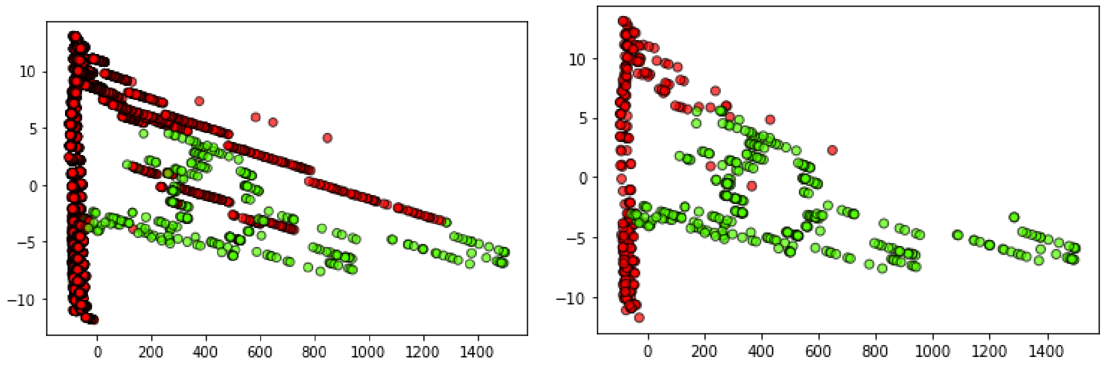
	f1-score	support	accuracy
0	0,981519	1693	0,968820
1	0,470588	71	

	Predicted Occupancy=0	Predicted Occupancy=1
Occupancy=0	1673	20
Occupancy=1	43	28

It's pretty clear that in this case our classifier makes many mistakes whit the minority class, this is a common problem in the reality where the classes are usually imbalanced. More, our dataset is imbalanced with 96%-4%, so a trivial classifier (always positive) will give us an accuracy equal to 0,96. Thus the decision tree applied on this dataset is a little bit better than the trivial classifier. So, in order to solve this problem, we have tried different algorithm for the resample of the minority class or undersampling the majority class.

Random Undersampling

This algorithm aims to reduce randomly the majority in class in such a way to have the same number of instances for each the classes. For seeing graphically how the algorithm undersampling the majority class we have plotted the original imbalanced dataset reduce with PCA and the new undersampled dataset always reduced with PCA.



13- PCA imbalanced dataset on the left and PCA undersampled dataset on the right

From the graphs we can observe that the class ‘Occupancy=0’ which is the red circles are decreases and now we have the same number of instances for both the classes (326). Then, using this undersampled dataset we have built a decision tree that has be used for the classifier the datatest.

	f1-score	support	accuracy
0	0,977348	1693	0,957482
1	0,654377	71	

	Predicted Occupancy=0	Predicted Occupancy=1
Occupancy=0	1618	75
Occupancy=1	0	71

With this method the values are a little bit better than before, but the precision of the class 1 is still a problem. 75 instances which in reality belong to the class 0 are classified with the class 1, and, being the class 1 the minority of the testset, this became a problem in that the classifier duplicate the number of this class. Anyway, the classifier initialized with an undersampled datatest works better than the original one and ROC value comparison above prove it to us. However, the accuracy is a little less than 0,96 (which is the accuracy of a trivial classifier in our case), so give us worst result than a classifier that predict all the instances with the majority class. In this case the ROC value [16] is a little misleading caused by the 0 false negative provided from the algorithm.

Condensed nearest neighbor

Condensed nearest neighbor performs an undersampling based on the removal of the point which have as KNN a minority point. For the parameters ‘n_neighbors’ we choose 5 that is the one that give us the best values.

	f1-score	support	accuracy
0	0,985119	1693	0,971655
1	0,702380	71	

	Predicted Occupancy=0	Predicted Occupancy=1
Occupancy=0	1655	38
Occupancy=1	12	59

Watching the classification report, this type of resample give us the best result, the accuracy is better than the trivial classifier. Also, the precision and the recall in the overall is provide finest results.

Random Oversampling

With random oversampling our aim is to increase randomly the number of instances of the minority class in order have the same shape of the class. So, after that we have ran the random oversampling the dimension of our dataset is: {Occupancy=0: 6414, Occupancy=1: 6414}.

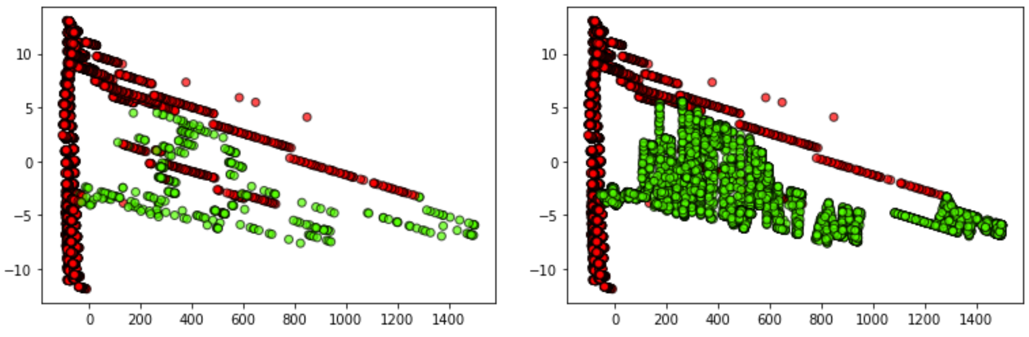
	f1-score	support	accuracy
0	0,989958	1693	0,980725
1	0,760563	71	

	Predicted Occupancy=0	Predicted Occupancy=1
Occupancy=0	1676	17
Occupancy=1	17	54

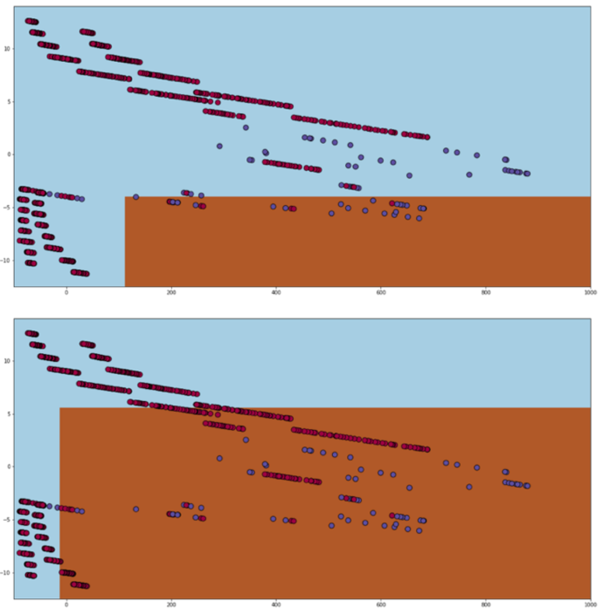
With decision tree initialized with the dataset resample with random oversampling we have obtained a very good results, the accuracy is very high and greater than 0,96. Also the values for the recall and the precision are good enough.

SMOOTE

Smoote algorithm over-sample the minority class through interpolation.



14- PCA pre and post SMOOTE



15- Decision Boundary comparison

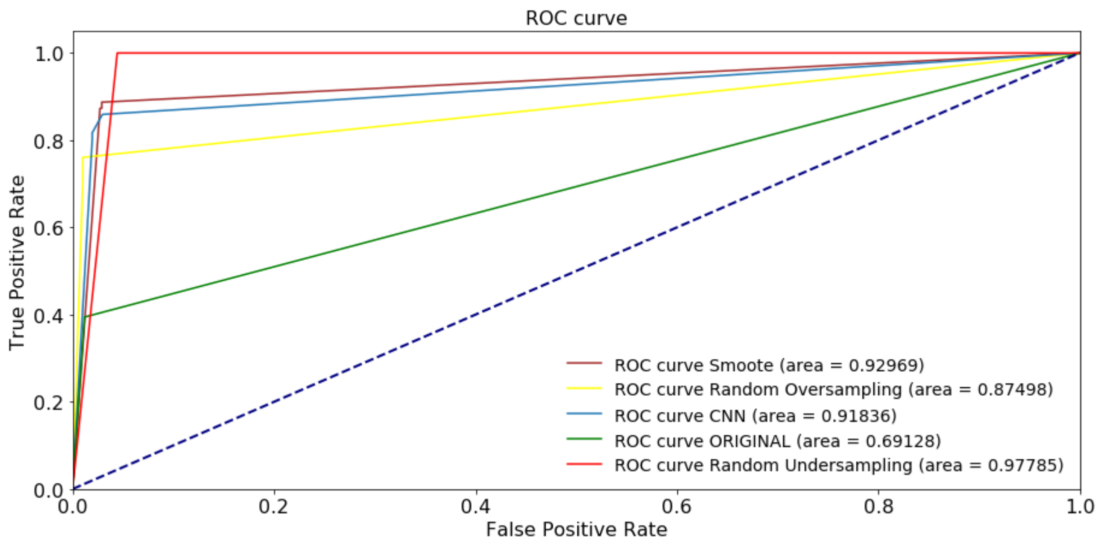
From the graph [14] we can see how the algorithm have worked seeing the different between pre-SMOOTE and post-SMOOTE. The SMOOTE algorithm fills the central part of the chart. This thing would influent a lot the decision boundary in the algorithm. Indeed, from the graph on the side we can see how the decision boundary changes.
The first one, show us the original boundary of the decision tree trained with the initial imbalanced dataset. In the second graph instead, we see the same graph but with the decision boundary of the decision tree trained with the dataset resampled using SMOOTE. In addition, on both the graphs are plotted the test dataset in such a way to see how every point is predicted.
It is almost clear how the SMOOTE algorithm influenced the decision boundary. The big effect of the SMOOTE is on the center of the plan and for this the boundary in the second chart about the class 'Occupancy=1' is much bigger than the first.

Here instead we have the results of the decision tree classifier.

	f1-score	support	accuracy
0	0,982959	1693	0,967687
1	0,688524	71	

	Predicted Occupancy=0	Predicted Occupancy=1
Occupancy=0	1644	49
Occupancy=1	8	63

The results are not bad, even if are not the best provided from algorithm for the managing of imbalanced case. The accuracy gives us a better result than the trivial classifier and the recall has a good value for both the value of the classes.



19- ROC curve comparison

Classification task II

In this task we are going to use different classifier in order to find the better one between:

1) SVC (linear and non-linear)
2) Perceptron (single-layer and multi-layer)
3) Deep neural network
4) Ensemble

As for the first classification task, for train the classifier we have used the dataset called "*datatraining.csv*", then the classifier will be tested using the file called "*datatest.csv*" and "*datatest2.csv*". The attributes "date" and "DayName" are dropped before the application of the algorithms because for our aim in these tasks are useless.

Linear SVC

	Predicted Occupancy=0	Predicted Occupancy=1
Occupancy=0	1639	54
Occupancy=1	3	969

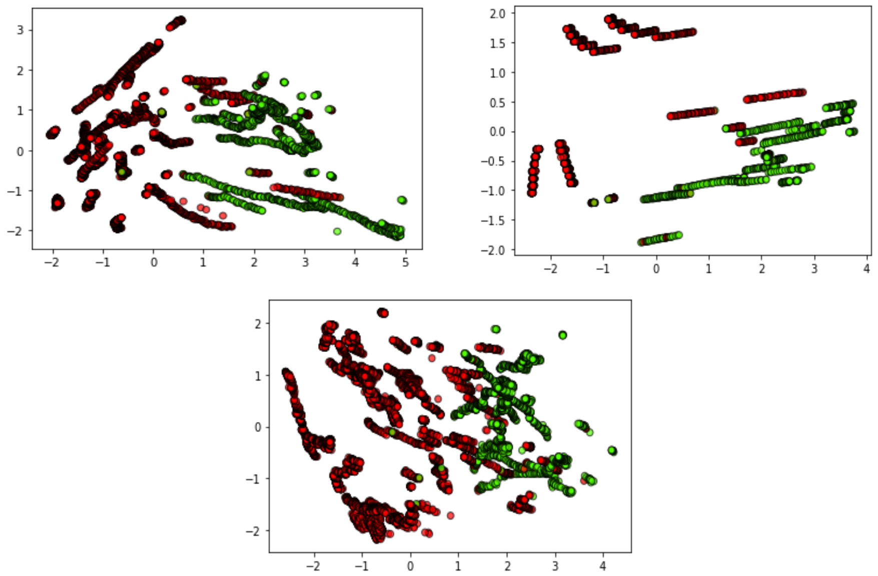
Here we use the Linear SVC algorithm with all the attributes for the classification. First, we have normalized the datasets with the Standardscaler and then, we have done the GridSearch for parameters estimations. The parameter tuned in that case is 'C' which represent the regularization of the algorithm. So, thanks to the GridSearch we have obtained 0,002 as the best possible value of C. For the research of this value we have tried with first the range (1, 20) and then the range (0.001, 1). In this way, after a lot of computation we can be sure that this C is the best. The result of the classification on the two datatest using the best value of C previously obtained are the following:

datatest1	f1-score	support	accuracy
0	0,98	1693	0,978611
1	0,97	972	

datatest2	f1-score	support	accuracy
0	1,00	1693	0,993232
1	0,98	972	

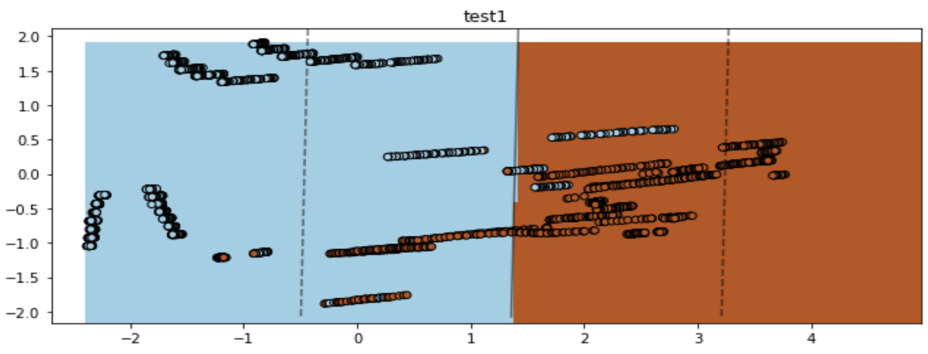
	Predicted Occupancy=0	Predicted Occupancy=1
Occupancy=0	7649	54
Occupancy=1	12	2037

The same algorithm has been used also for the classification of the training set in such a way to discover if we have any type of overfitting or underfitting. The accuracy obtain in that case was 0,987719 which mean that the result is really similar to the ones for the testset. So, we can say that we don't have any type of underfitting or overfitting, considering that the algorithm provides us a good and similar results for all the datasets used. Then another interesting thing to analyze for understand better how this algorithm works is to see the decision boundary of that. For doing this we have to plot the datasets and so reduce the dimensionality to 2. We have used PCA in such a way to reduce the dataset in 2 dimensions (using the parameters of the PCA function n_components=2) and plot it for see the distribution. So, we have obtained those shapes:



20- Reduction of the Training set/Test set 1/Test set2 after the application of PCA

Then, on those reduced datasets, we have tried the Linear SVC algorithm using the C value obtained before for see the results.



21- Decision boundary of Test set 1 - support vectors

From the plotting of the vectors above we can see the decision boundary of the classifier (blue part Occupancy = 0 and brown part Occupancy = 1). It seems that there are not problems, the prediction seems good and the lowest precision value is 0.95. Obviously these are not the real boundary, in that case we see the decision boundary of the 2-dimensions datasets and the result of the algorithm seen previously correspond to the datasets using all the features, but these projections of the decision boundary are pretty important for understanding which type of shape our classes have in the chart.

Non-Linear SVC

Here we have run the Non-Linear SVC algorithm, using all the attributes. This type of classifier gives us the possibility to use different kind kernel. In our study we have used three different kernels: linear kernel, radial basis function(rbf) kernel and poly kernel. First of all, we have used the GridSearch for tuning the hyper-parameters. The tuning of the parameters changes depending on the kernel chosen, so we have run the GridSearch three different times, one for each type of kernel tested. So, let’s start with the **linear kernel**. In this first case the parameter submitted to the GridSearch is ‘C’ that as for the linear SVC represent the regularization of the algorithm. The best value of C founded in that case is 0,1. The best value has been calculated between 0,1 and 10. Using this C, we obtain result not useful. In fact, all the instances of the datatest1 are classified as ‘NotOccupied’ so we can say that this kind of classifier don’t provide a good classification. With kernel equal to **rbf** instead we have good results. First of all, we went to check the best value for the parameters using as usual the GridSearch. The parameter tested in that case in ‘gamma’, the range in which we went to check is (0.01, 5), so after a very long computation the best value of gamma coming out from the GridSearch is 0,1. Using this parameter, we have obtained very good results:

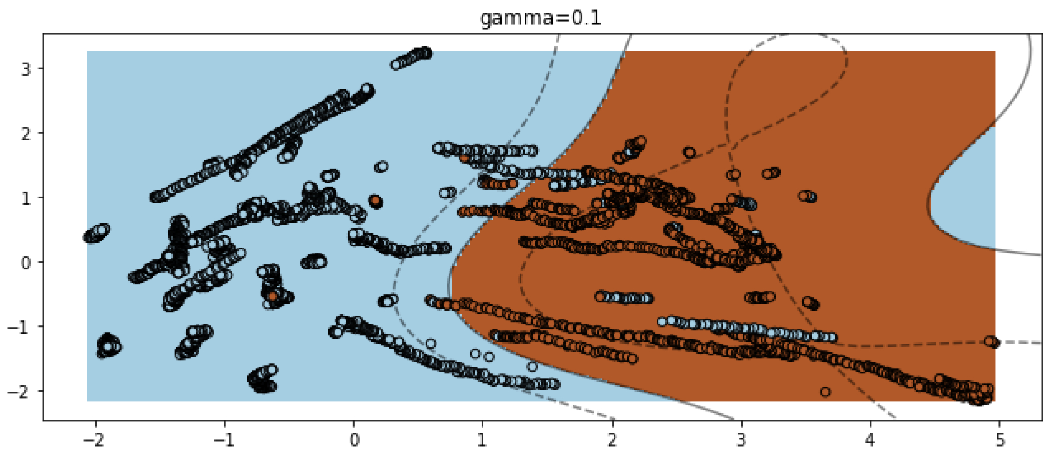
datatest1	f1-score	support	accuracy
0	0,97	1693	0,961350
1	0,95	972	

datatest1	Predicted Occupancy=0	Predicted Occupancy=1
Occupancy=0	1648	45
Occupancy=1	58	914

datatest2	f1-score	support	accuracy
0	1,00	7703	0,993232
1	0,98	2049	

datatest2	Predicted Occupancy=0	Predicted Occupancy=1
Occupancy=0	7649	54
Occupancy=1	12	2037

Testing it also for the training test we have obtained an accuracy equal to 0,988579, so we can see without doubt that this type of classifier works very well avoiding underfitting and overfitting. Then, in such a way to discover how this classifier classify our instances we have decided to plot the dataset in two dimensions, using always PCA for the reduction, and se the shape of the decision boundary for the training set.



22 - Decision Boundary SVC with kernel=rbf

As we can see, the SVC with kernel=rbf and gamma=0.1 gives us a good shape of the decision boundary of the dataset in two dimensions. The last kind of kernel tested is ‘**poly**’. Initially, as usual, we went to tune the parameters of this function. The parameters tested are ‘gamma’ and ‘degree’, both tried between 0 and 10. So, after the GridSearch on these two parameters in the range previously showed we have obtained the best value of them: degree=4 and gamma=8. These two parameters are the ones that give us the best result crossed between the two testsets and the training set, but as we can see from the tables beside the classification reports are not so good as the rbf.

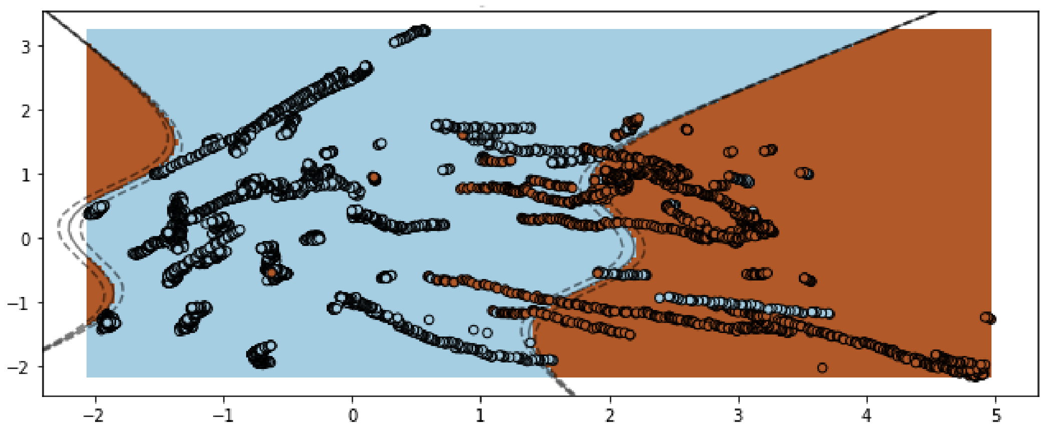
datatest1	f1-score	support	accuracy
0	0,87	1693	0,805622
1	0,64	972	

datatest1	Predicted Occupancy=0	Predicted Occupancy=1
Occupancy=0	1648	45
Occupancy=1	58	914

datatest2	f1-score	support	accuracy
0	0,78	7703	0,700574
1	0,54	2049	

	Predicted Occupancy=0	Predicted Occupancy=1
Occupancy=0	7649	54
Occupancy=1	12	2037

The results of this classifier on thee training set instead are very good, for example with an accuracy equal to 0,996438 and just 29 instances misclassified out of 8143. Which is a clear example of overfitting, our model works very well on the training set but give us a lower result on the testsets. So, we can easily say that this model it’s not a good model for our purpose.



23 - Decision Boundary SVC kernel=poly

So, in conclusion, looking at the graph of the decision boundaries and at the classification reports is quite obvious that the type of kernel which represent better our dataset is 'rbf'.

Perceptron

In this part of the project we went to try the perceptron classifier. We have used both single layer and multi-layer perceptron in such a way to find the best one between the two.

SINGLE LAYER: For doing the single layer perceptron, first of all, we have tried the usual GridSearch in order to find the best attribute of the function. The parameters put under the tuning are:

- penalty: l1 or l2
- alpha: [0.00001, 0.0001, 0.001, 0.01, 0.1, 1, 5]
- tol: [1e-1, 1e-2, 1e-3, 1e-4, 1e-5]

And the results of the GridSearch give us as best values: penalty=l2, alpha=0.0001, tol=1e-1. Using these parameters however, we don't get good value at all, for simplicity, using only the accuracy as metric for the validation, we have a good value on the test1: 0,842401 and a bad value for the test2: 0,584187. These values could be enough for understand that the single layer perceptron is not a good classifier in our case, but comparing these accuracy with the accuracy obtained on the training set, which is 0,918503, we can also say that our model in that case is overfitting, especially on the testset 2. These assumptions are enough for say that the single layer perceptron is not a good chose in our case

MULTY-LAYER: For the multi-layer perceptron, the biggest problem is to find the right number of hidden layers to have and their size. Another parameter to estimate is the solver, that could be 'adam', 'lbfgs', or 'sgd'. So, starting with the greatest problem, the one about the number and size of the hidden layer, we have tried the MLPClassifier with the default value of this parameter (1 hidden layer with size =100). In this way we have obtained a very good value of accuracy for the training set and for the first test set (0,98 and 0,97) but we have a little problem with the test set 2 (accuracy=0,87). So, taking the classifier with the default parameters as landmark, we went to run the usual GridSearch in order to estimate the best values of various parameters. For the parameter 'hidden_layer_size' we have started from some standard form and, then, we have restricted the target on the best value returned form the tuning, looking for some values in their round. For instance, we start looking in the different values [(10), (50), (150), (20, 40), (50, 100), (100, 200), (20, 40, 60), (50, 100, 200), (32, 64, 128)]. The classifier return as best value for 'hidden_layer_size' 10, so now we went to look around it, re-run the GridSearch changing the parameters in [(5), (10), (13), (15), (18), (20), (22), (25), (28), (30), (33)]. From this second round of the GridSearch the best value returned was 15, so we have decided to use it for our model. The second round of the parameters tuning was run also taking into account the other parameters to tune. So, after all, the parameters founded were:

'activation': 'logistic', 'hidden_layer_sizes': 15, 'learning_rate': 'constant', 'max_iter': 200, 'momentum': 0.1, 'solver': 'adam'.

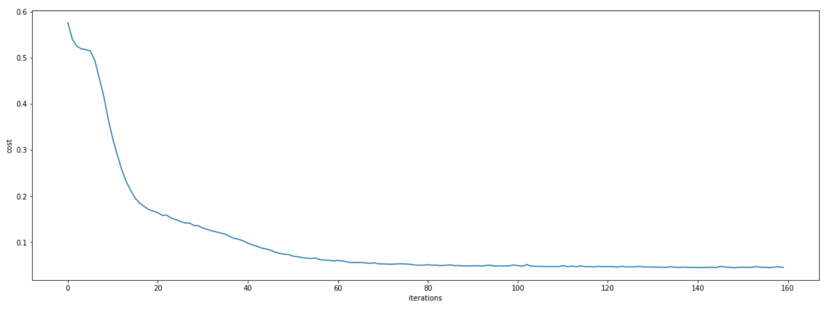
datatest1	Predicted Occupancy=0	Predicted Occupancy=1
Occupancy=0	1639	54
Occupancy=1	3	969

datatest1	f1-score	support	accuracy
0	0,98	1693	0,978611
1	0,97	972	

datatest2	f1-score	support	accuracy
0	0,99	7703	0,987284
1	0,97	2049	

datatest2	Predicted Occupancy=0	Predicted Occupancy=1
Occupancy=0	7635	68
Occupancy=1	56	1993

The value of the accuracy for the training set is 0,987351 which assure us the lack of overfitting or underfitting.



24 - Loss curve

In the graph we can see how the loss curve change during the iterations and how, after the eightieth iteration, the loss curve for the training set stabilizes without important peak.

Deep neural network

In this part we have used the deep neural network in order to classify the instances. We have built three different models in order to be able to confront them. The layers of each architecture were been chosen in three different way, obviously, having a binary class we must have the last layer with size 1:

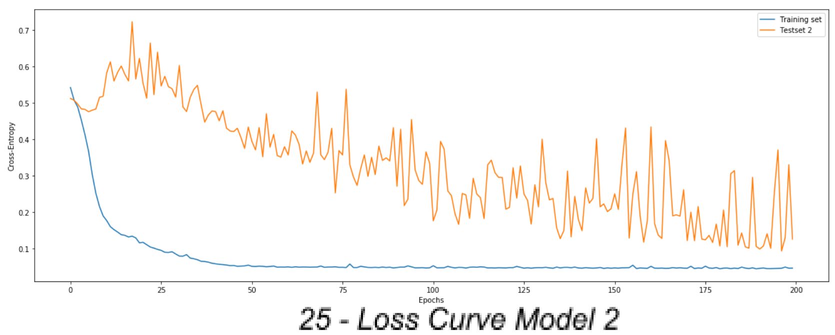
- The first model has a standard number and size of the layer, three layers with a descending size, 200, 100 and 1. The model has been compilated with batch_size=100 and 50 epochs.
- The second model was chosen after a lot of attempts in such a way to have the best possible result having just 3 layers. Thus, we have 3 layers with a descending size, 60, 40 ,1. The model has been compilated with batch_size=100 and 200 epochs.
- The third and last model has instead four layers, also in this case we tried to find the best values for this layer and after some attempts, we decided to use the size 100, 50, 25, 1. The model has been compilated with batch_size=100 and 200 epochs.

All the model has been trained and validated using two different partition of the training set. So, first of all we have split the training set in two parts, the bigger one (80%) has been used for the training and the smaller one (20%) for the validation. Then, all the models have been tested using the two testsets and the reports are showed beside. For simplicity, we will use only the values of the accuracy and of the loss for compare the different model.

	Training set	Test set 1	Test set 2

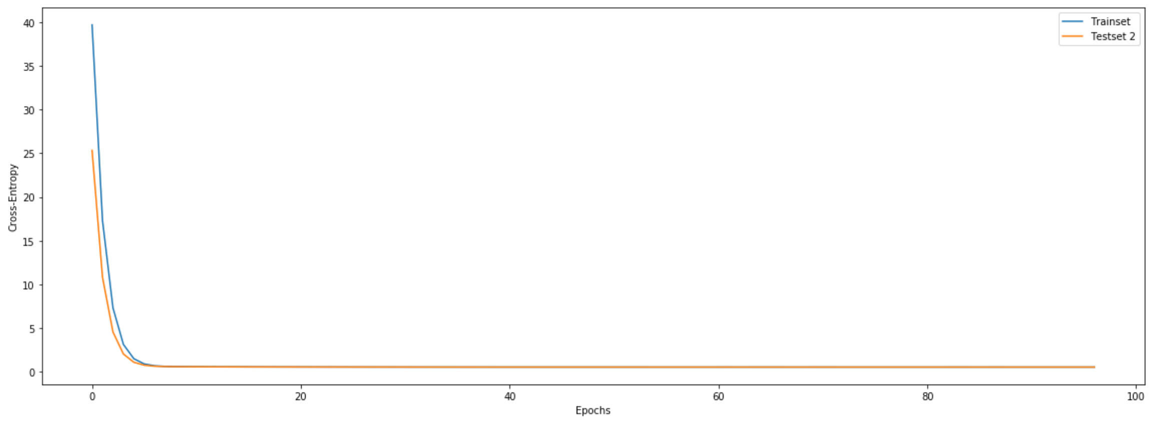
Model 1	Accuracy: 0,988333 Loss: 0,043260	Accuracy: 0,978612 Loss: 0,081621	Accuracy: 0,907609 Loss: 0,307249
Model 2	Accuracy: 0,988333 Loss: 0,044435	Accuracy: 0,978612 Loss: 0,081335	Accuracy: 0,948318 Loss: 0,108519
Model 3	Accuracy: 0,987105 Loss: 0,047070	Accuracy: 0,978612 Loss: 0,083112	Accuracy: 0,768458 Loss: 0,616416

From the result, we can observe that with all the three models give good performance on the training set and on the first test set, but we can't say the same thing on the second test set. In this last case only the second model give a good performance while the other two model, especially the last one are in overfitting. So, in that case the model 2 is the best created, from the classification report is the only one that give us good results about the test set 2. Now, in order to see how our model 2 work, we have decided to plot the chart of the loss of the training set and of the loss of the testset 2 along the iterations.



25 - Loss Curve Model 2

How can we see, the two curves converge, but the real problem is the big instability about the loss of the testset 2. That's is a big problem, our model doesn't work very well, even if the loss of the testset 2 varies in the range 0.15, 0.35, which are small values in overall, these variations give a very instable situation to our model. So, now we have delimited our problems, with the training set and the first test set we don't have any problem, the issue is with the second test set, the only model that can give us good values on the second test set is the model 2, but, as we haven see, this model is very instable on that dataset. Thus, our purpose from now, about the deep neural network, is to find a good modify on the second model that can give us good and stable results on the test set 2 and that don't worsen the result on test set 1. The first try is with the early stopping. It's quite obvious that the early stopping can't give us any improvement in that case, the graph above show us that reducing the number of epochs is not useful. In fact, we can use 4 different score metrics for stopping the algorithm when this metric is flatting, but while the metrics about the training set stabilize after the 50th iterations (both loss and accuracy), for the testset 2 we never had a stabilization, which is also our problem. So, after many attempts, we go to next try. The good modify that show us improvement is L2 regularization. In fact, after a lot of try for find the good value for this parameters, we discover that giving 0,9 to this for all the layer we get results for testset 2 similar to the original model speaking about the accuracy and loss, but also that the curve of the loss along the iterations flattens out earlier and remains stable for all the epochs.



26 - Loss Curve Model 2 L2 regularization

In addition, this new model doesn't worsen the result on test set 1 and on the training set as we can see form the table beside.

	Training set	Test set 1	Test set 2
Model 2	Accuracy: 0,988333 Loss: 0,044435	Accuracy: 0,978612 Loss: 0,081335	Accuracy: 0,948318 Loss: 0,108519

Model 2 L2 regularization	**Accuracy: 0,988279** **Loss: 0,044512**	**Accuracy: 0,978612** **Loss: 0,081335**	**Accuracy: 0,947539** **Loss: 0,108701**

So, with these changes, we have obtained a new model that give us good results for all the datasets, that is stable and that avoid overfitting and underfitting.

Random Forest

In this part we went to try the random forest classifier in order to understand if this algorithm could give us a good estimation of the instances of the datasets. The first thing done in this part is the one of having a good estimations of the parameters of the algorithm. So, in order to do this, we start with the usual GridSearch on these parameters:

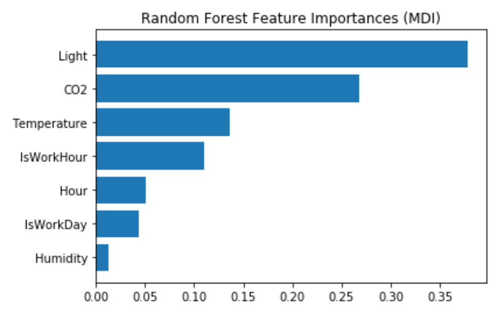
- 'n_estimators': [5, 8, 10, 13, 15, 18, 20, 30, 40, 50, 60, 80, 100],
- 'max_depth': range (2,5),
- 'min_samples_split': range (2,5),
- 'criterion': ['gini', 'entropy'],
- 'min_samples_leaf': range (1,5),

And, using all the features in the dataset and crossing the classification on the training set and the two testsets we have obtained these results:
'criterion': 'gini', 'max_depth': 3, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 100.

With these parameters we have obtained a very good value for all the datasets, for example, looking at the accuracy we have:

- Training set: 0,988824
- Test set 1: 0,978236
- Test set 2: 0,993232

These results are very good, but we can do something more. For example, computing the features importance we can quickly see that not all the features have the same importance in the classification. For example, is clear that the big part of the work is done by the variables 'Light' and 'CO2' but we can't say obviously that using all these two attributes could be enough for have a good estimations. Thus, we went to run again the GridSearch again, but, this time, using as parameters a set mad by us. In this set we put a first list with all the attributes, then a second with all the attributes without the least important (Humidity) and so on until to the last list with the two more important attributes.



27 - Feature importance

The set of combined features is the one beside:

1. ('IsWorkDay', 'Hour', 'IsWorkHour', 'Temperature', 'Humidity', 'Light', 'CO2'),
2. ('IsWorkDay', 'Hour', 'IsWorkHour', 'Temperature', 'Light', 'CO2'),
3. ('IsWorkHour', 'Temperature', 'Light', 'CO2'),
4. ('Temperature', 'Light', 'CO2'),
5. ('Light', 'CO2')

The GridSearch was run with the same set of parameters to tune seen before in such a way to have for each set of attributes the best parameters possible. The result obtained are interesting, the accuracy for each datasets are similar, just some millesimal figures change but all the classifier with no the all set of features need at most 20 estimators for give us these results while the classifier with all the feature need 100 estimator for provide us it. So for rapidity, we can see that using only the attributes 'Light' and 'CO2' with the right parameters give us a good results very similar to the ones with all the features:

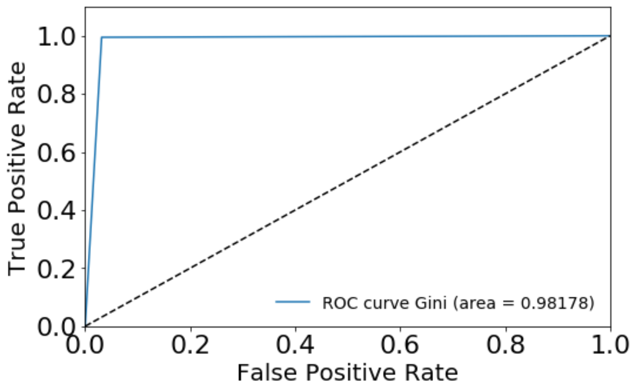
Parameters set: 'criterion': 'gini', 'max_depth': 4, 'min_samples_leaf': 2, 'min_samples_split': 2, 'n_estimators': 10.

datatest1	f1-score	support	accuracy
0	0,98	1693	0,978236
1	0,97	972	

datatest1	Predicted Occupancy=0	Predicted Occupancy=1
Occupancy=0	1640	53
Occupancy=1	5	967

datatest2	f1-score	support	accuracy
0	1,00	7703	0,992719
1	0,98	2049	

datatest2	Predicted Occupancy=0	Predicted Occupancy=1
Occupancy=0	7644	59
Occupancy=1	12	2037



The accuracy value of this classifier on the training set is 0,9888247. This classifier doesn't have any problem of overfitting or underfitting and provide us good results so is a good choose.

28 - ROC Curve Random Forest with 'Light' and 'CO2'

Bagging Classifier

With this classifier the trivial thing is to find the good base estimator and the number of estimators to compile for get a good result. So, in order to do this, we ran the GridSearch using as a base estimator some of the classifier seen until now:

- RandomForestClassifier(criterion="gini", max_depth=3, min_samples_split=2, min_samples_leaf=1,random_state=0),
- KNeighborsClassifier(n_neighbors=81, weights='uniform'),
- DecisionTreeClassifier(criterion='entropy', max_depth=3, min_samples_split=2),
- GaussianNB(),
- SVC(kernel='linear', C=0.1),
- SVC(kernel='rbf', gamma=0.1)

Then, we add also the parameter 'n_estimators' as a parameter to tune in the range: [5, 8, 10, 13, 15, 18, 20, 30, 40, 50, 60, 80, 100]. All the classifier used in the GridSearch for the parameter 'base_estimator' are already implemented with the parameters found in their tuning. The results of the grid search were as 'base_estimator' the GaussianNb, thus Naïve-Bayes and 5 as number of estimators to create. These are the results running the bagging classifier with these parameters:

datatest1	f1-score	support	accuracy
0	0,98	1693	0,978611
1	0,97	972	

datatest1	Predicted Occupancy=0	Predicted Occupancy=1
Occupancy=0	1639	54
Occupancy=1	5	967

datatest2	f1-score	support	accuracy
0	1,00	7703	0,993232
1	0,98	2049	

datatest2	Predicted Occupancy=0	Predicted Occupancy=1
Occupancy=0	7649	54
Occupancy=1	12	2037

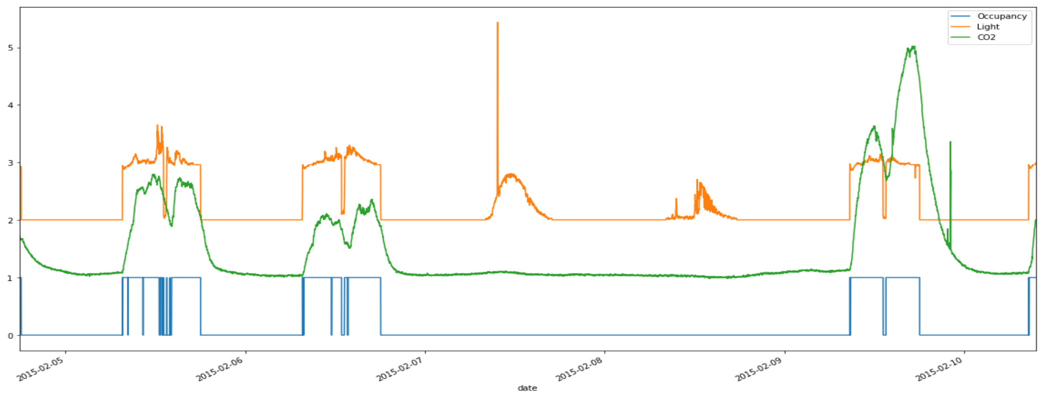
The accuracy using the training set is 0,988210 which means that both overfitting and overfitting are avoided.

Time Series Analyses

In this part we are going to exploit the temporal information of our dataset in order to use it for clustering, forecasting and classification of these time series. The features in our dataset that can be used for a good analyzes of the time series are:

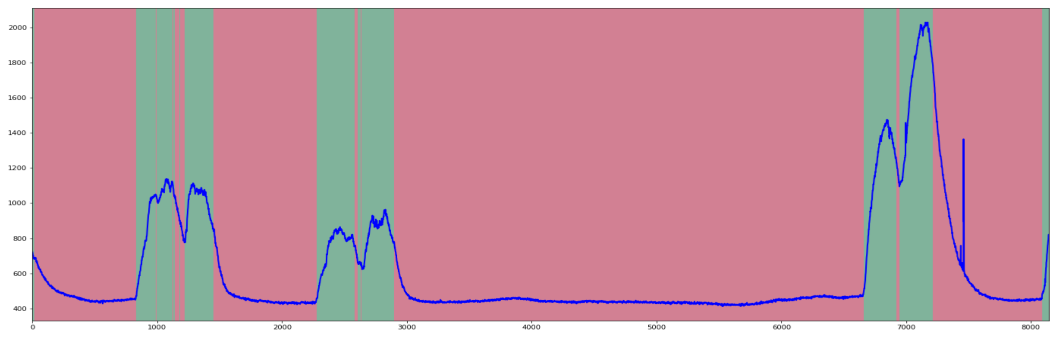
- Light
- CO2
- Temperature
- Humidity

These features give us a single time series for each attribute that show us how these information change during the days take into account. The values of these attributes are taken one time a minute. Between these features we choose to take for us to analyze Light and CO2. These two have the best correlations with the class, they have a trend that can be very useful in the forecasting part and they are the more significant features of the datasets.



29- 'Light', 'CO2', 'Occupancy' comparison

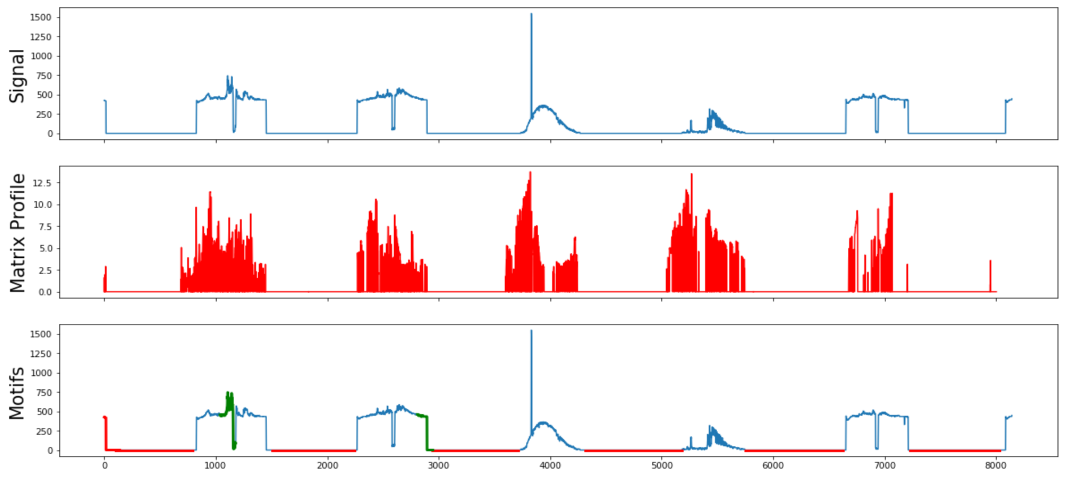
First, seeing the three different time series above we can easily see how the different peaks in Light and CO2 correspond to the part of the day where the class occupancy is equal to one. This fact is also well descripted in the initial graph, where, when the background is red means that the class is zero, when is green the class is one. For the attribute Light [2] the correlation between a peak and occupancy equal to one is very high, it’s clear that when the light goes down to zero also the occupancy does the same thing.


30- 'CO2' class correlation

Looking at the same graph but with CO2, we can observe the same thing, but this time less obvious, for example the last peak of CO2 is really different from the other, is higher and the big part of the decrease is not in the class Occupancy=1. Instead CO2 has an interesting property, differently from Light, also in the middle part of the graph, where the occupancy is zero, the graph stays low and doesn't have a littler peak, due to the natural light that comes into the office. In general, it seems that the peak of CO2 is a bit shifted forward respect to the occupancy, probably due to the fact that the CO2 in the office need some minutes after the open to rise, and some minutes after the closure to come back to low values.

MOTIFS

The first thing that we have analyzed was motifs, scilicet we were looking for repeated patterns in our time series. For this part we have analyzed the time series of light, because of their strong trend which could give us best results. Before calculating the motifs of the time series, we had to calculate the matrix profile of it. The window used for the matrix profile was 150, this number was chosen because we needed a length of the motifs that might be significant in our time series which has a length equal to 8143 samples, for example a length of the motifs equal to 10 would have been inconsistent. So, after many tests we discovered that the length of 150 gives us the better result in what we are looking for. Beside we can see the result and comparison between the time series of Light, the matrix profile extracted and the relative time series with the motifs parts.

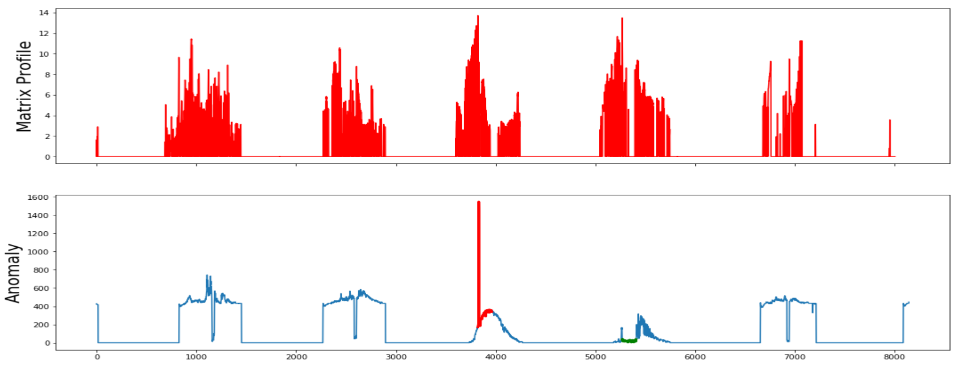


31- Matrix Profile and motifs of Light

Here we can see a clear repeated pattern is when the light is zero, this pattern is repeated every day when it is night, it is also clear the correlation between the low part of the matrix profile and this motifs. The algorithm finds out also another motifs, the green one. The pattern in this case shows us that the light quickly decreases to zero from a high value. This type of situation is frequent, especially during the lunch break, as we can see in the first green pattern, and during the finish of the work hour, the second green pattern. These motifs we found have a great correlation with the class, along with the red motifs the class is always zero cause of the night, and the green motifs instead show us when the class goes from one to zero.

Anomaly Detection

In this part we want to analyze some anomaly in our time series of Light. The time series used is the stock time series, so we haven't done any transformation to it in order to normalize or smooth it. For discover the anomaly we had to rebuild the matrix using the same window as before.



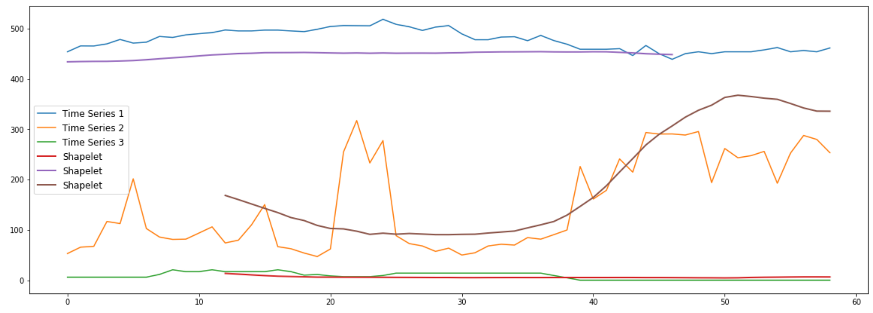
32- Matrix Profile and anomalies of Light

The anomaly founded is clearly the peak which is marked in red. This peak of light is obviously an anomaly, the day is not a workday, so the occupancy is equal to 0 and, as we have already seen, in that case the light is always under 370. This anomaly in fact corresponds to the peak of the matrix profile. There is another anomaly founded, the green one. This one is an anomaly because the graph light in those points doesn't have a "gaussian" form which is typical for a non-workday. This anomaly is clearly less important to the first one, in fact this doesn't change our knowledge about the distribution of the light during the day, in that points the class, studying only light, would be zero with or without the anomaly.

Shapelets

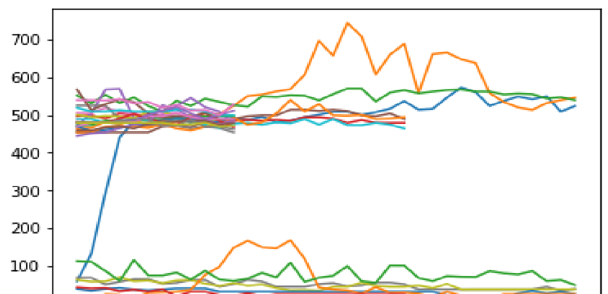
In this case, for finding the shapelets of the time series we need more than one time series. In fact, the shapelets are useful for finding some shape of the time series which represent a peculiarity for a specific class.
So, in this case, we have splitted the time series of light grouped by the hour. Thus, we have now one time series of light for each hour in the dataset. All the time series, for fitting shapelets algorithms have to have the same length, so first, we have applied some transformation in order to have all the time series with length equal to 60. So now we have 135 time series. For discovering the shapelets we have used different methods, but we will present only the two that give us better results.

Shapelet Model: First of all, we went to see the size of our shapelets using the Grabocka method. In this way we have obtained 3 different shapelets with length equal to 47. Then, we have implemented the function ShapeletsModel of tslearn.shapelets with shapelets size founded before and with a weight regularization equal to 0.7. Running the model we obtained an accuracy equal to 0.7925 which is quite good. Now we were able to extract the shapelet.



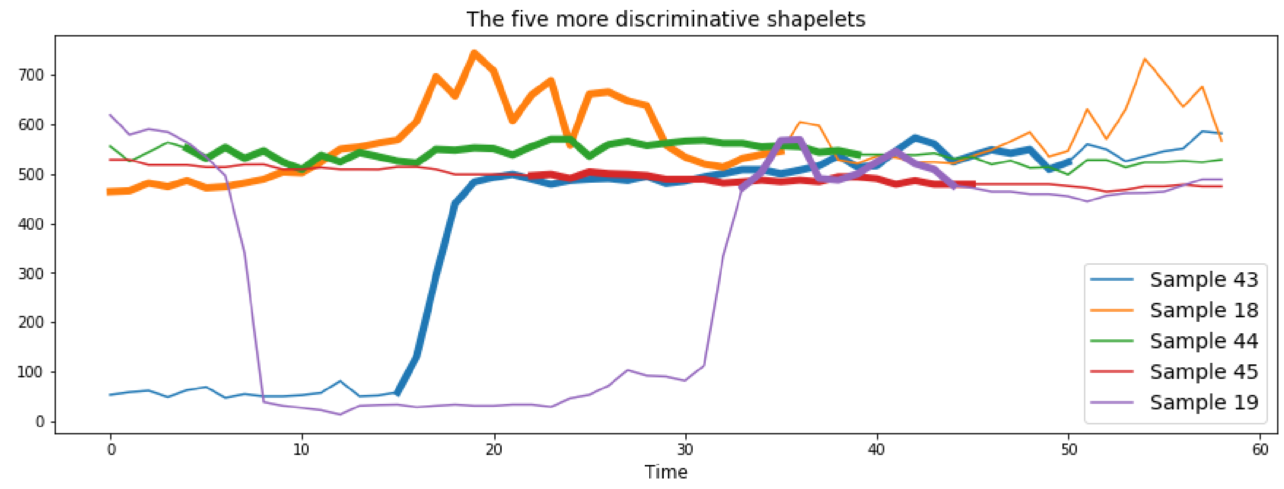
33-Shaplets and Time Series

Above we can see the three shapelets extracted (the red, violet and brown line) and three different time series that can be associated with these shapelets. Analyzing these shapelets we can find exactly the different values that light can have. The red one is referred to all the night hour, where the light is constant to zero, the brown one is when we are in a situation like non-workday but during the day, so the light is fewer due to the fact that the artificial light is zero and the violet one is when there is someone in the office, so the light is the sum of the natural and the artificial light.



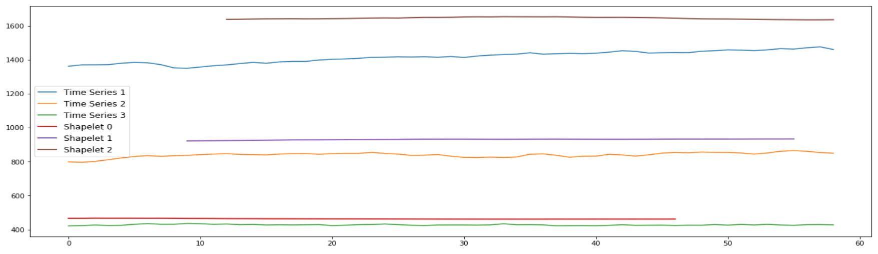
Shapelets Transform: Another way used for discovering the shapelets was the library ShapeletTranform from pyts.transformation. We have used a window_sizes=[12, 24, 36] and fitting the model we have found 29 different shapelets.

In this way we can see that the things are really confused, so, in such a way to improve the clarity of everything we have plotted just the five more discriminative time shapelets combined to the time series.



34- Shapelets of Light (on the top) and the the five more discriminative on the the bottom

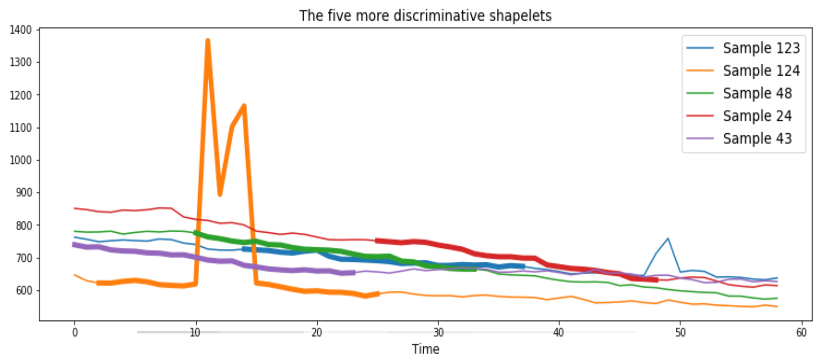
Now it's clearer to analyze the shapelets. The most important shapelet is the blue one, that shows us the rise of the light from 0 to a typical workday and occupied light value. All the other shapelets show us a situation where probably there is someone in the office, caused by the high value of the light. But from this graph it is also clear that all the shapelets regarding the class Occupancy=1, thus, with light zero or close to it are not so discriminative in our case. We have tried to do the same thing, the shapelets analyze also using the time series of CO2. The results in that case are really different. Using the ShapeletsModel library we have obtained a value of 0.9259 as accuracy, so a better result. As before we have obtained 3 different shapelets with length equal to 47 but in this case the shapelets are less indicative respect to the one founded with the same method with Light.



35- Shapelets of CO2 with ShapeletsModel

Also using the Shapelets Transform we haven't obtained very clear shapelets, plotting just the five more discriminative results are not so discriminant.

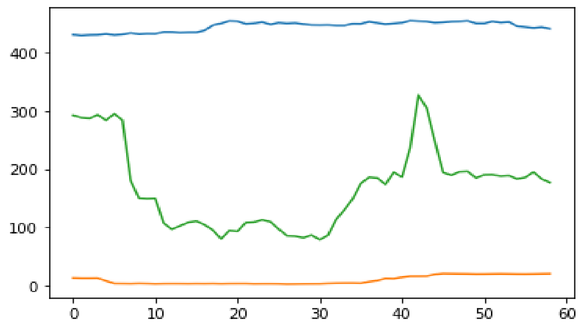
In this case the most important shapelets found is the orange one, but, as we have already said, from these shapelets are not so significant in our case, the class is not really distinguished and all of the shapelets are really close one to each other.

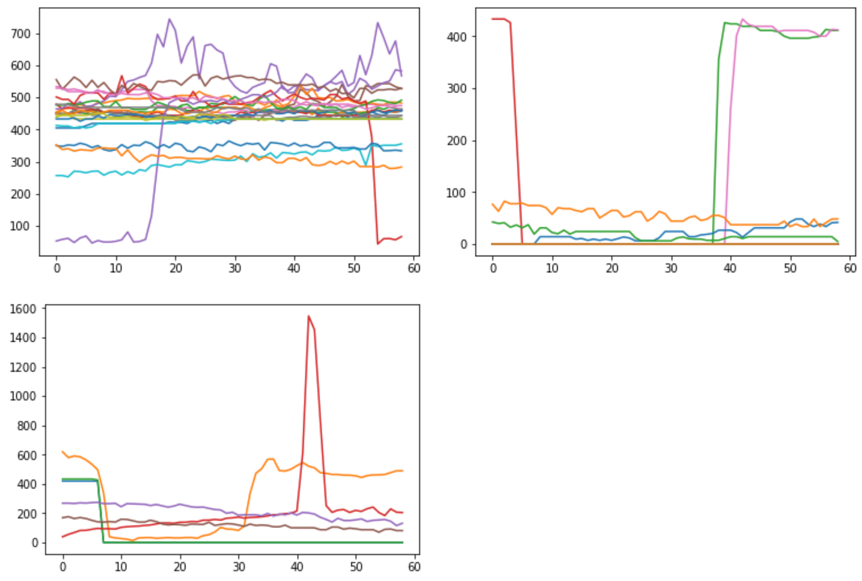


Clustering

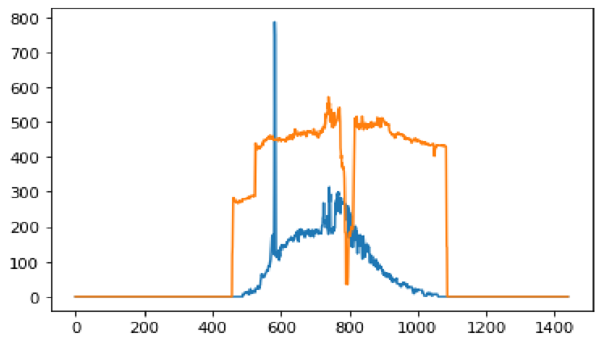
The second part of the study of the time series is about the clustering. In this task indeed we have applied some clustering techniques in order to group the time series based on different factors depending on the type of clustering used. For clustering the time series obviously, we need more than 1 time series, thus we have used the same technique used for the shapelets, we have divided the dataset in more than one time series. We have used two different splits in order to make more than 1 try. The first split is on the day, this split will be not much significant, in fact only 5 different time series are created. The second split is the one more important, is the split based on hour, exactly the same split used for the shapelets.

Shape-Based Clustering: For the shape-based clustering we have used the algorithm 'TimeSeriesKMeans', for the split on hour the best number of clusters is 3, our purpose in that case it is to obtain in these 3 clusters the three different types of hour that we can find. The first cluster returned in fact, with label 1, contains all the hours that, based on light, are without doubt not occupied, this cluster will contain all the time series in our split of hour where the light is always a straight line close to zero. The second cluster, with label 0, is the cluster contain all the hour that without doubt are occupied, thus the time series where the light is constant over 300. The last cluster, with label 2, contains all the hour that are boundary hour, so the ones where we can't say with precision basing only with light the occupancy. This in fact, is the graph of the three centroids of the clusters, it's really discernable how these centroids work.


37- TimeSeriesKMeans centroids


38- Clusters distributions

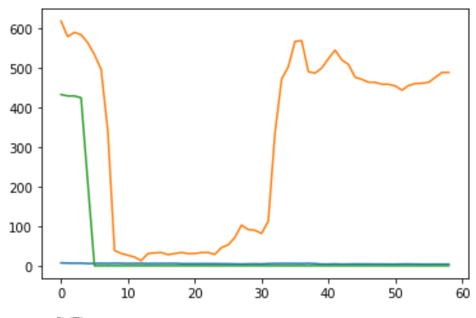
Here instead we can see how the time series are divided in the three clusters. The first cluster, label 0, contains all the hours with occupancy equal to 1, it's clear to see that all the time series are over 300. In the second one instead the time series are always close to zero that means that there is no light or there is few light, a clear indicator of occupancy equal to 0. The last one is instead the cluster with the tricky hour, the ones where the class is not easily identified. A way to understand better the tricky hour is the division by day. Using the same type of clustering on the dataset splitted on day we can understand if these hours belong to a work day or not a work day in such a way to put it in the right class. So, in order to do this, we have applied the same algorithm on this dataset, but, this time, using K equal to 2, in such a way to obtain the division work-day and nonwork-day.


39- TimeSeriesKMeans centroids on Days

These are the centroids of the clustering on the dataset splitted on day, basing only on the label (we have only 5 time series so it is easy to understand if the label is correct), we can say that this type of clustering work perfectly on our goal for this dataset.

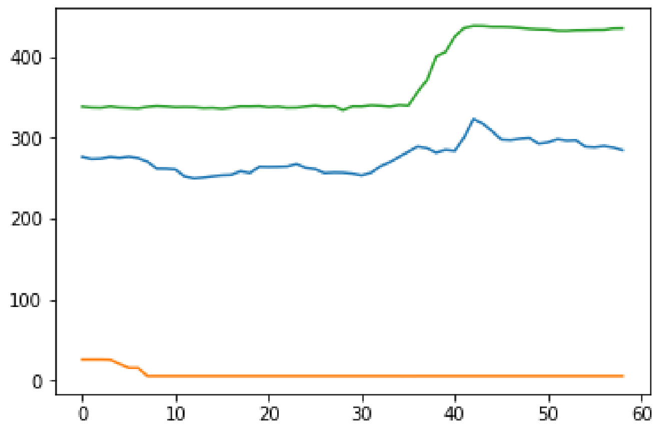
Featured-Based Clustering: The feature-based clustering is based on the clustering of the time series thanks to the different features of each time series. After calculating the different features of the time series (mean, median, variation, etc..) we have put all in a list and used the KMeans algorithm for doing the clustering. The best number of clusters for the classification is always 3 for splitting the hours and 2 for splitting the day.

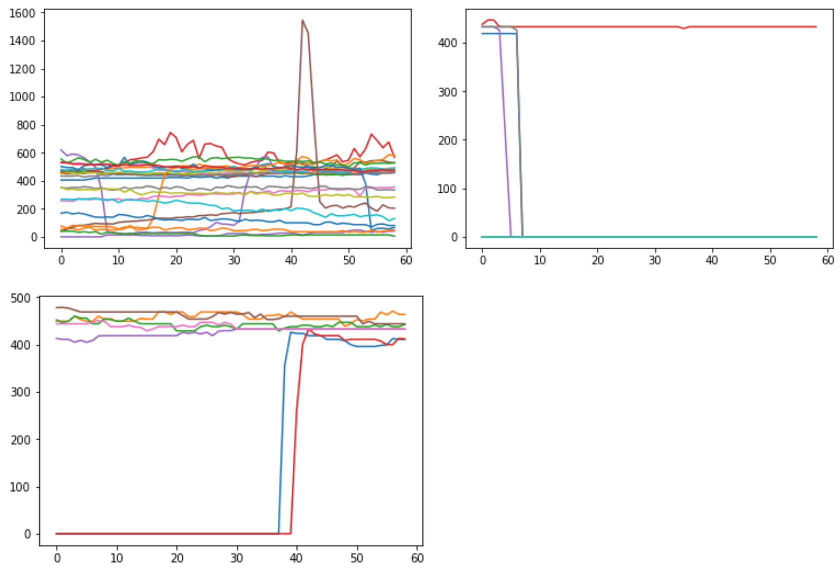
While for days, we have obtained a good clustering, similar to the shaped-based, but for the hours this time the clustering is not so good.


40- Featured-Based centroids

The good results for days given are probably from the strong difference between the time series and the low number of them. For hours, we can see that the centroids are not so significant and also looking at the labels we see that the greatest part of the time series are in the same cluster. So, we can say that this type of clustering is not so relevant in hour case.

Compression-Based Clustering: The last type of clustering used is compression based. For doing that first we have compressed the time series and then we have used that for doing the clustering using the DBScan algorithm. The results of this type of clustering are good, similar to shape-based for hour, while for the split on day the results are not good enough, in fact all the time series are clustered under the same cluster. Speaking about the split in an hour we can see how the centroids are similar to the ones in shape-based, that give us a good confidence about the results. But, going deeply, we see that the cluster created are not so significant and a little messy, at least for our initial purpose


41- Compression-Based centroids


41- Clusters distributions

In fact seeing the different cluster we can say that is not easy to understand the meaning of a cluster like for the shape-based, there is no significant division and we can also see evidence of errors like the red time series in the second cluster. So in conclusion we can say that the best type of clustering is the shape-based that reaches our purpose of the division for meaning of the time series.

Forecasting

In this part of the project we went to try the forecasting, i.e. the prediction of future values of a given attribute. The features used are 'Light' and 'CO2' for the reasons seen before, but we will explain better the procedure for 'Light' then we will also show the result for 'CO2' without a lot of explanation for avoiding a lot of repetition.

Light: The first and very important thing done is see if the original time series of light is stationary in order to understand if it's possible to forecast it. The technique used for knowing if a time series is stationary or not is the Dickey-Fuller test. Unfortunately, the original time series is not stationary, so, in order to make it stationary we have to apply some transformation to it. We were fortunate, in fact, the log transformation of the time series is stationary as we can see form the result of the Dickey-Fuller test:

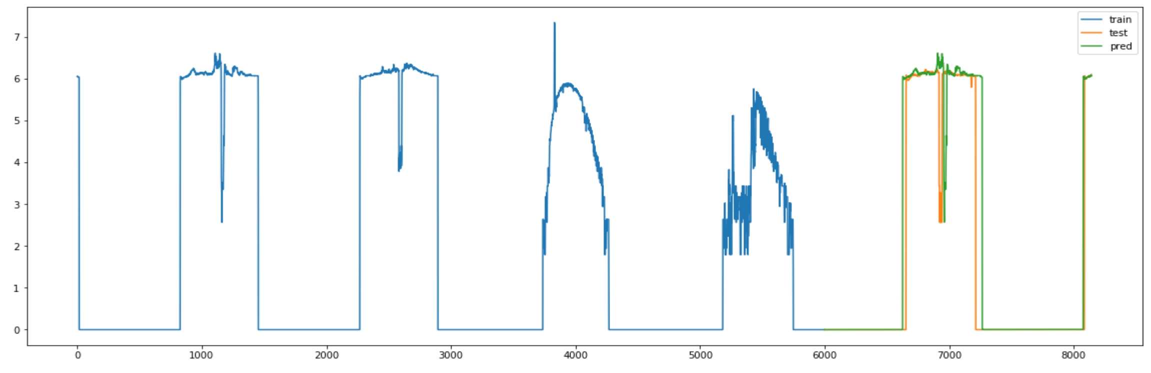
```
      Results of Dickey-Fuller Test:
Test Statistic                   -3.700680
p-value                           0.004104
#Lags Used                        7.000000
Number of Observations Used    8135.000000
Critical Value (1%)              -3.431154
Critical Value (5%)              -2.861895
Critical Value (10%)             -2.566959
```

The Test Statistic in fact is less than all the critical value and p-value is very small which is good for our forecasting. So, we know that the logged time series of 'Light' is a good candidate.
Then, we went to check if hours have a trend and a seasonality in order to understand which forecasting model will be better to use. So, we have decomposed the time series seeing the trend, the seasonality and the residual. There we have seen that our time series has a seasonality, so the best models to forecast it would be an exponential smoothing with the holt-winters seasonal method or with an ARIMA model, but in order to show all the possible models. First to start with the forecast we have split the dataset in training set at test set.

Simple Exponential Smoothing: This method is obviously not adapted for our time series because it is not good for data with trend or seasonality and in fact the prediction is totally wrong. This model gives the greater weights to the last observation and in our case the last observations are all equal to zero, that is the reason why we obtain a straight line close to zero.

Holt's Linear Trend Method: Also this model is not a good choice in our case and the prediction is equal to the prediction made with the simple exponential smoothing.

Holt-Winters Seasonal Method: This method works for us. In fact, it allows the seasonality and the trend that is perfect for us.

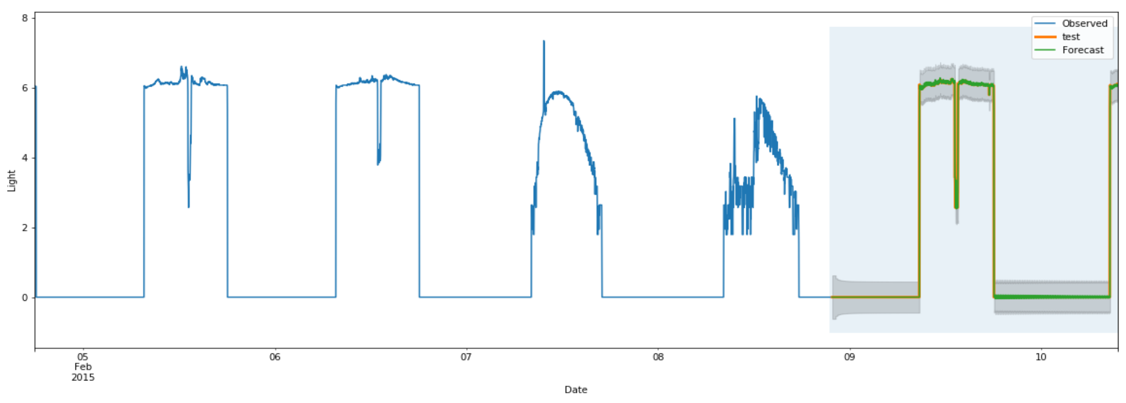


42- Forecast with Holt-Winters Seasonal Method

As we can see the prediction is pretty good, with just a little error about the length of the of the peak of light. These are the errors of the prediction:

```
MAE 0.333
RMSE 1.307
MAD 0.004
R2 0.772
MAPE 0.729
```

ARIMA: The arima model is the one that give us the best results of the forecasting for 'Light'.
First ,we have run a MLE for the estimation of the parameters by minimizing the AIC. The parameters order and seasonal order that we have obtained are: order (1,0,0) and seasonal order (0,1,1,12).



43-Forecasting with ARIMA

As we can see the result with the ARIMA are very good, the grey part is added for a confidence of the prediction. Here it's clear that the results are optimal, and the prediction is very close to the original one. And these are the summary and the result of the model. The errors of this ARIMA model are:

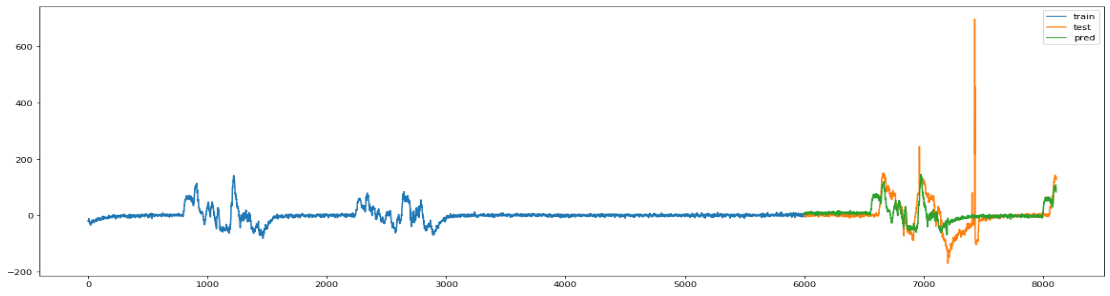
```
MAE 0.024
RMSE 0.225
MAD 0.006
R2 0.993
```

CO2: For CO2 we will be shorter. As for the 'Light' time series the first thing that we went to check was the stationarity of the time series. Also in that case the original time series is not stationary so in order to make it become we have applied some transformation. The modified time series that resulted stationary was the rolled one with a window equal to 30. This time series in fact, according to the Dickey-Fuller test is stationary with also a very low p-value.

```
      Results of Dickey-Fuller Test:
Test Statistic                 -6.770467e+00
p-value                         2.649885e-09
#Lags Used                      2.400000e+01
Number of Observations Used     8.088000e+03
Critical Value (1%)            -3.431159e+00
Critical Value (5%)            -2.861897e+00
Critical Value (10%)           -2.566960e+00
```

So then as usual we have splitted the dataset in training and test sets in order to be able to predict it. This time we won't present all the models but only the ones that give good results.

Holt-Winters Seasonal Method: In this time series we have both trend and seasonality, so this is a right case for this type of model.



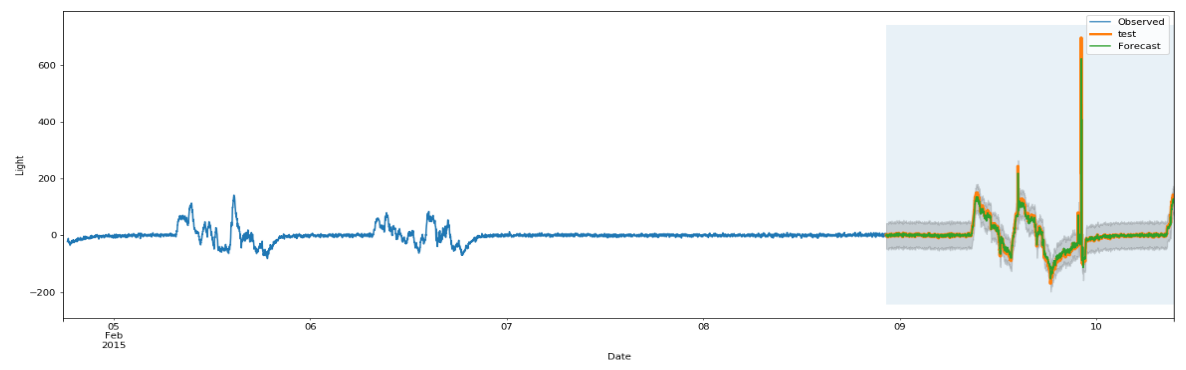
44-Forecasting with Holt-Winters Seasonal Method

The results are not so bad, also considering that the part that we are predicting is, also in the original time series, a little different than the usual, in fact, we have a peak in that part.

The errors of this forecasting are:

```
MAE 27.318
RMSE 44.259
MAD 12.395
R2 0.331
```

ARIMA: Also in this case the ARIMA model is the best one. The order and seasonal order are, also this time, estimated with a MLE and the result of the forecast is:



45-Froecasting with ARIMA

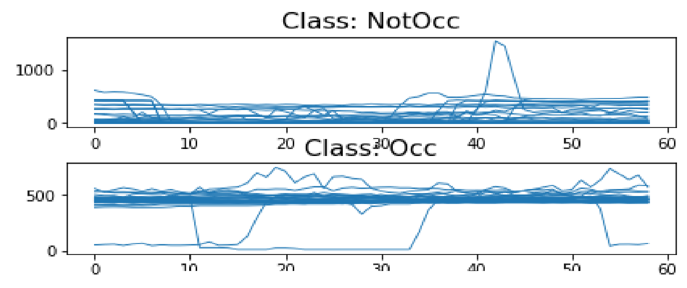
The prediction this time is not so perfect, but we can say that is really good, the errors in that case are a little greater than the ones saw with 'Light' but are better than the Holt-Winters Seasonal Method:

```
MAE 5.952
RMSE 23.189
MAD 3.150
R2 0.816
```

Classification of Time Series

For the classification task we have used different classificatory on different types of datasets. For simplicity we will show only some type of classifier used during this task specifying which kind of dataset (multivariate or univariate) has been used.

Shapelet classifier: In this classifier we have used the same dataset used for the shapelets discovering or for clustering. In this case we have done this classification on a univariate dataset using only 'Light'. So, the dataset has been splitted on hour and for the decision of the class of each time series we have set to each time series the majority class in that hour in the original dataset. The same thing has been done with the test set. So, after calculating the shapelet we have run the shapelet model with a weight regularizer equal to 0.3. With this classifier the result is perfect, all the time series are classified correctly, so with accuracy equal to 1. From the graph we can see all the time series based on the class predicted with the ShapeletsModel.



46- Time Series Classified

With this time series splitted by hour we have also trained some classical classifiers like K Neighbors Classifier and the Decision Tree Classifier.
The results with these classifiers are equal to each other, They are not so perfect as for the previous classifier but are very good. In fact, only 1 time series has been misclassified as a reason for which we can also say that these classifiers work well on that dataset.

	f1-score	support	accuracy
0	0,982456	28	0,976744
1	0,965517	15	

	Predicted Occupancy=0	Predicted Occupancy=1
Occupancy=0	28	0
Occupancy=1	1	14

So, in conclusion about these classifiers we can say that is a good choice, but for run it with the shapelets model and with the classical classifier. The difference between the three different classifiers tested are minimal and the results are quite similar.

Convolutional Neural Network: The CNN classifier has been tried on two types of datasets. The first test has been done on the univariate dataset of ‘Light’ splitted on hour and the second test on the original multivariate dataset using only the attribute relevant (Light, CO2, Humidity and Temperature) and always splitted on the hour. Let’s start with the univariate. In that case we have used a neural network 4 layers, the first three with size equal to 100, 150, 200 and the last one with size equal to the number of the output that is 2. Then we ran the model with an early stopping and 50 epochs.

	f1-score	support	accuracy
0	0,962962	28	0,953488
1	0,937508	15	

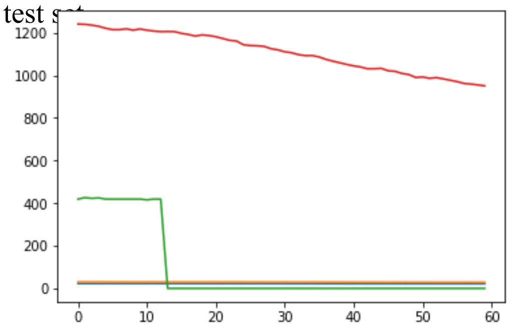
	Predicted Occupanc y=0	Predicted Occupanc y=1
Occupanc y=0	26	2
Occupanc y=1	0	15

Also in that case we have obtained good outcome, just two misclassified time series. For the multivariate the things are similar. In this case, the dataset has been splitted on hour and the features relevant in the time series are ‘Light’, ‘CO2’, ‘Humidity’ and ‘Temperature’, all the other features are or irrelevant, like ‘HumidityRatio’. Running this algorithm in this way, we have obtained good results. The model has been initialized with 4 layers, the first three are with size equal to 200, 300 and 150 with the last layer with size 2, the number of input that we need. Then we have run the model with an early stopping on the validation loss, with patience equal to 10 and 50 epochs.

	f1-score	support	accuracy
0	0,981818	28	0,976744
1	0,967741	15	

	Predicted Occupanc y=0	Predicted Occupanc y=1
Occupanc y=0	27	1
Occupanc y=1	0	15

So, as we can see from the tables above, just one hour time series has been misclassified out of the 43 in the test s



47- Misclassified Time serie

Here we show the hour that has been misclassified, probably all the problems come from the fact that the label of the hour is set from the majority class in that hour. In the hour above the class is 0, as we can see also from the green line that represents the light and, in that case, is close to zero. Probably the misclassification is given from the combination of a middle way of light that for the 25% of the hour is a light typical of a work time and the high value of CO2, another thing typical of a moment when the occupancy is equal to 1.

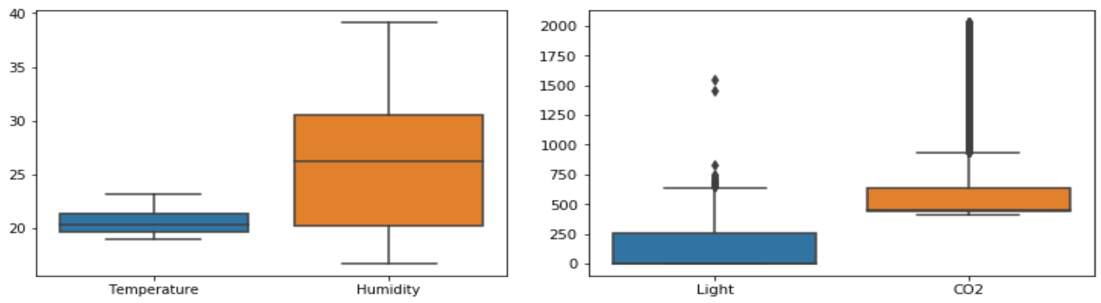
Long Short-Term Memory: We will show the results of this classifier applied on all the important features of the dataset splitted on hour, so the same dataset used in the last CNN showed, the multivariate dataset. The decision of the right number and size of the layers took a lot of time, the final dimension of that is: 3 layers, the first two with dimensions equal to 230, 100 and the last equal to number of outputs, 2. The model has been run with an early stopping on the validation loss, patience equal to 5 and 10 epochs. The model gives us a perfect result, all the hour time series are classified correctly, so we have an accuracy equal to 1.

In conclusion of the classification we can easily say all the classifiers give us good performance, both for univariate and multivariate datasets. The worst classifier returns just 2 misclassified hours. But, in order to estimate the best classifiers, we can say that for the univariate dataset the best one is the ShapeletsModel and for the multivariate dataset is the LSTM, both with a perfect accuracy, 1.

Outlier detection

In this task, our purpose is to find the outliers present in the 'hour' dataset using it with the whole number of features except 'Humidity Ratio' which is strictly related to 'Humidity'. We have used 4 different models to discover these outliers. The first one is the simplest, just a simple visual approach using one boxplot for each feature. Then we have used two models for the density based, DBSCAN and LOF, one model for the angle based, ABOD, and one for the distance based, KNN. For each model we will the distribution of the outliers, and then we will see a comparison between KNN and ABOD.

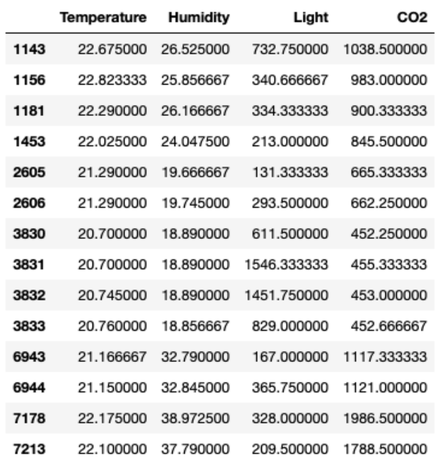
Boxplot: The boxplots are the simplest way to see the outliers. This method is just visual.



48- Boxplot

How we can see, for 'Temperature' and 'Humidity' we don't see any type of outliers. We can't say the same thing about 'Light' and 'CO2'. Starting from 'Light' we can see that the distribution of the value is mostly between 0 and 250 with a maximum around 650. In this boxplot it's clear that we can find some outliers, most of them are close to the maximum, but 2 outliers are completely out of the context, probably they correspond to the anomaly found in the part about the anomaly detection. In fact, while the maximum of the boxplot is around 650 these two outliers have value around 1500, clearly out of the normal distribution. About 'CO2', in that case we have a lot of outliers, according to the boxplot, the maximum of it is around 1000 and so, always according to the graph, all the values of 'CO2' over 1000 are outliers. This thing seems to be quite strange and not clear, as we remember, the last peak of the time series is widely over the maximum of the boxplot.

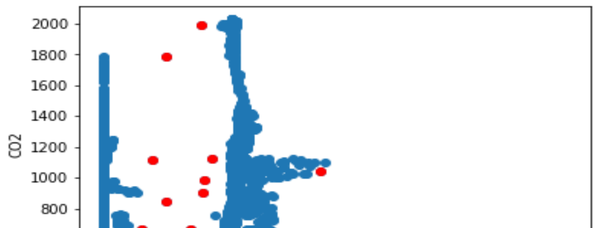
DBSCAN: DBSCAN is a density-based approach for discovering outliers. Running this algorithm with 50 epochs and min_samples=4 we have obtained 14 missing values.



	Temperature	Humidity	Light	CO2
1143	22.675000	26.525000	732.750000	1038.500000
1156	22.823333	25.856667	340.666667	983.000000
1181	22.290000	26.166667	334.333333	900.333333
1453	22.025000	24.047500	213.000000	845.500000
2605	21.290000	19.666667	131.333333	665.333333
2606	21.290000	19.745000	293.500000	662.250000
3830	20.700000	18.890000	611.500000	452.250000
3831	20.700000	18.890000	1546.333333	455.333333
3832	20.745000	18.890000	1451.750000	453.000000
3833	20.760000	18.856667	829.000000	452.666667
6943	21.166667	32.790000	167.000000	1117.333333
6944	21.150000	32.845000	365.750000	1121.000000
7178	22.175000	38.972500	328.000000	1986.500000
7213	22.100000	37.790000	209.500000	1788.500000

These are the outliers founded with the DBSCAN. Analyzing these we can easily find some congruences with the boxplots. For example, id 3831 and 3832 are the big outliers founded before in light. Also, in CO2 are identified some outliers both here that in the boxplots, like id 1143 and the last four rows. But this time the outliers are not reliable just to one feature but are about all the features, so in order to understand better the distribution of these outliers is important to see them in a plan.

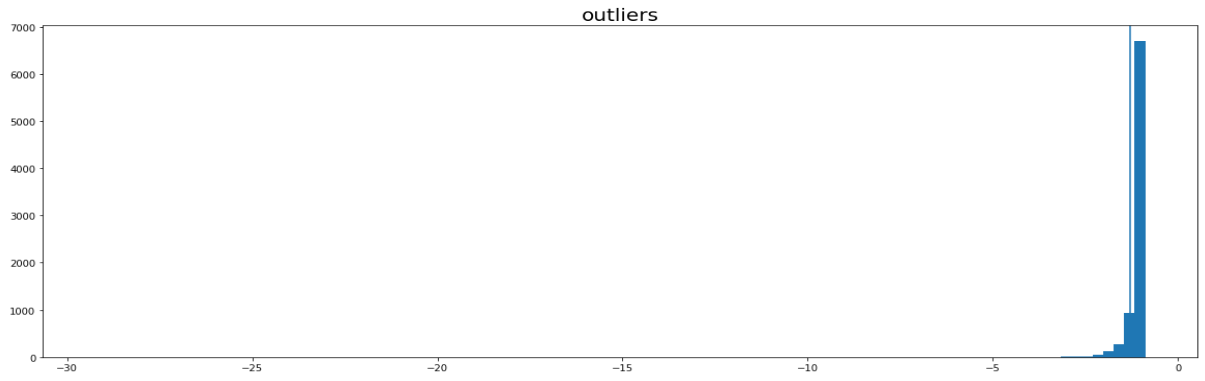
This is the plot of the values of 'Light' and 'CO2' and, colored in red, we can see the outliers found. This combination of features for the plot is the most important between all the combinations, also because we have already seen that both 'Temperature' and 'Humidity' have no one outliers. Here we can see that this time, also if some red points are in the normal 'range' of the feature, compared with the normal distribution of the two attributes are outliers.



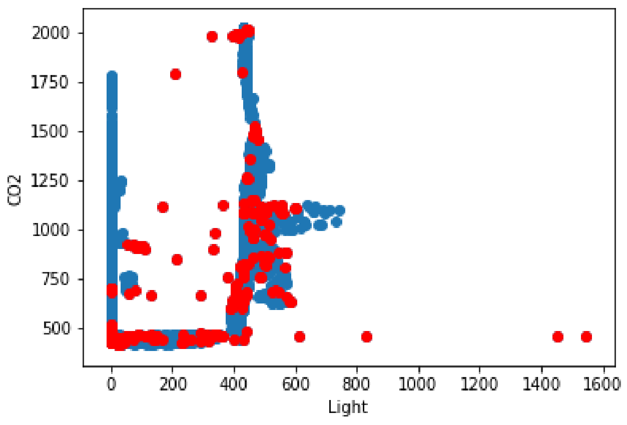
This algorithm gives us good results, all the values returned seem to be true outliers because they are very isolated values.

49-Outlier distribution DBSCAN

LOF: LOF is another density-based approach for discovering the outliers. In that case, the outliers founded by this algorithm are 815, whatever are the number of neighbor septate. This number seems to be too big compared to the other methods seen so far. So, let's see the negative outlier factor distribution of the dataset.

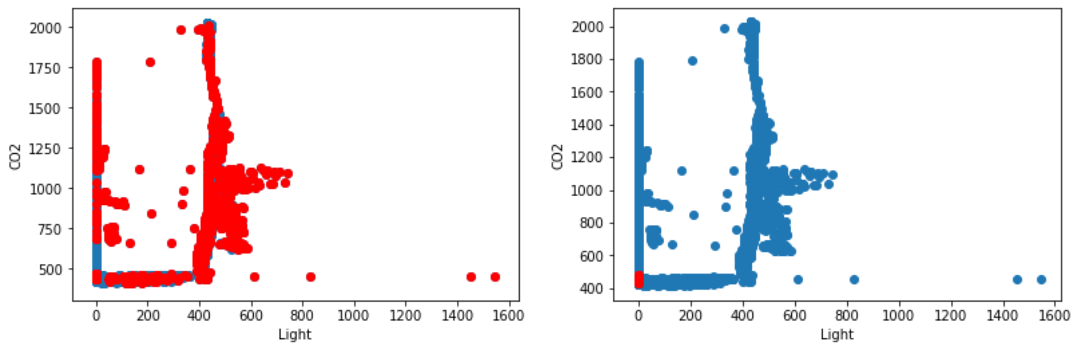


50-Outlier factor distribution



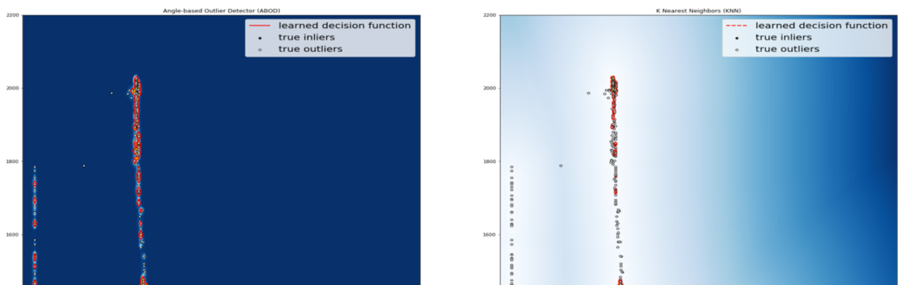
So, watching this graph we can assume that the greatest part of the outliers is really close to the threshold septet and so are not so away from the normal distribution.
In fact, this situation is easily identified in the next graph where we went to see the distribution of
all
the values with the outliers identified marked in red.
It is quite obvious that the greatest part of the outliers seems to be in the rest of the group, also seeing it with other features the outliers found are in the normal distribution.

ABOD: ABOD is an angle-based method for the detection of the outliers. With this algorithm we have obtained 797 outliers. Reducing it we have obtained a few numbers of outliers, 43 instead of 797, but, plotting the scatter plot we see that also using all the reduced number of outliers these are not so reliable to which are true outliers.



51- Reduced and not reduced outlier distribution

KNN



KNN is a distance-based method for discovering the outliers. The number of neighbors used in this case is 10. For seeing the result we have decided to confront it

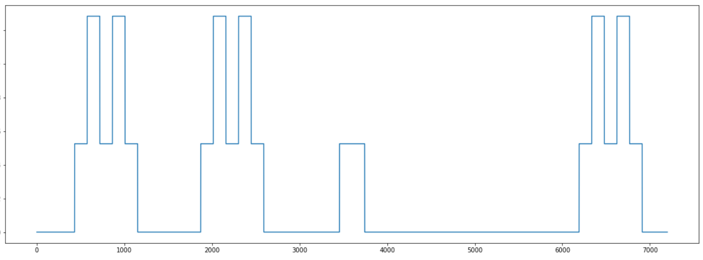
with ABOD, in fact we have obtained a number of outliers very close to the one obtained with ABOD.
In fact, as for ABOD, the number of outliers seems to be too big and not so relevant for our purpose. Seeing some of the previous results, like DBSCAN, the outliers plotted in 2-D using ‘Light’ and ‘CO2’ are really messed up with the inliers and this situation doesn’t change so much changing the feature for the plot.

52- KNN and ABOD comparison

In conclusion about the outlier detection we can say that the best method is the DBSCAN, in fact it finds a low number of outliers compared to the other ones. Mostly, all the other methods consider as outlier the final peak of CO2, due to the fact that this peak is really bigger than the other, but for us it is not good, the CO2 is strongly influenced by the number of the people in the office, fact that give us the idea the these values are not outliers and the DBSCAN give us the best result.

Frequent Pattern Mining

In this part we went to check eventually frequent pattern in our dataset. In order to do this, it is basic to divide the dataset in more than one time series and to convert them in a discrete format. Our aim is to find frequent pattern in these time series so the division for our would have been useless, therefore the split chosen for obtain more time series has been the one for days. The type of approximation chosen was the symbolic aggregate approximation (SAX).



53- Time serie discretized with SAX

Here we see the time series reconstructed after the division for days and the consequent approximation with SAX. The following pattern mining has been applied on the splitted dataset; the graph is only for discover how the time series have been converted in a discrete form.
Now, we will show just some frequent pattern discovered with a length maximum of 4 and with two types of confidence, 100% and 80%.
With confidence equal to 100% we have found just one pattern, it is the one composed by four zero, {0 ,0 ,0 ,0} which, convert in the graph means the value at 0. We are looking at the discretized graph of ‘Light’, so this pattern means that in every day studied there is almost four time where the light is equal to zero. This is very obvious, rather, in the reality, choosing also longer pattern we can see that this pattern of zero is longer. This pattern corresponds to the fact that during the night the light is always zero and splitting the dataset by the day in all of them we can find this type of situation. Whit confidence equal to 80% we have obtained 2 different frequent patterns. The first one is {1,1,1,1} which correspond in a situation where there is enough light for make it as a work-moment; this 1 in the graph means the point 0.52. This pattern may be tricky, the third day seems like to have a work-moment situation when in the reality it isn’t. The second pattern is: {1,1,0,0}. This means that in 4 days out of 5 we have situation where we have both light greater than zero and light equal to zero. This is quite obvious also considering the two different patterns previously analyzed.