UNIVERSITÀ DI PISA

DATA MINING II
ADVANCED TOPICS AND APPLICATIONS

Air Quality

Matteo Galeazzi
Irene Parlanti
Chiara Spampinato

November 2020

Contents

1 Introduction

The provided dataset is about Air Quality, a measure to see how polluted or how clean the air is. Data refers to the responses of multisensor devices deployed within a city of Northern Italy during a period of a whole year. Since the dataset has not a target variable for the Basic and Advanced classifiers, we decided to set it with *"is weekend"*. By analysing parameters like the temperature, nitrogen oxides or carbon monoxide, we had to verify whether a record of the dateset was made during weekend or not. As regards the Time Series Classification we used another target variable that predicts, by seeing gas values, if it's day or night.

2 Data Understanding & Data Preparation

The dataset contains 9358 instances of hourly averaged responses from an array of 5 metal oxide chemical sensors embedded in an Air Quality Chemical Multisensor Device. It is composed of 15 attributes:

- Date (DD/MM/YYYY);
- Time (HH.MM.SS);
- CO(GT), true hourly averaged concentration CO in mg/m^3;
- PT08.S1, (tin oxide) hourly averaged sensor response;
- NMHC(GT), true hourly averaged overall Non Metanic HydroCarbons concentration in microg/m^3;
- C6H6(GT), true hourly averaged Benzene concentration in microg/m^3;
- PT08.S2, (titania) hourly averaged sensor response;
- NOx, true hourly averaged concentration in ppb;
- NO2, True hourly averaged NO2 concentration in microg/m^3;
- PT08.S3, (tungsten oxide) hourly averaged sensor response;
- PT08.S4, (tungsten oxide) hourly averaged sensor response;
- PT08.S5, (indium oxide) hourly averaged sensor response;
- T, Temperature in C°;
- RH, Relative Humidity (%);
- AH, Absolute Humidity.

By analysing the dataset we noticed two attributes that were not present in the dictionary. Since that columns, (*Unnamed 15* and *Unnamed 16*), contain only NaN values, we decided to drop them, as well as the **NMHC(GT)** variable which has many missing values too. After that, we fixed the remaining missing values, which were tagged with -200, by replacing this value with the mean of the every single column.

2.1 Target Variable

As far as it concerns the target variable, at first we created a new variable, **Weekday**, which assigns a number from 0 to 6 to the days of the week (where Monday=0 and Sunday=6). Then, we created another attribute, **Weekend**, our target variable, which is true when weekday is equal to 5 or 6. As we can see from Figure 1 in our dataset there are more Workdays, 73.1%, labeled with 0, than Weekend 28.7%, labeled with 1, as we could easily expect.
After that, we decided to analyze and visualize attributes distribution with respect to the target variable in order to see how gas distribution varies during weekends. For instance, as shown from Figure 2, it's commonly possible to detect a threshold from which it's likely to identify whether it's weekend or not.

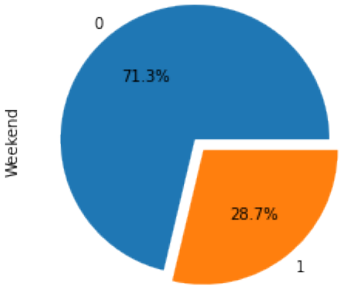

Figure 1: Weekend Distribution

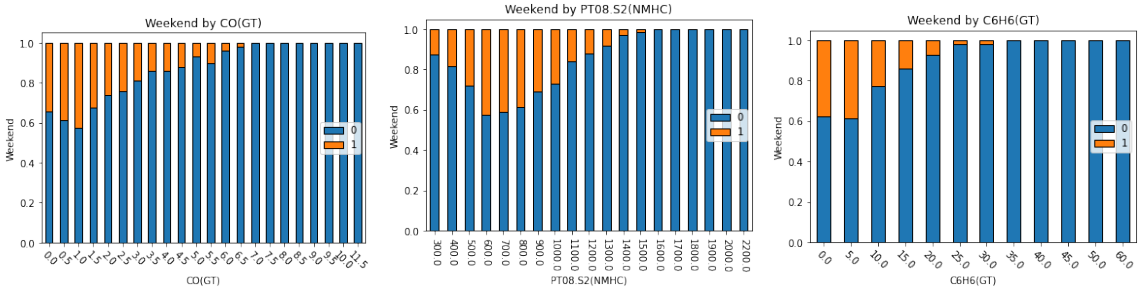

Figure 2: Gas distribution by Weekend

The Correlation matrix displayed in Figure 3 shows us no meaningful correlations among the attributes and our target variable therefore we can't distinguish any particular attributes that help us in identifying if it's weekend or not.

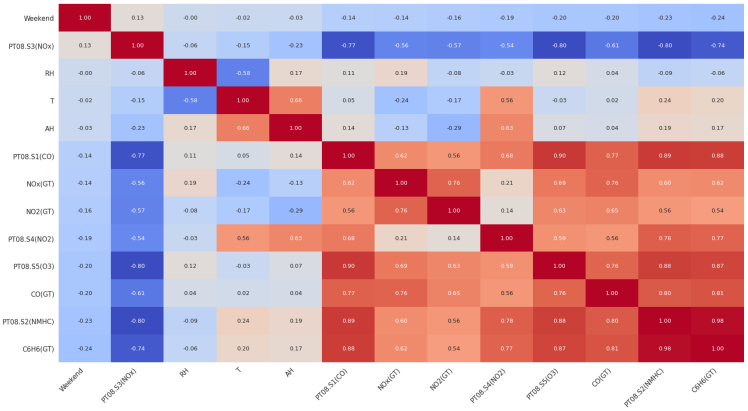

Figure 3: Correlation Matrix

Since the correlations among gas and weekend variable were not satisfying, and, as we will see the basic/advanced classifiers did not performed always so well, in the process of time series classification we considered an alternative target variable in order to maximize the performances that will be explained in the specif Section 8.

3 Basic Classifiers

In the following section, we will describe the methodologies and the algorithms used during the classification and the obtained results. The main goal of this task was to predict the variable called **Weekend**.

3.1 K-Nearest Neighbors

In order to find the best value of **K**, we tried two different procedures. First of all, we plotted K in terms of Accuracy, trying different value of K $\in$ [1,50]. The results of K=1 and K=3 were pretty good, but when plotting the error rate, the lowest value was shown by K=1, as shown in the Figure 4.

Figure 4: Error rate Vs K Value

We tried both weights=distance and uniform but the best result was given by a combination of the algorithm *Ball Tree* and *weights=distance*, which was predictable since the large dimension of the dataset.

The Cross Validation provided an Accuracy of 0.9707 (+/- 0.007) and a F1-Score of 0.9646 (+/- 0.009). The results are displayed in Figure 5. As showed in the plot, we got a good result, with a ROC AUC score of 0.99.

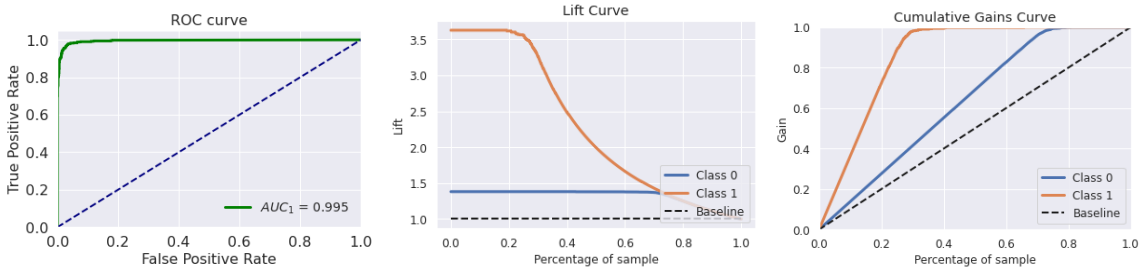

Figure 5: Results obtained with k-NN

3.2 Naive Bayes

The Naive Bayes algorithm consists in estimating the probability density of the features by making the assumption that continuous values associated with each class are distributed according to a normal distribution. For this analysis we decided to split the classification in 3 parts. The first attempt was made with all the numerical attributes in the dataset and the results are displayed in Table 1.

The second one was made with a subset of numerical attributes that contains only the main gas of the dataset, which are: 'CO(GT)', 'C6H6(GT)', 'NOx(GT)','NO2(GT)'. Finally, we also did a prediction with categorical values, but since we only have 'Month' and 'Hour', created by us using the Date and Time variables, the prediction did not work very well.

	All Attributes	Subset
Accuracy	0.94	0.95
Recall	0.94	0.95
F1	0.94	0.95
Precision	0.94	0.95
ROC	0.93	0.94

Table 1: Naive Bayes Results

As we can see in the graphs below we have the cumulative gains and ROC curves referring to the analysis of all numerical attributes with following results:

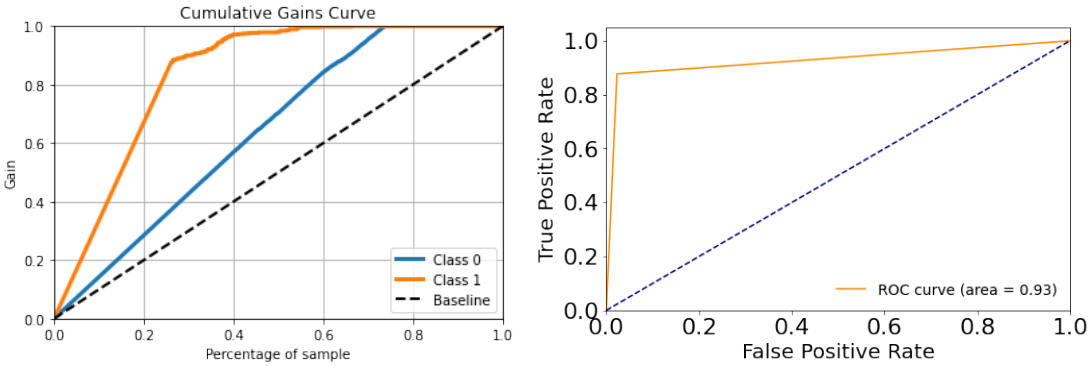

Figure 6: Naive Bayes numerical attributes results

3.3 Logistic Regression

In this section we will approach the Logistic Regression and we will use only the numerical attributes of the dataset after having standardized them.

To evaluate the parameters for the logistic regression we used Logistic Regression algorithm with the Cross-Validation (CV) approach. The hyperparameters performances were estimated by taking into account the Accuracy value and tested over a validation set of amplitude = 10, according to the 10-fold Crossvalidation method. The results obtained are not so good because in term of Accuracy we have 0.7345 and in term of ROC curve value we have 0.596. Results are shown in Table 2 and in Figure 8:

	Not Weekend(0)	Weekend(1)
Precision	0.75	0.61
Recall	0.93	0.26
F1	0.83	0.37

Table 2: Results

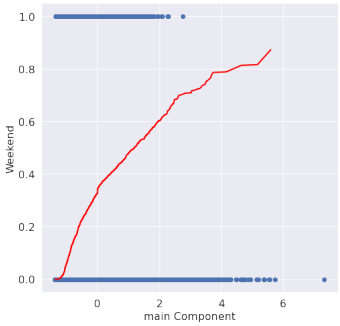

Figure 7: Main component graph

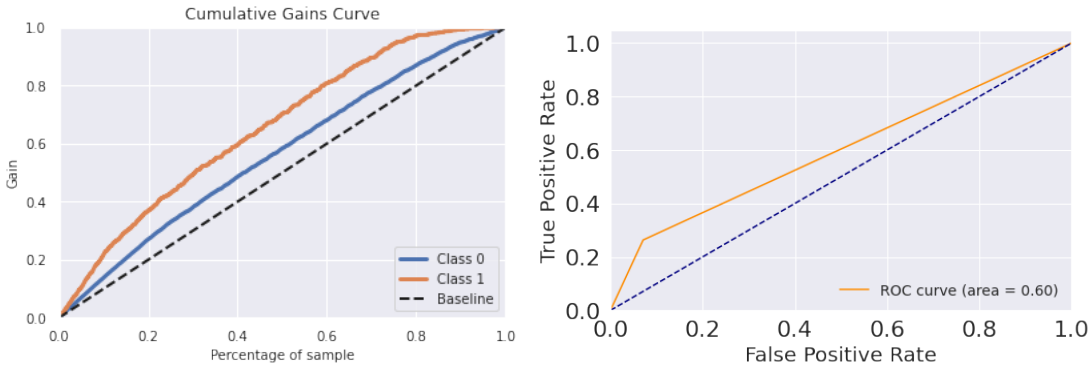

Figure 8: Logistic regression numerical attributes results

3.4 Decision Tree

For the implementation of Decision Tree algorithm we decided to apply some transformations to our dataset dropping the variables 'Month' and 'Weekday' and replacing them with only the 'Hour' variable represented by 24 columns, one for each hour. At this point, in order to find the best combination of parameters, we decided to consider and test the following:

- min_samples_leaf: [5,100,10,10]
- max_depth: [9,5,1,7]

The best Accuracy score resulted from min_samples_leaf=5 in combination with max_depth=9 with an accuracy of 0.76 and a ROC value of 0.64.

Nevertheless, for the purpose of finding the best parameter setting we also decided to try GridSearch algorithm selecting both criteria 'gini' and entropy and balanced class weight. The best parameters selected by the GridSearch approach are: class_weight='balanced', criterion='gini', max_depth=9, max_features=None, max_leaf_nodes=None, min_samples_leaf=10, min_samples_split=2, splitter='best'. The result for this algorithm is pretty the same of the previous one with an accuracy of 0.68 and a ROC value of 0.70.

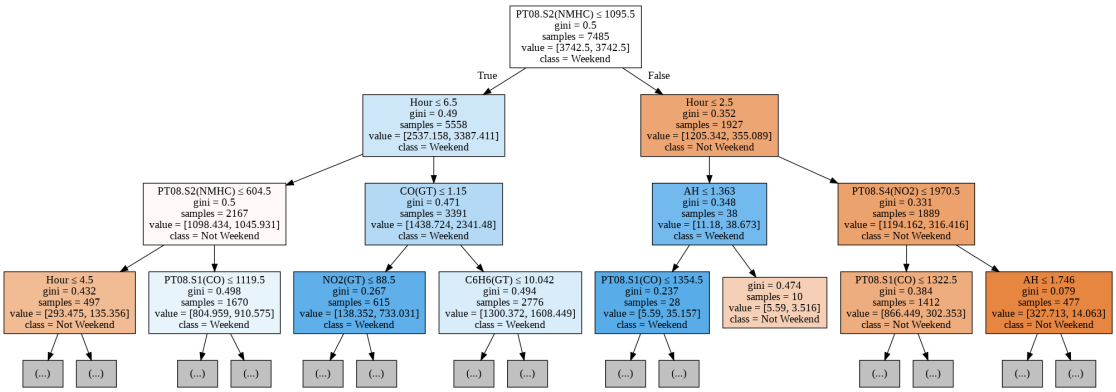

Figure 9: DecisionTree with GridSearch

4 Dimensionality Reduction

In this paragraph we will approach to the dimensionality reduction, where our aim is to understand if our basic classifiers can have better performances on a reduced dataset. This approach could help us in obtaining a low-dimensional representation that retains some meaningful properties of the original dataset.

In order to transform the data we used two approaches:

- Feature Selection:
  - Variance thereshold;
  - Univariate feature selection;
- Feature Projection.

4.1 Feature Selection

Feature selection consists of two approaches such as Variance threshold and Univariate feature selection. The former deals with a smaller number of records of the original dataset while the latter makes a selection of a number of features in order to find the most representative.

As it concerns the Variance Threshold approach we set a threshold of 0.5 so that the attributes whose variance did not meet this threshold have been removed. Nevertheless, we did not observe better performances in any basic classifiers.

Univariate feature selection approach, instead, was implement considering from a maximum number of 10 up to 1 features with the SelectKBest algorithm, decreasing the number of features at every iteration. In this case too, we couldn't appreciate better results, in particular KNN and Decision Tree classifiers registered significant losses in terms of accuracy while the other two basic classifiers had similar results to the analysis over the original dataset.

In order to better illustrate data variability we decided to implement ANOVA Test whose result is shown in Figure 10:

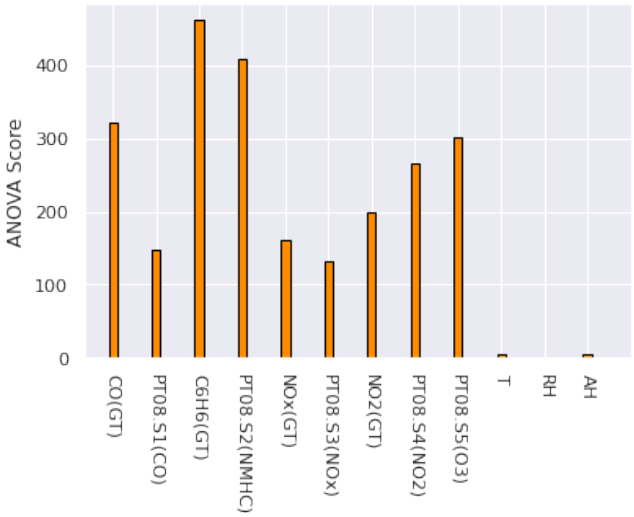

Figure 10: ANOVA Test

The C6H6 (GT) attribute appears to be the most significant in the dataset, or rather, the attribute that explains most of the variability of the data, followed respectively by PT08.S2(MNHC), CO(GT) and PT08.S5(O3).

4.2 Feature projection

Using Principal Component Analysis (PCA) for dimensionality reduction involves zeroing out one or more of the smallest principal components, resulting in a lower-dimensional projection of the data that preserves the maximal data variance.

Regardless of the classifier, the training set representation after PCA and the fraction of variance are shown in Figure 11 :

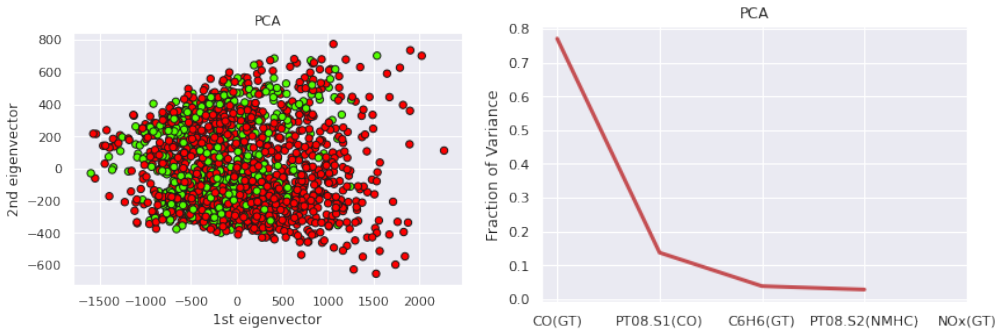

Figure 11: PCA

5 Imbalanced learning

In our Dataset the records are already imbalanced. In fact, as we can see in the Subsection 2.1, there are more Workdays than Weekend. We decided to increase the original imbalance, making the weekend distribution reach the %96 by deleting some "Weekend" records. In order to drop always the same records we fixed a predetermined seed. In the table below we want to show the basic classifiers results on this imbalanced dataset.

	KNN	Naive	DecTree	LogRegress
Accuracy	0.98	0.96	0.95	0.97
F1	0.96	0.95	0.95	0.96
Precision	0.95	0.94	0.95	0.94
Recall	0.98	0.97	0.95	0.97

Table 3: Basic classifiers results on imbalanced dataset

After, we tried to rebalance the dataset using both Undersampling and Oversampling techniques. As far as it concerns **Undersampling**, we implemented methods such as Random Undersampling and Condensed Nearest Neighbor (CNN) and as regards **Oversampling**, we tried SMOTE and Random Oversampling. Then, we executed every basic classifier again on the rebalanced dataset. Among all methods implemented, only **CNN** had slightly better results in every case. Results are displayed in the table below.

	KNN	Naive	DecTree	LogRegress
Accuracy	0.97	0.97	0.80	0.97
F1	0.96	0.95	0.87	0.96
Precision	0.96	0.94	0.95	0.94
Recall	0.97	0.97	0.80	0.97

Table 4: Basic classifiers results with CNN

6 Linear Regression

In this section we will approach the Linear Regression algorithm and in order to evaluate the performances of each model we'll take into consideration three evaluation metrics, which are:

- R^2, coefficient of determination, that defines how much variability of Y variable the model can explain;
- MSE, Mean Squared Error, that computes the mean square error between the observed values of Y and the values of Y estimated by the regression model;
- MAE, Mean Absolute Error, that evaluates the absolute error between observed and estimated or predicted Y values.

The models that we have considered for our Linear Regression analysis are explained in the following sections.

6.1 Two-Dimensional Linear Regression

For the analysis of the linear regression in two dimensions we have to choose two attributes. Our choice was based on the results of the PCA and the ANOVA score mixed with our personal considerations based on evidence. In fact, we selected the attribute **CO(GT)** because it has a good ANOVA score and also because it is released in all kind of incomplete combustion. In particular, it is produced by all internal combustion vehicles (e.g. cars), but also by domestic heating systems and industrial systems, so it is a quite common gas. In the same combustion processes where CO(GT) is produced, we can find high values for the attribute **NOx(GT)**, so we decided to use the latter in pair with CO(GT) expecting a good correlation between them. Specifically, for the construction of the model we selected the attribute NOx(GT) as the dependent variable **Y**, and the CO(GT) attribute as independent variable **X**. The obtained model was:

$$Y = 5.71649039 + 112.03358291X \tag{1}$$

The model and related results are represented in the following chart:

	Results
R^2	0.581
MSE	15672.498
MAE	88.916

Table 5: 2 Dimension Linear Reg. Results

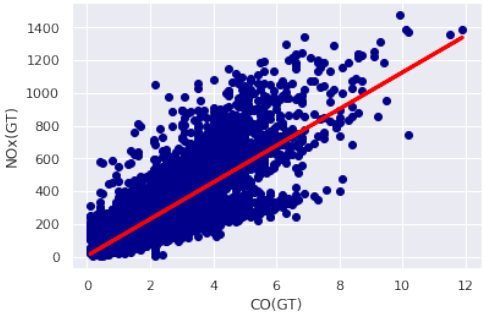

Figure 12: Linear Regression 2 attributes

6.2 Lasso, LassolarsCV and Ridge

Basically, LASSO regularization produces the identical model of the simple linear regression, giving also the same results in term of R^2, MSE and MAE. So we tried to implement the LassoLarsCV regularization. This method uses Lars algorithm and cross validation to estimate the parameters but,

despite that, results are unchanged. Finally, we implemented the Ridge model but we did not have significantly different results to show too.

6.3 Multiple Linear Regression

In order to implement the Multiple Linear Regression we used a dataset with **NOx(GT)** as dependent variable and all the other numerical attributes as independent variables.
The perfomances of Multiple Linear Regression model are significantly better in each of the evaluation metrics, as we might have expected: not only the R^2 does improve, but also the MSE and MAE values decrease.

	Results
R^2	0.815
MSE	7052.016
MAE	60.927

Table 6: Multiple Linear Reg.

7 Advanced Classifiers

In the present chapter we will analyze one by one what we have defined as "advanced classifiers", in particular we have:

- Support Vector Machine;
- Neural Networks;
- Ensamble Classifiers.

7.1 Support Vector Machine

Support Vector Machines are a supervised learning model that represents the decision boundary using a subset of the training examples, known as the support vectors. A SVM classifier builds a model that assigns new data points to one of the given categories. Thus, it can be viewed as a non-probabilistic binary linear classifier. We tried both Linear and Non Linear SVM.
In order to implement the SVM, at first we run the Standar Scaler on the numerical attributes and the GridSearchCV in regards to getting the best parameters. As a preliminary action, we executed the PCA on the dataset, to find the two more representative attributes. Afterwards we applied that algorithm, testing different values for **C**, the parameter representing the penalty of misclassifying training instances: C=1, 10, 100, 1000. Since Linear SVM didn't give us satisfying results we preferred to give deeper explanations of Non Linear SVM. We tried different combinations of Kernel functions such as Sigmoind, Polynomial and Radial Basis Function and different values of C, which are the same of Linear SVM. The best score was obtained with the **RBF** Kernel function and **C=1000**, which performed a 82% accuracy.

Accuracy	0.82
Precision	0.82
Recall	0.83
F1	0.82
ROC	0.77

Table 7: Results

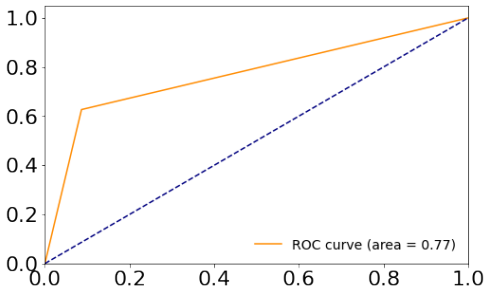

Figure 13: ROC Non Linear SVM

7.2 Neural Network

In the following section we will deal with the building of an Artificial Neural Network starting with the simplest Single Perceptron. After using the StandardScaler function on the dataset, we executed the Perceptron setting max_iter = 150, tol= 0.001 and eta0=0.01, obtaining an accuracy score of 0.72. Then, we also tried the Multilayer Perceptron considering different kind of activation functions: tahn, identity and logistic. The most performing results were obtained with these parameters:

- hidden_layer_sizes=(128, 64, 32);
- alpha=0.1;
- learning_rate='adaptive';
- activation='tanh';
- early_stopping=False;
- momentum=0.9;
- random_state=10.

Accuracy	0.84
Precision	0.84
Recall	0.84
F1	0.84

Table 8: Results

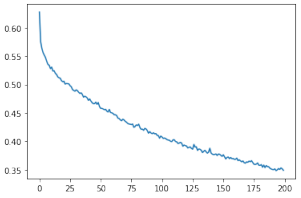

Figure 14: Loss curve

7.2.1 Deep Neural Network

After building a Neural Network, we also tried to build a more complex model, a **Deep Neural Network**, using the Keras library, in particular the Sequential and Dense models. We used two different architectures in order to define respectively two build model functions.

```
59/59 [==============================] - 0s 799us/step - loss: 0.3398 - accuracy: 0.8499
59/59 [==============================] - 0s 766us/step - loss: 0.3746 - accuracy: 0.8360
Loss 0.339812, Accuracy 0.849893
Loss 0.374640, Accuracy 0.836004
```

Figure 15: Evaluation results

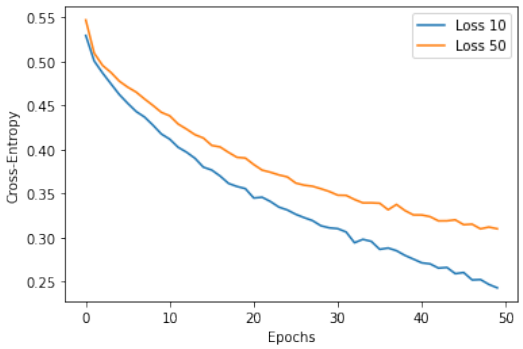

Figure 16: Evaluation results

As optimizer, the 'ADAM' parameter was chosen and the Loss Function was set to binary_crossentropy. As we can see from Figure 15 the two models gave us similar results in terms of accuracy and loss, we can't appreciate any significant improvement when changing the value of the Batch_size from 50 to 10.

Afterwards, we tried to build another model using the 60% of records to train the model and the 20% to validate it with Batch_size = 10 and Epochs = 100. With these parameters we obtained an accuracy of 0.80 and a loss score of 0.55. Then, we modified our models adding regularization parameters such as L2, Drop-Out and Early stopping but none of the 3 attempts lead to higher accuracy.

7.3 Ensemble Classifiers

Based on the wisdom of the crowd, ensemble methods' main feature lies on the combination of many different base classifiers in order to presumably get a more accurate final prediction. Within this section we have analyzed three ensemble models: Random Forest, Bagging and AdaBoost.

7.3.1 Random forest

Random forest is a kind of ensemble approach as it combines single predictions made by many decision trees and returns as output the class which is the mode of the resulted classes from each individual classification task.

In order to find the best combination of parameters we decided to consider and test the following:

- n_estimators: 1, 2, 5, 10, 20, 30, 40, 50, 55, 100;
- max_depth: 1, 4, 6, 8, 10;
- criterion: 'gini','entropy';
- min_samples_split: 2,4,5,10,20,50;
- min_samples_leaf: 1,2,4,5,10,20,35,50.

GridSearchCV approach was implemented so as to find the best parameters to use among all those above-mentioned. The performances of the selected hyperparameters and model was measured on a 5 amplitude validation set, following the 5-fold method, then evaluated taking into account the accuracy score. The best model turned out to have the following parameters:

- n_estimators: 100;
- max_depth: 10;
- criterion: 'gini';
- min_samples_split: 2;
- min_samples_leaf: 1.

The importance of attributes in the construction of the tree is shown in the Figure 17:

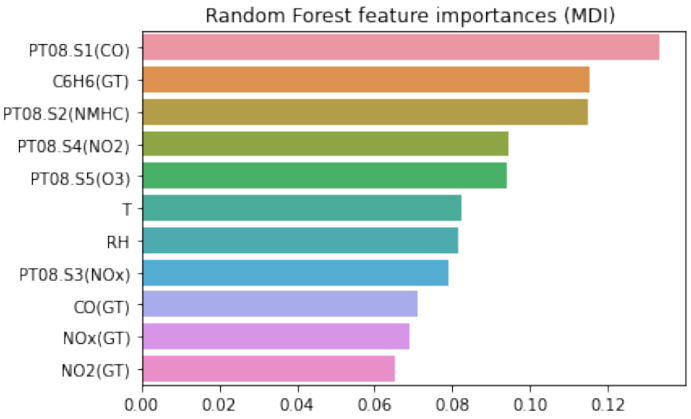

Figure 17: Random Forest Features Importance

In Table 9 and in the graph below in Figure 18 summarize the results of the Random Forest classifier in terms of accuracy, ROC, and F1 score:

Accuracy	0.81
F1	0.80
ROC	0.71

Table 9: Results

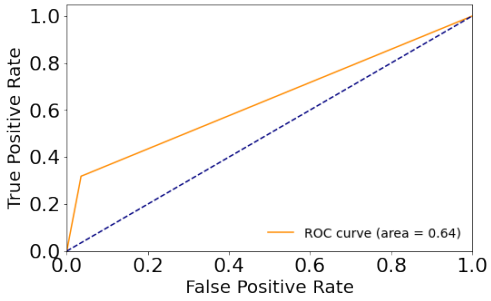

Figure 18: ROC Random Forest

In general, however, the model classifies the Weekend = 0 class better than the Weekend = 1 class. Then, in Figure 19 we can see the whole tree:

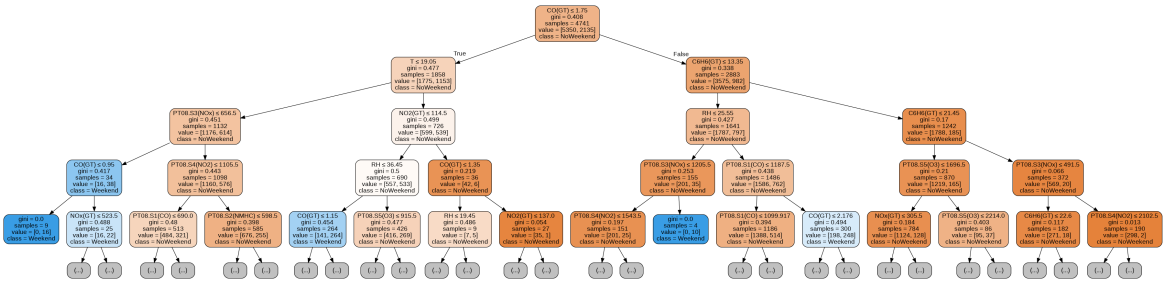

Figure 19: Random Forest Tree

7.3.2 AdaBoost

We decided to apply the AdaBoost model on the Random Forest classifier built earlier. The search for the best parameters was performed using the GridSearchCV like before with the same values discussed in the Random Forest section. For the hyperparameters of AdaBoost instead we tried the following values:

- n_estimators: [1, 2, 5, 10, 20, 30, 40, 50, 55, 100];
- learning_rate: [0.01, 0.05, 0.1, .5, 1.0],
- algorithm: ['SAMME.R', 'SAMME'].

Using the GridSearchCV on the AdaBoost parameters, the optimal values found are: n_estimators=100, learning_rate=0.5, and algorithm:'SAMME.R'. With these parameters we reached an Accuracy of 0.83, an F1 Score of 0.82 but we had poor results in terms of AUC as in the random forest classifier.

Furthermore we want to show in the following Figure 20 both the decision boundary for the Random Forest rating with AdaBoost and the distribution of the decision scores for the two classes. Samples with a score greater than 0 are classified as 1, the others as 0.

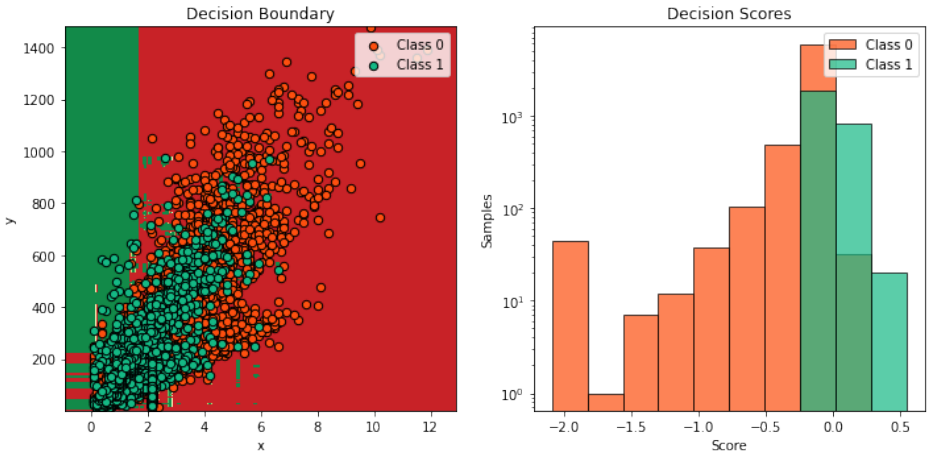

Figure 20: Decision Boundary and Decision Scores

7.3.3 Bagging

Since the Random Forest classifier performed better than the Decision tree classifier we decided to implement Bagging model on the Random Forest classifier built earlier. By using the GridSearchCV approach, it was decided to evaluate the following hyperparameters in order to find the best model in terms of accuracy:

- n_estimators: [1, 5, 10, 50, 100];
- max_samples: [1, 10, 50, 100,500,1000];
- max_features: [1,2,3,4,5,6,7,8,9,10,11].

What we expected was that the best model was that one with a higher number of samples, in fact the best parameters are the following: n_estimators = 100, max_samples = 1000, and max_features = 11. Nevertheless, the results of the implementation of bagging on Random Forest classifier are worse than the AdaBoost results.

In conclusion, we might state that, among all basic classifiers implemented for the purpose of the analysis, the best outcomes in terms of accuracy were reached by **K-Nearest Neighbors**, followed by **Naive Bayes** over numerical attributes only. On the other hand, we could not equally appreciate **Logistic Regression** and **Decision Tree** performances in predicting the target variable **Weekend**. As far as it concerns the carrying out of advanced classifiers, we might agree that all methods showed lower accuracy scores compared to that of basic classifiers. Basically, **Non linear Support Vector Machine** appeared to be more suitable to the dataset distribution, namely non linearly separable, so that it reached 0.82 of accuracy. **Neural Networks** mainly got to significant results, since both the implementation of a **Multi-Layer Perceptron** and a **Deep Neural Network** reached the 0.84 value of accuracy. Ultimately, the combination of many basic classifiers, for instance decision tree classifier, in **Random Forest** approach let us observe a slightly worse accuracy value of 0.81.

8 Time Series

In this section we will analyze the data as a time series thanks to the presence of temporal attributes, **Date** and **Time**. The Date attributes present records of a whole year, but for this study we will focus only on one month and some gas like CO(GT), NOx(GT), NO2(GT), PT08.S4(NO2) and PT08.S1(CO), as well as the temperature, T. Our choice was based on the Misa Report of 2004, that highlighted those gas as the responsible for health issues due to air pollution. We chose the month of October because it was considered a "standard" month regarding the urban traffic since it is not influenced by summer months or holidays in general. Setting a new variable as a combination of Date and Time, called **Datetime**, as index of the dataset, all features were separately analyzed as univariate temporal series.

8.1 Motifs, anomalies and shapelets

In this preliminary exploratory phase, we will compute the profile matrix for each variable of the dataset considering it as univariate time series, in order to discover possible motifs or anomalies in the series. Each individual Time series contains the data referring to October 2004, for each day. Since we have a huge amount of records, we will represent the data in 60 minute intervals (this is automatically done because we have data for each hour). First of all we can see in the following Figure 21 the trend of all variables over the time period taken into consideration:

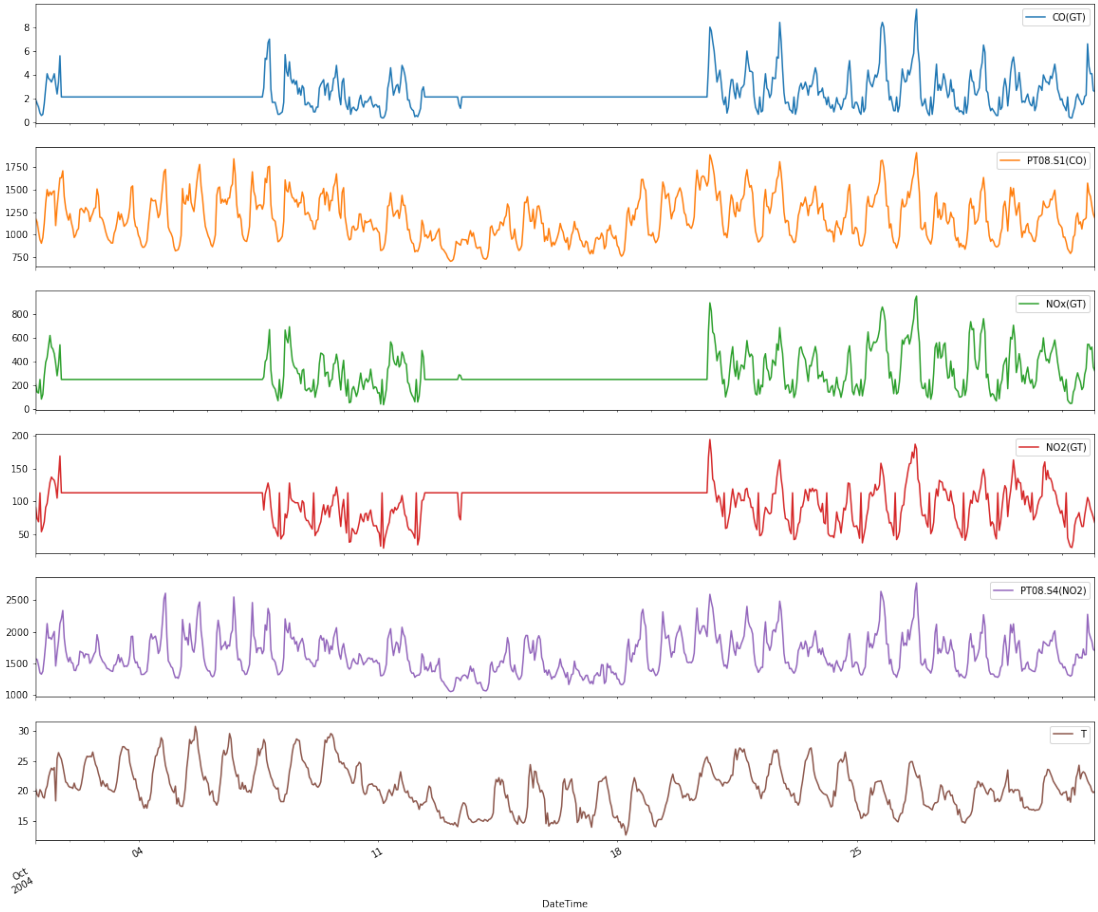

Figure 21: TS Trends

At this time, in order to do the analysis of motifs and anomalies we have to build the Matrix Profile. Before introducing the Matrix Profile it might be interesting to discuss it briefly. Since a

motif is a repeated pattern observable in a time series while a discord is an anomaly similarly visible in such a series, a certain Matrix Profile is a way to identify them. It has two primary components: a distance profile and profile index. So, the Matrix Profile stores the distances in Euclidean space meaning that a distance close to 0 is more similar to another sub-sequence in the time series while a distance far away from 0, say 100, is unlike to any other sub-sequence. Given the Matrix Profile computation, it is simple to find the top-K number of motifs or discords. In this sense we can consider the lowest distances as a signal to identify the motifs while the largest distances as to identify the discords.
To carry out the analysis we used the STOMP algorithm since it is the fastest among the alternatives. We decided to do the motif analysis and anomaly discovery for each features. For anomalies discovery we fixed the amount of anomalies to 15, a number of zones to exclude = 5 and a time window size=300. For each variable, in the period of the considered time, we obtained a motif and an anomaly zone, however, we will only report the results of the variable CO(GT) since it will also be used for further analysis:

- Motif: we have 1 motif [18, 169, 354] with distance = 17.805. This motif refers to 3 days, in particular to the 1-10-2004 at 18:00, to 08-10-2004 at 01:00 and to 15-10-2004 at 18:00;
- Anomalies: 15 Anomalies detected in the red zone of the graph, the period begins on 7-10-2004 at 10:00 and finishes on 11-10-2004 at 21:00;

The results are also displayed in the following Fig. 22

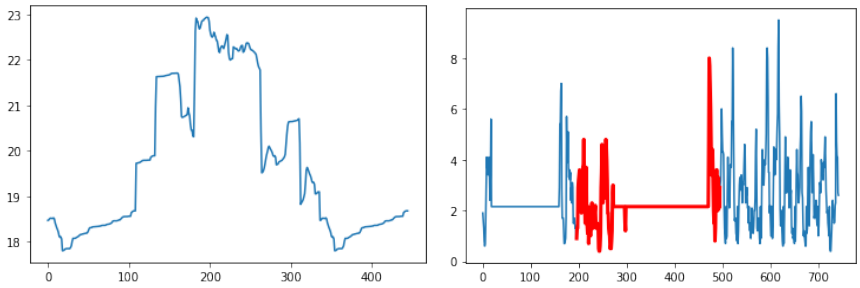

Figure 22: Motif and Anomalies on CO(GT)

8.2 Univariate Time Series Exploration

Since from motifs and discords analysis no variable came out clearly better than the others, the choice of **CO(GT)** was made according to the choices of the previous sections. Given some anomalies within the previous analysis of CO(GT) variable (see Section 8.1), we concentrated on the last week of October for univariate analysis. In the following Fig. 21 we displayed the trend of the variable CO(GT) over the time period taken into consideration:

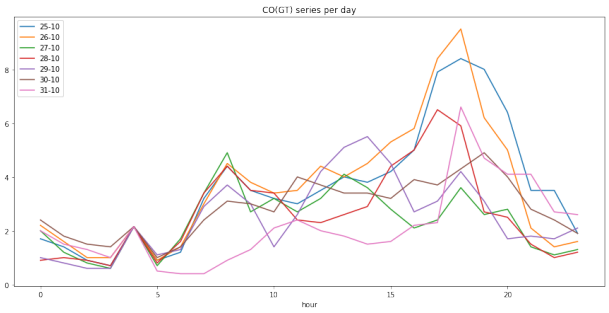

Figure 23: Univariate TS Trends

After, we proceeded to the reduction of the noise from the series and, using the moving average based approach, we searched for any trend and seasonality components, for which, however, we did not get significant results. As it will be shown later in the report, we also tried other approaches to identify trend and seasonality components that we will go deeper into the univariate time series classification section.

8.3 Clustering

In the present section we will discuss about clustering which tries to group different time series into different groups according to similarity measurement. For the purpose of the analysis we will focus only on CO(GT) variable taking into account the last week of October to avoid anomalies. We will work on the series without noise using three different techniques:

- Shape-based;
- Compressed-based;
- Approximation-based.

The evaluation metric we will base on for the evaluation of each techniques is the inertia value, that tells how far away the points within a cluster are. In Shape-based clustering we used the TimeSeriesKMeans algorithm which has been tested with different combinations of parameters: [n_clusters=2, verbose=1, metrics='euclidean', 'dtw'].
After that we moved on to the second technique, the Compressed-based one. In this case we noticed that the best algorithm was the DBscan algorithm with these parameters [eps=1, min_samples=5, metric='precomputed']. Finally, we approached the Appoximation-based technique with the TimeSeriesKMeans with these paramenters [n_clusters=2, metric='euclidean', 'dtw', max_iter=7]. According to inertia value, Approximation-based approach with DTW metric turned out to be the best one in clustering task. The results are displayed in the following table and in Figure 24:

Best Technique	Approximation based
Algorithm	TimeSeriesKMeans
Inertia	0.0422

Table 10: Results

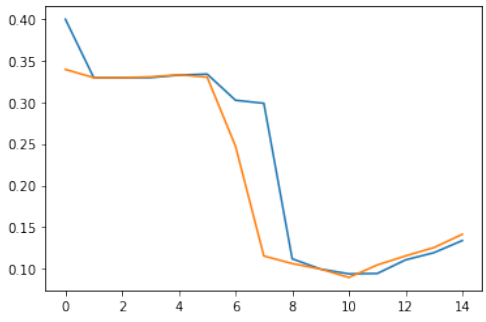

Figure 24: Approximation Based Clustering

8.4 Forecasting

The idea is to use the same time series used for the univariate Section, so the CO(GT) gas and the last week of October. We will use the series without noise, but, some transformations have also been made in order to identify the best series in terms of stationarity. The transformation that we took into considerations with respect to the original dataset (without noise) are:

- Logarithmic Tranformation;
- Differentiated Transformation.

In order to evaluate the stationartiy of the time series we will use the **Dickey-Fuller** test. Thanks to the Deckey-fuller test, we were able to notice that the original series without noise, transformed with the **Differentiated Transformation**, is stationary.

For the differentiated transformation we obtained a test statistic value of -3.5378. In general, the more negative is this statistic, the more likely we will reject the null hypothesis. As part of the output, we get a look-up table that helps in determining the ADF statistic. We can see that our statistic value of -3.5378 is less than the value of -3.4425 at 1%. In this case we can say that with this transformation we obtained a stationary series because this suggests that we can reject the null hypothesis with a significance level of less than 1%. Rejecting the null hypothesis means that the process has no unit root and in turn the time series is stationary or does not have time-dependent structure.

The Autocorrelation and Partial Autocorrelation graphs of the transformed time series are shown in the Figure 25 below. This graphs are also useful for choosing the parameters of some Forecasting models because the Autocorrelation Function (ACF) measures the correlation between a time series and its lagged version. Instead the Partial Autocorrelation Function (PACF) measures the correlation between the time series and its lagged version after eliminating the variations already explained by the intervening comparisons.

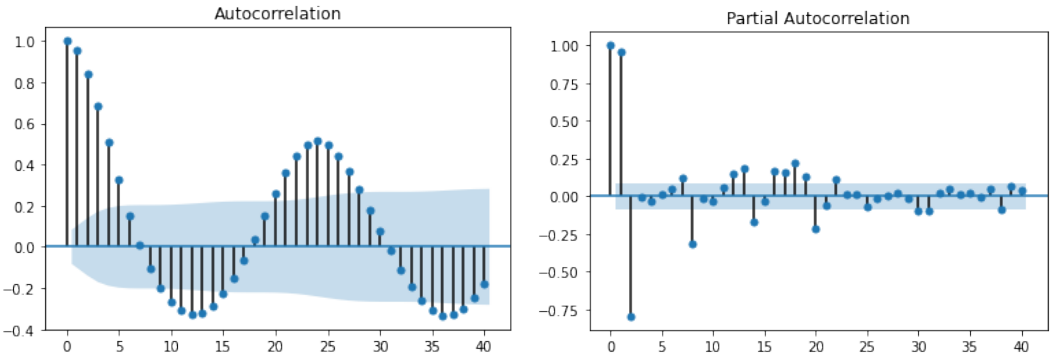

Figure 25: Autocorrelation and Partial Correlation

Then, we carried out the forecasting task on the stationary series and in order to do that we tried five models:

- Simple Smoothing;
- Holt Winter's Exponential Smoothing;
- Exponential Smoothing;
- ARIMA & SARIMAX;

These models were applied, as we said, on the transformed series with the differentiated transformation, because it is the only one that is stationary according to the Dickey-Fuller test results.

In order to evaluate the performances of the models we will use the AIC, MAE and MAPE metrics. For the sake of synthesis, a summary of the best models results is reported in the following table.

	SimpleSmooth	Holt	ExpSmooth
AIC	-1859.78	-2313.43	-1383.19
MAE	0.130	0.458	0.462
MAPE	3.800	1.063	1.066

Table 11: Forecasting Results

As regards ARIMA model, we have no better results, but we want to explain the reasoning behind its implementation. The ARIMA model is a Auto-Regressive Integrated Moving Averages and has this parameters: (p,d,q). To estimate them we based on the graphs of autocorrelation and partial autocorrelation shown before, but, to understand better the threshold we want to re-propose the graphs with a different layout to better identify the parameters, as we can see in the Fig. 26

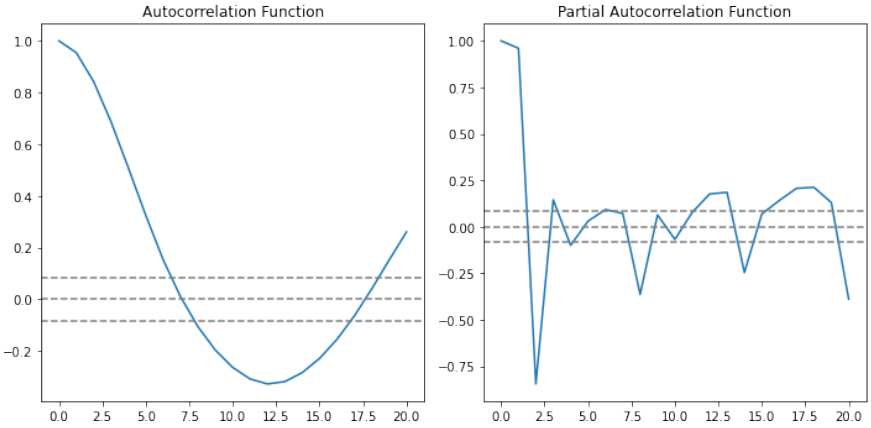

Figure 26: ARIMA correlation

Two dotted lines are the confidence interval that we use to determine the 'p' and 'q' values. We can identify the value of p where the PACF chart crosses the upper confidence interval for the first time, p=2. Instead, the value q where the ACF chart crosses the upper confidence interval for the first time, q=7. So we used the parameters (2,0,7) as parameters of ARIMA model but we did not have better results to show. The same approach has been used for the SARIMAX model but also in this case we had no significant results.

8.5 Classification

In the following section, we will describe the methodologies and the algorithms used during the time series classification and the obtained results. The main goal of this task is to predict if it is Day or Night.

8.5.1 Univariate Classification

In this task we decided to use a dataset formed by the attribute CO(GT). The timeframe was different with respect to the previous tasks: we analyzed the period that goes from the first of September to the end of December. Since we were not sure if predicting Weekend was the best choice, in fact some basic and advanced classifiers did not give satisfying results, we wanted to explore another target configuration. We decided to predict if it was Day, 0, or Night, 1, through the analysis of different techniques: **Shapelet discovery**, **Feature based**, **KNeighborsCalssifiers**, **DecisionTree**, **CNN** and **LSTM**(keras). Among the Shaplet Discovery algorithms, we tried: ShapletModel, LearningShaplet, grabocka_params_to_shapelet_size_dict. The most performing one was the ShapletModel whose results are shown in Table 12.

Using the Feature-based, CNN, LSTM, classifiers we got the same performances in terms of accuracy with a value of 0.67.

For the classic classifiers two combinations were explored: Decision Tree combined with Feature-based Classifier but better performances were reached by KNN in combination with Shaplet-distances-based-Classifiers that was also the best in the whole analysis.

	Parameters	Accuracy
Shaplet-distances-based-Classifiers + KNN	leaf_size=30, metric='minkowski', n_neighbors=5, p=2, weights='uniform'	1.00
ShapletModel	optimizer='sgd', weight_regularizer=.01, max_iter=200, verbose=1)	0.67
Feature-based Classifier + DecisionTree	criterion: gini, max_depth=8	0.67
CNN	monitor='loss', factor=0.5, patience=50, min_lr=0.0001	0.67
LSTM	loss='sparse_categorical_crossentropy', optimizer='adam'	0.67

Table 12: Classification Results

8.5.2 Multivariate Classification

As regards the multivariate classification analysis, we focused on CO(GT), NOx(GT), PT08.S1(CO) keeping the same timeframe and target variable. Therefore, we will show the multivariate analysis done with LSTM model that gave better performances compared to LSTM univariate analysis. The following parameters were set: kernel_inizializer = 'TruncatedNormal', epoch=20, dropout=0.3.
At the end we observed an accuracy value of 0.78.

9 Sequential Pattern Mining

In this section we will look at the dataset for the purpose of finding any frequent patterns. To carry out this analysis we extended the timeframe to the whole dataset in order to have more satisfying results. As in the previous task, only CO(GT) attribute was considered and its original time series was transformed using the Compression Symbolic Aggregate Approximation (SAX) with the following setting:

- Segment Length: 30 records;
- n_symbols: 15.

that is also displayed in the Fig. 27 below:

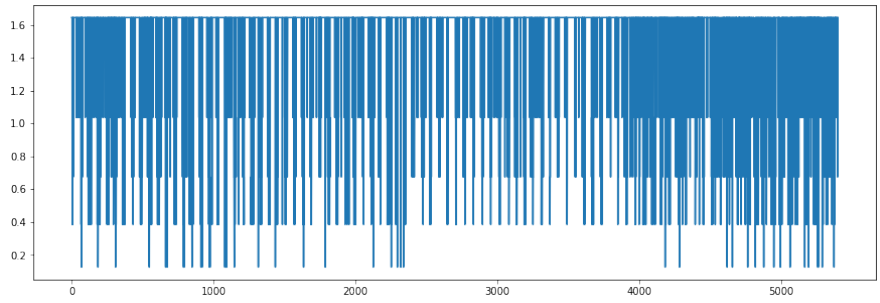

Figure 27: sax transformation CO(GT)

At this point, to better perform the analysis, it was necessary to create a transactional dataset which allowed us to create some sequences of dimension of 14.
In order to find any frequent patterns, we explored its preconfigured sequences with the implementation of the PrefixSpan library. All patterns have been extracted considering a frequency equal to 2 and a minimum length of 3. With these parameters we found 96 frequent patterns, but in the following table we will display only the top10.

Frequency	Frequent Pattern
3	(3, 4, 4, 4), (4, 4, 4, 4), (4, 4, 4, 4), (4, 4, 4, 4)
3	(4, 4, 4, 4), (4, 4, 4, 4), (4, 4, 4, 4), (4, 4, 4, 4)
2	(4, 4, 4, 4), (3, 4, 4, 4), (4, 4, 4, 4), (4, 4, 4, 4), (4, 4, 3, 4)
2	(4, 4, 4, 4), (4, 4, 3, 4), (4, 4, 4, 4), (4, 4, 4, 4)
2	(4, 4, 4, 4), (4, 4, 3, 4), (4, 4, 4, 4), (4, 4, 4, 4), (4, 4, 4, 4)
2	(4, 4, 4, 4), (4, 4, 3, 4), (4, 4, 4, 4), (4, 4, 4, 4), (4, 4, 4, 4), (4, 4, 3, 4)
2	(4, 4, 4, 4),(4, 4, 3, 4),(4, 4, 4, 4),(4, 4, 4, 4),(4, 4, 4, 4),(4, 4, 3,4), (4, 4, 3, 4)
2	(4, 4, 4, 4), (4, 4, 4, 4), (4, 3, 4, 4), (4, 4, 4, 4), (4, 4, 4, 4), (4, 4, 4, 4), (4, 4, 4, 4), (4, 4, 4, 4)
2	(4, 4, 4, 4), (4, 4, 4, 4), (4, 4, 4, 4), (4, 4, 4, 4), (4, 4, 3, 4)
2	(4, 4, 4, 4), (4, 4, 4, 4), (4, 4, 4, 4), (4, 4, 4, 4), (4, 4, 3, 4), (4, 4, 3, 4)

Table 13: Frequent Patterns

After that, we extracted from the time series only 'closed patterns', which are patterns in which none of the subsets has the same support value compared to that of their nearby subsets.

Since we didn't found any significant closed pattern with respect to the top10 ones, we won't give their representation, but, taking into consideration this kind of patterns, we identified 20 of them. With the objective of exploring deeper the frequent patterns, we tested also different frequencies and minimum length, as we can see in Table 14:

Parameters	n-Pattern	n-ClosedPattern
Freq=2, min_length=4	15	7
Freq=3, min_length=2	8	4
Freq=3, min_length=3	5	3

Table 14: SPM

10 Outliers Detection

Since our dataset did not have a target variable, we necessarily had to slightly reprocess it. Besides, it appeared to be slightly different from its description, too. We necessarily have to drop some columns and rows that exceed our dataset dimensions. For this task we decided to analyze the true hourly averaged concentration of the gas except for NMHC(GT) which contains too many missing values. Those gas were: CO(GT), C6H6(GT), NOx(GT) and NO2(GT).

In order to find outliers we used various approaches: the basic Boxplot as a visual approach, two density-based such as DBSCAN and LOF, a distance-based like k-NN, and a NN, Autoencoder. We also tried an angle-based model, ABOD, which was not suitable for our analysis, perhaps because it works better with "high-dimensional" datasets.
The first method to be applied was the visual approach: it appeared to be useful at a very first sight. Boxplots were generated, as shown in Figure 28, individuating a total of 1029 outliers in the whiskers of the plots. It gives an idea of the general data distribution but, since it is not automatic, the outliers boundary identification is subjected to one's decision.

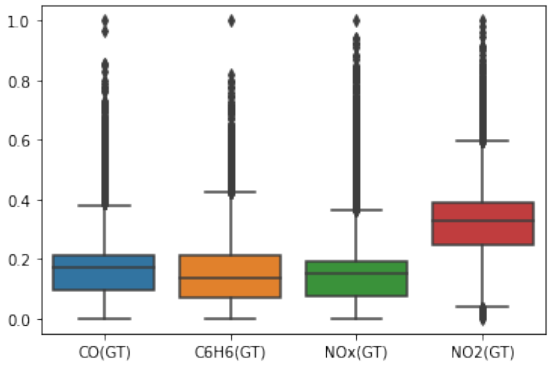

Figure 28: Boxplot

Secondly, we also tried to implement a distance-based approach, that is k-NN. Since k-NN uses underlying trends in the data to make predictions, prediction errors are telltale signs of data points that do not fit to the overall trend. Basically, by assigning to each point its distance from its k-NN as outlier score, it classified 931 points as outliers.

K	100
Outliers	931
Aver. Outlier score	70.93
Aver. Normal data score	12.69

Table 15: k-NN Parameters and Results

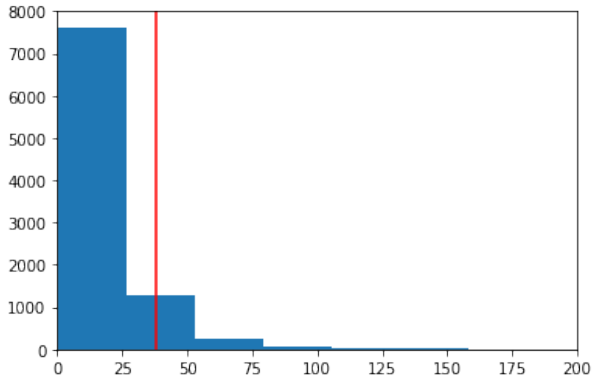

Figure 29: k-NN Outliers

We also implemented a kind of density-based approach to anomalies detection. General idea of these methods is: low density areas of data distribution may identify anomalies in that distribution.

Namely, normal distributed points have a similar distribution density compared to the neighbourhood; conversely, points considered to be outliers have a surrounding density different from that of their neighbours.

As shown in Figure 30, the optimal value for eps is where the curve has its maximum curvature, that is eps = 0.04 and min points=4 was chose as best value. DBSCAN labelled 815 points as outliers.

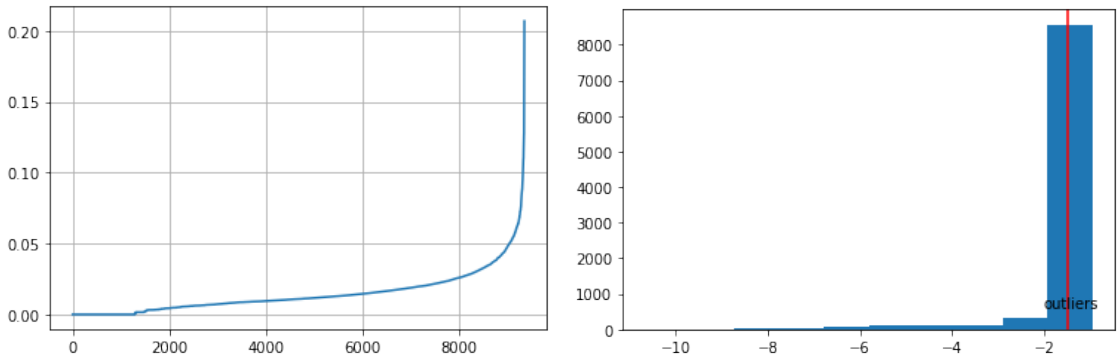

Figure 30: Knee Method for DBScan and LOF Outliers Distribution

Nevertheless, density-based approaches may have problems with different density areas of distribution. In such cases key factor density appears to be less informative.

LOF (Local Outlier Factor) may be the answer, though. Indeed, the local neighborhood density is computed for each sample p: that is the average of the ratios of the density of the sample p and the density if its neighbors. High LOF value means that point is an outlier. With this approach, 1112 outliers were detected.

In the end, we decided to try also with a NN model, Autoencoder. This last method represents each record through the compression and decompression of a Deep Neural Network. Through the reconstruction error of the DNN, it tagged as outliers the points that generate a large value for that error. With 4 hidden Layers and after 40 Epochs, it identified 935 records as outliers.

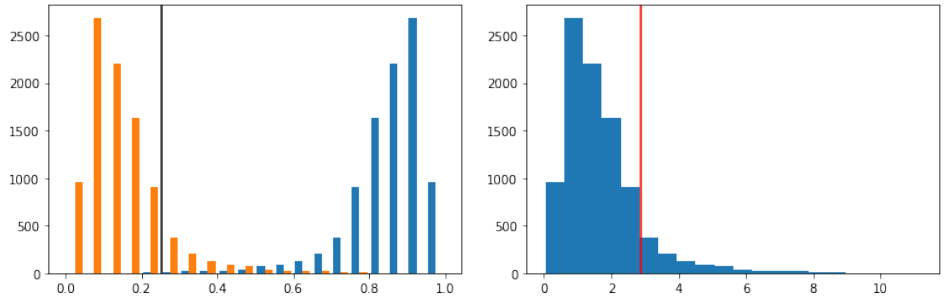

Figure 31: Autoencoder Outlier Distribution
Probability and Anomaly score