Air Quality and weather open data: application of ML methods

Elena Fusillo

Dipartimento di Fisica e Astronomia
Alma Mater Studiorum - Università di Bologna

July 31, 2023

Abstract

This paper aims to define a ML model capable of making valid predictions of the air pollutant *Nitrogen Dioxide (NO_2)* after 24 hours for three stations of the *ARPAE air quality monitoring network* [3] of the province of Rimini, exploiting the historical data of the pollutant [1] and the weather data [2]. The data has been collected and manipulated to have an optimal structure to be batched and fed to *machine learning* algorithms. Different *neural network* structures are compared to choose the one that provides the most accurate predictions. Finally, results of the various algorithms and some considerations are shown.

1 Introduction

Air pollution, i.e. the alteration of the composition of the air caused by the diffusion of particular harmful substances, is a very serious and widespread problem both in terms of human health and the environment. Obviously, the effects on health worsen as the concentration of these substances in the air increases. For this reason, the air pollution values are constantly monitored by stations scattered throughout the national and international territory. Given that air quality can seriously affect the health of the citizen, in addition to the current values provided by the various stations, it would be very useful to have forecasts on these polluting agents; the main objective will therefore consist in the generation of forecasts using some of the main *machine learning* techniques.

2 Air quality monitoring network

To monitor and control air quality, the European Union has issued directives [4] which have then been implemented by the various EU countries. In Italy, the management of the air quality network is entrusted to the various regional bodies ARPA, which stands for *"Regional Agency for Environmental Protection"*: in particular, in Emilia-Romagna ARPAE is the official body that deals with the publication of data relating to the weather and air quality through open dataset with standardized format, freely usable, reusable and redistributable, according to the indications of the user license.

The current ARPAE air quality monitoring network is made up of **47 monitoring stations** distributed throughout the region according to the criteria of representativeness of the territory and the cost-effectiveness of the monitoring system.

The instrumentation is managed according to the provisions of the D.M. 30/03/2017 [5]. The acquired data is subjected to daily, monthly and half-yearly validation procedures in accordance with the provisions of the ARPAE Quality Management System; moreover the measurement network is certified according to the UNI EN ISO 9001:2015 standard.

Each monitoring station is classified according to:

- the zoning criteria: *west plain, east plain, apennines, agglomeration*;
- the type of station: *urban, suburban, rural*;
- the type of prevailing emissions: *background, traffic, industrial.*

2.1 Air pollutants: nitrogen dioxide NO_2 and its legal limits

The air pollutants that are measurable with the ARPAE monitoring stations are 11, namely: Nitrogen Dioxide, Nitrous oxide NO, Nitrogen Oxides NOX, Carbon Monoxide CO, Sulphur Dioxide SO_2, Ozone O_3, Xylene, Toluene, Benzene and particulate matter $PM_{2,5}$ and PM_{10}. The location of the monitoring stations in Emilia-Romagna is shown in Figure 1.

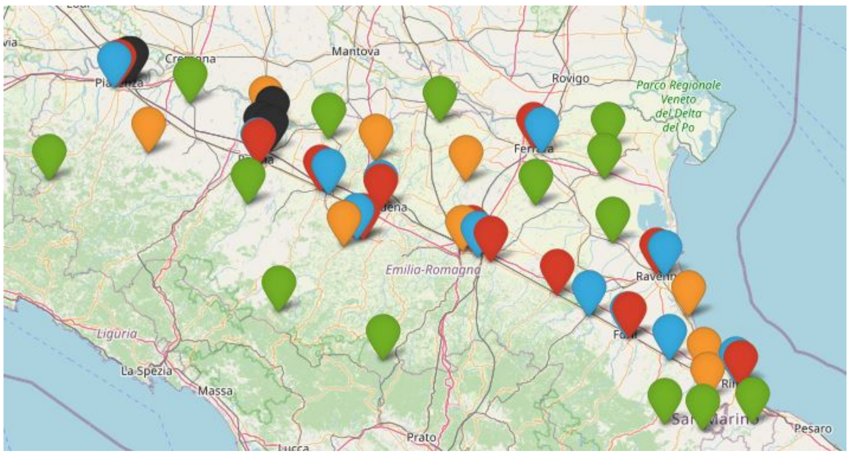


Figure 1: Distribution of air quality monitoring stations in Emilia-Romagna. Marker legend: green - rural background, orange - suburban background, blue - urban background, red - urban traffic, black - local.

Obviously none of them is capable of monitoring all these compounds, but only a subset of them. Almost all of the stations are capable of measuring PM_{10} and NO_2, while many, but not all, are capable of measuring Benzene, CO, $PM_{2,5}$ and O_3.

In this paper we will focus on one of the main pollutants produced by them: **nitrogen dioxide (NO_2)**, which is a compound mainly produced by combustion processes, which take place, for example, in motor vehicles or domestic heating systems. It is an irritant gas and can therefore cause serious respiratory problems; in severe cases it can even lead to death.

As for any other pollutant, limits have been set for the concentration of *nitrogen dioxide* (shown in Tab. 1) [8].

NO_2 limits

Hourly limit	Annual limit
200 $\mu g/m^3$	Maximum 18 overrun of hourly limit + 40 $\mu g/m^3$ as annual limit value

Table 1: Hourly and annual limits for measured NO_2 values.

2.2 Air quality data

Speaking of air quality data, ARPAE Emilia-Romagna offers both real-time data and historical archives, which are accessible through two main channels: the ARPAE website [7] and the OpenData portal [1].

In the specific case of this work, data from across the region were retrieved from the OpenData portal which allows you to obtain it directly in CSV format via a shared Google Drive folder, to then focus exclusively on data from the province of Rimini.

The strength of the ARPAE data is the completeness of the historical archives: in fact, hourly *validated* measurements of up to 11 pollutants are provided for each station of the regional territory from 2010 until 2022, thus giving the possibility to carry out analysis on the data. In total there are 47 fixed stations distributed throughout the region, plus 5 local stations which monitor situations of particular interest (industrial activities).

2.2.1 Weather data

In order to improve the quality of air quality forecasts, measurements relating to weather conditions may in fact be useful. The various regional ARPA offer the most complete meteorological data and with a good historical archive. It was decided to focus exclusively on the province of Rimini and therefore on the data offered by ARPAE Emilia-Romagna. Both real-time measurements and historical archives are available and accessible through the Dext3r webpage [2], from which it is possible to obtain the data directly in CSV, XLS or PDF format.

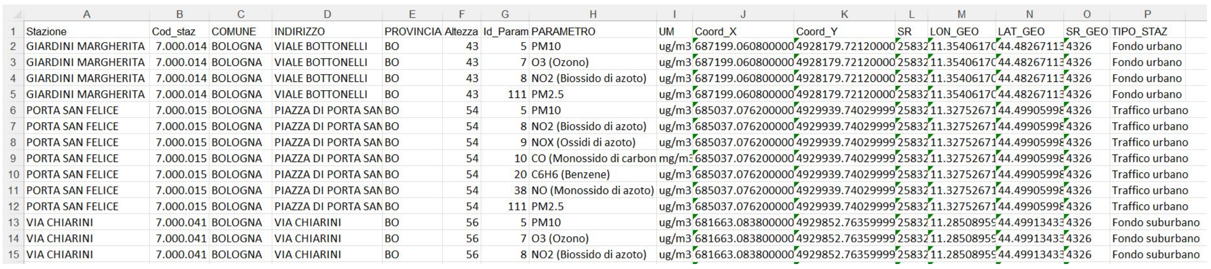
3 Data retrieval and organization

As already specified, data are available through two channels: the OpenData portal [1] for the air quality data and the Dext3r webpage [2] for the weather data.

3.1 Import of ARPAE data on air quality (OpenData)

On the OpenData portal of ARPAE it is possible to find files in CSV format containing historical validated data relating to air quality from 2010 to 2022, shared in a Google Drive folder. Two more files are shipped with the data, which contain the identifiers of the various stations and of the various parameters necessary for correctly interpreting the records, that are:

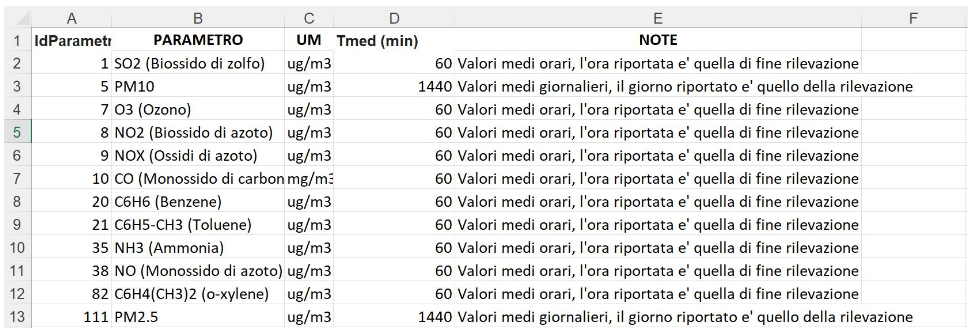
- "QARIA_Stazioni.xlxs": it contains the information about the *master data* of the monitoring stations (Fig. 2);



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1	Stazione	Cod_staz	COMUNE	INDIRIZZO	PROVINCIA	Altezza	Id_Param	PARAMETRO	UM	Coord_X	Coord_Y	SR	LON_GEO	LAT_GEO	SR_GEO	TIPO_STAZ
2	GIARDINI MARGHERITA	7.000.014	BOLOGNA	VIALE BOTTONELLI	BO	43	5	PM10	ug/m3	687199.060800000	4928179.72120000	2583	11.3540617	44.4826711	4326	Fondo urbano
3	GIARDINI MARGHERITA	7.000.014	BOLOGNA	VIALE BOTTONELLI	BO	43	7	O3 (Ozono)	ug/m3	687199.060800000	4928179.72120000	2583	11.3540617	44.4826711	4326	Fondo urbano
4	GIARDINI MARGHERITA	7.000.014	BOLOGNA	VIALE BOTTONELLI	BO	43	8	NO2 (Biossido di azoto)	ug/m3	687199.060800000	4928179.72120000	2583	11.3540617	44.4826711	4326	Fondo urbano
5	GIARDINI MARGHERITA	7.000.014	BOLOGNA	VIALE BOTTONELLI	BO	43	111	PM2.5	ug/m3	687199.060800000	4928179.72120000	2583	11.3540617	44.4826711	4326	Fondo urbano
6	PORTA SAN FELICE	7.000.015	BOLOGNA	PIAZZA DI PORTA SAN	BO	54	5	PM10	ug/m3	685037.076200000	4929939.74029999	2583	11.3275267	44.4990599	4326	Traffico urbano
7	PORTA SAN FELICE	7.000.015	BOLOGNA	PIAZZA DI PORTA SAN	BO	54	8	NO2 (Biossido di azoto)	ug/m3	685037.076200000	4929939.74029999	2583	11.3275267	44.4990599	4326	Traffico urbano
8	PORTA SAN FELICE	7.000.015	BOLOGNA	PIAZZA DI PORTA SAN	BO	54	9	NOX (Ossidi di azoto)	ug/m3	685037.076200000	4929939.74029999	2583	11.3275267	44.4990599	4326	Traffico urbano
9	PORTA SAN FELICE	7.000.015	BOLOGNA	PIAZZA DI PORTA SAN	BO	54	10	CO (Monossido di carbon	mg/m	685037.076200000	4929939.74029999	2583	11.3275267	44.4990599	4326	Traffico urbano
10	PORTA SAN FELICE	7.000.015	BOLOGNA	PIAZZA DI PORTA SAN	BO	54	20	C6H6 (Benzene)	ug/m3	685037.076200000	4929939.74029999	2583	11.3275267	44.4990599	4326	Traffico urbano
11	PORTA SAN FELICE	7.000.015	BOLOGNA	PIAZZA DI PORTA SAN	BO	54	38	NO (Monossido di azoto)	ug/m3	685037.076200000	4929939.74029999	2583	11.3275267	44.4990599	4326	Traffico urbano
12	PORTA SAN FELICE	7.000.015	BOLOGNA	PIAZZA DI PORTA SAN	BO	54	111	PM2.5	ug/m3	685037.076200000	4929939.74029999	2583	11.3275267	44.4990599	4326	Traffico urbano
13	VIA CHIARINI	7.000.041	BOLOGNA	VIA CHIARINI	BO	56	5	PM10	ug/m3	681663.083800000	4929852.76359999	2583	11.2850895	44.4991343	4326	Fondo suburbano
14	VIA CHIARINI	7.000.041	BOLOGNA	VIA CHIARINI	BO	56	7	O3 (Ozono)	ug/m3	681663.083800000	4929852.76359999	2583	11.2850895	44.4991343	4326	Fondo suburbano
15	VIA CHIARINI	7.000.041	BOLOGNA	VIA CHIARINI	BO	56	8	NO2 (Biossido di azoto)	ug/m3	681663.083800000	4929852.76359999	2583	11.2850895	44.4991343	4326	Fondo suburbano

Figure 2: Master data of the monitoring stations.

- "parametri.xlsx": it consists in a *look-up table* (Fig. 3) which links the pollutant name to the "ID parameter" (the name of the column in the data files).



	A	B	C	D	E	F
1	IdParametr	PARAMETRO	UM	Tmed (min)	NOTE	
2	1	SO2 (Biossido di zolfo)	ug/m3	60	Valori medi orari, l'ora riportata e' quella di fine rilevazione	
3	5	PM10	ug/m3	1440	Valori medi giornalieri, il giorno riportato e' quello della rilevazione	
4	7	O3 (Ozono)	ug/m3	60	Valori medi orari, l'ora riportata e' quella di fine rilevazione	
5	8	NO2 (Biossido di azoto)	ug/m3	60	Valori medi orari, l'ora riportata e' quella di fine rilevazione	
6	9	NOX (Ossidi di azoto)	ug/m3	60	Valori medi orari, l'ora riportata e' quella di fine rilevazione	
7	10	CO (Monossido di carbon	mg/m3	60	Valori medi orari, l'ora riportata e' quella di fine rilevazione	
8	20	C6H6 (Benzene)	ug/m3	60	Valori medi orari, l'ora riportata e' quella di fine rilevazione	
9	21	C6H5-CH3 (Toluene)	ug/m3	60	Valori medi orari, l'ora riportata e' quella di fine rilevazione	
10	35	NH3 (Ammonia)	ug/m3	60	Valori medi orari, l'ora riportata e' quella di fine rilevazione	
11	38	NO (Monossido di azoto)	ug/m3	60	Valori medi orari, l'ora riportata e' quella di fine rilevazione	
12	82	C6H4(CH3)2 (o-xylene)	ug/m3	60	Valori medi orari, l'ora riportata e' quella di fine rilevazione	
13	111	PM2.5	ug/m3	1440	Valori medi giornalieri, il giorno riportato e' quello della rilevazione	

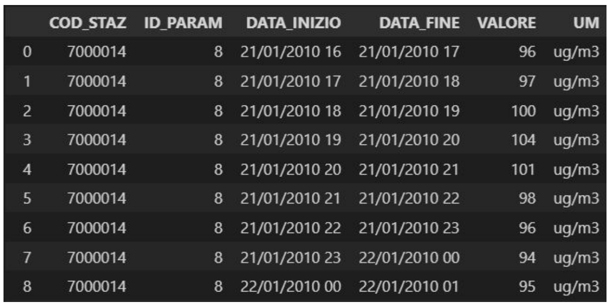
Figure 3: Look-up table of the air pollutants.

A total of 11 pollutants are measured, where the recorded values are the average values with respect to the sampling frequency, the time reported is that of the end of the detection. Between the downloaded data, there are 2938 .csv *validated* files, one for each combination of year, station and air pollutant measured. These filename convention is *"storico_YYYY_CCCCCCCC_PPP.csv"*, where:

- *YYYY*: year (4 digits);
- *CCCCCCCC*: station code (8 digits);
- *PPP*: pollutant number (3 digits).

The content of a single CSV file looks like Fig. 4, where:

- *COD_STAZ*: integer (from 7 to 8 digits);
- *ID_PARAM*: integer (from 1 to 3 digits);
- *DATA_INIZIO*: DD/MM/YYYY hh (daily or hourly frequency);
- *DATA_FINE*: DD/MM/YYYY hh (daily or hourly frequency);
- *VALORE*: integer or real, measured mean value;
- *UM*: string (it stands for "unit of measure", 5 characters).



	COD_STAZ	ID_PARAM	DATA_INIZIO	DATA_FINE	VALORE	UM
0	7000014	8	21/01/2010 16	21/01/2010 17	96	ug/m3
1	7000014	8	21/01/2010 17	21/01/2010 18	97	ug/m3
2	7000014	8	21/01/2010 18	21/01/2010 19	100	ug/m3
3	7000014	8	21/01/2010 19	21/01/2010 20	104	ug/m3
4	7000014	8	21/01/2010 20	21/01/2010 21	101	ug/m3
5	7000014	8	21/01/2010 21	21/01/2010 22	98	ug/m3
6	7000014	8	21/01/2010 22	21/01/2010 23	96	ug/m3
7	7000014	8	21/01/2010 23	22/01/2010 00	94	ug/m3
8	7000014	8	22/01/2010 00	22/01/2010 01	95	ug/m3

Figure 4: Air quality data, raw.

Here, there is a record for each measurement; each of them has a date, a value, a type of pollutant (for example 8=Nitrogen dioxide), as well as the reference to the station that carried out the measurement. However, the data structured in this way are not suitable for being supplied to a neural network: it is necessary to identify a way to be able to group them, so that each record contains the concentration of the pollutant and the weather information, for each *nitrogen dioxide* monitoring station under consideration.

Some of these monitoring stations are dropped out from subsequent data analysis due to their nature, that are: 5 local stations which monitor situations of particular interest (stations number 2000229, 2000230, 2000232, 5000020, 5000024), 2 dummy stations (stations number 7000046, 7000047) and 1 stations with hourly missing data (station number 10000002).

From the evidence of the original file formatting, it was decided to group all the air quality data into two datasets (one for each sample rate) and then extract only the selected monitoring stations so as to possibly allow to expand the study to other stations, cities, provinces or other types of pollutants for those interested.

First of all, all the years and all the pollutants distinguished by sampling frequency for each single station were combined by means of a combination of *Pandas* “left” merge and concatenation on the horizontal axis.

Missing data: KNN imputation

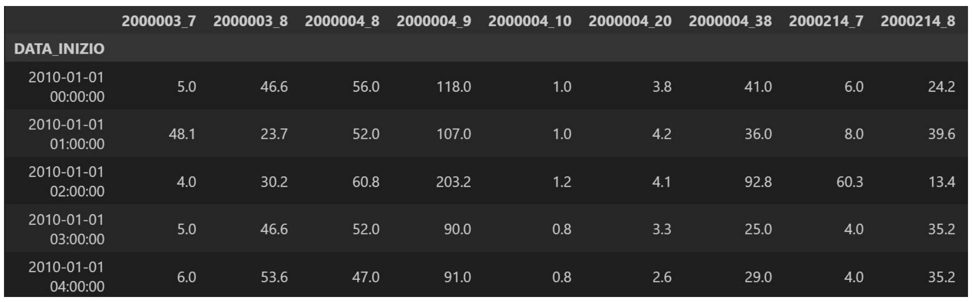
Analyzing the obtained data, it is possible to notice that, for some time intervals of variable length, the measurements both of air pollutants and weather conditions are null or absent. Possible values are:

- *zero*: it is a measured value indeed;
- *NaN*: either that pollutant is not measured in the considered monitoring station, or the data was invalidated following some *data capture criteria* [10].

Since it is not possible to provide a value not present (*NaN*) as input to a neural network, it was necessary to complete the missing data. It was chosen to proceed with the use of the *KNN imputer algorithm*, made available by the *scikit-learn* Python library [9].

It is an algorithm that exploit the *k-Nearest Neighbors* approach, imputing each missing feature using uniformly averaged values from $k = 5$ nearest neighbors that have a value for the feature, where the distance is measured via an euclidean distance metric that supports missing values. It is important to notice that when the number of available neighbors is less than k and there are no defined distances to the training set, the training set average for that feature is used during imputation.

Finally, through a *Pandas* concatenation on the vertical axis all the stations with hourly and daily data were joined in two separate files depending on the sampling frequency. An example of the hourly dataframe is shown in Figure 5 (some attributes have been omitted for simplicity).



	2000003_7	2000003_8	2000004_8	2000004_9	2000004_10	2000004_20	2000004_38	2000214_7	2000214_8
DATA_INIZIO									
2010-01-01 00:00:00	5.0	46.6	56.0	118.0	1.0	3.8	41.0	6.0	24.2
2010-01-01 01:00:00	48.1	23.7	52.0	107.0	1.0	4.2	36.0	8.0	39.6
2010-01-01 02:00:00	4.0	30.2	60.8	203.2	1.2	4.1	92.8	60.3	13.4
2010-01-01 03:00:00	5.0	46.6	52.0	90.0	0.8	3.3	25.0	4.0	35.2
2010-01-01 04:00:00	6.0	53.6	47.0	91.0	0.8	2.6	29.0	4.0	35.2

Figure 5: Pandas dataframe structure of the hourly air quality data.

In fact, in Figure 4 there was only one *nitrogen dioxide* monitoring stations: 7000014. After data manipulation, a new series of records will therefore be generated, in which a particular date will appear only once, which will contain the various measurements for each station identified. An example of such a conversion is shown in Figure 5.

This type of data organization is propaedeutic to *Pandas multiindexing*, which enables you to store and manipulate data with an arbitrary number of dimensions in lower dimensional data structures.

3.2 Import of ARPAE data on weather (Dext3r)

As previously discussed, in order to be able to make more accurate forecasts with the neural networks that will be defined later, it also seemed necessary to collect weather data since it is assumed that the pollution value does not vary on a regular basis, but depends on many other factors; some of them could be, in fact, date and time, or weather conditions.

The most significant ones, on which air pollution probably depends, will therefore be adopted as input features with hourly sampling rate; they are:

- *Temperature (1001):* Temperature alters the properties of the atmosphere, affecting the rate of dispersion of air pollutants;
- *Humidity (1002):* As with temperature, humidity alters the properties of the atmosphere, affecting the speed of dispersion of atmospheric pollutants;
- *Precipitation (1003):* Precipitation manages to trap atmospheric pollutants, bringing them to the ground and thus reducing their concentration in the air;
- *Wind speed (1004) and direction (1005):* The wind moves or disperses air pollutants by moving air masses.

The measurements relating to these quantities are available at various temporal frequencies and with different types of aggregation (maximum, minimum, average, instantaneous value).

Taking the temperature as an example, the location of the weather stations in Emilia-Romagna capable of carrying out this measurement is shown in Fig. 6. The locations of the weather stations capable of measuring the desired quantities does not correspond exactly to those of the air quality monitoring network.

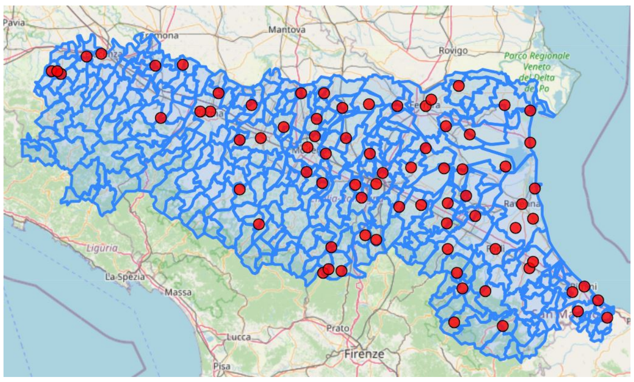
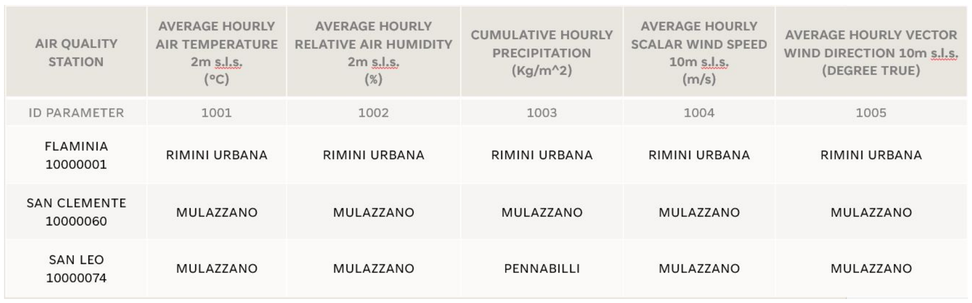


Figure 6: Distribution of weather monitoring stations which monitors temperature in Emilia-Romagna.

The selected input features will be related to *nitrogen dioxide* monitoring stations; yet since the meteorological stations are located in different places, the most appropriate meteorological measurements were decided to be the ones spatially closer and which had sufficient data for the chosen feature. This should allow to obtain higher quality data (obviously if the missing data are not too many), probably allowing the neural network to achieve better results.

In Fig. 7 are shown the choosen *a priori* correspondences between air quality stations (first column) and weather stations (table content) of the province of Rimini for each feature considered.



AIR QUALITY STATION	AVERAGE HOURLY AIR TEMPERATURE 2m s.l.s. (°C)	AVERAGE HOURLY RELATIVE AIR HUMIDITY 2m s.l.s. (%)	CUMULATIVE HOURLY PRECIPITATION (Kg/m^2)	AVERAGE HOURLY SCALAR WIND SPEED 10m s.l.s. (m/s)	AVERAGE HOURLY VECTOR WIND DIRECTION 10m s.l.s. (DEGREE TRUE)
ID PARAMETER	1001	1002	1003	1004	1005
FLAMINIA 10000001	RIMINI URBANA	RIMINI URBANA	RIMINI URBANA	RIMINI URBANA	RIMINI URBANA
SAN CLEMENTE 10000060	MULAZZANO	MULAZZANO	MULAZZANO	MULAZZANO	MULAZZANO
SAN LEO 10000074	MULAZZANO	MULAZZANO	PENNABILLI	MULAZZANO	MULAZZANO

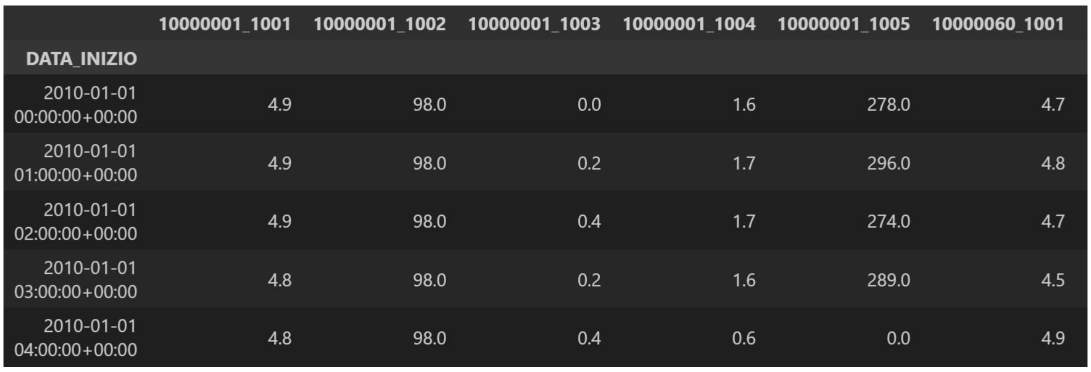
Figure 7: Match between Rimini's air quality stations and weather monitoring ones.

On the Dext3r webpage of ARPAE Emilia-Romagna it is possible to find files in CSV format containing historical data relating to weather conditions. However, data requests must be fragmented as the website manages files up to a maximum of 25,000 rows at a time.

It was therefore necessary to send many separate requests, then the obtained data were merged in such a way as to form a *.csv* file with an hourly sampling rate for each station and for each meteorological feature.

Also here, through a *Pandas* concatenation on the vertical axis all the stations with hourly data were joined; subsequently the missing values have been filled through the use of the *KNN algorithm*, similarly to the air pollutants data.

The weather data records stored in the *Pandas dataframes* will thus have a structure similar to that shown in Figure 8 (some attributes have been omitted for simplicity). As can be seen, meteorological measurements from other stations have been associated with the *nitrogen dioxide* monitoring stations of the province of Rimini.

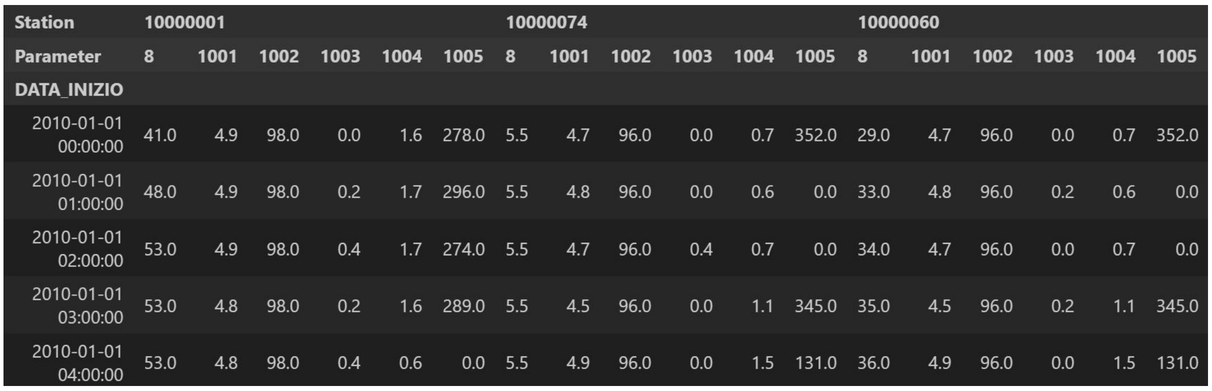


	10000001_1001	10000001_1002	10000001_1003	10000001_1004	10000001_1005	10000060_1001
DATA_INIZIO						
2010-01-01 00:00:00+00:00	4.9	98.0	0.0	1.6	278.0	4.7
2010-01-01 01:00:00+00:00	4.9	98.0	0.2	1.7	296.0	4.8
2010-01-01 02:00:00+00:00	4.9	98.0	0.4	1.7	274.0	4.7
2010-01-01 03:00:00+00:00	4.8	98.0	0.2	1.6	289.0	4.5
2010-01-01 04:00:00+00:00	4.8	98.0	0.4	0.6	0.0	4.9

Figure 8: Pandas dataframe structure of the hourly weather data.

3.3 Data concatenation and multi indexing

Once obtained the *Pandas dataframes* shown in Figure 5 and 8, concatenation can be done. Since it was decided to focus only on the pollutant *nitrogen dioxide* NO_2 present in the stations of the province of Rimini, the air quality data columns are first filtered according to these criteria. Subsequently, a *Pandas* concatenation on the vertical axis is carried out between the two dataframes in order to create a unique one (Figure 9). This, in turn, has been organized using *Pandas multi indexing*, which allows for more versatility in data manipulation. In doing so, each record corresponds to a specific day and time and contains for each selected station all the values of the selected input features: these data can be easily fed to a neural network.



Station	10000001						10000074						10000060					
Parameter	8	1001	1002	1003	1004	1005	8	1001	1002	1003	1004	1005	8	1001	1002	1003	1004	1005
DATA_INIZIO																		
2010-01-01 00:00:00	41.0	4.9	98.0	0.0	1.6	278.0	5.5	4.7	96.0	0.0	0.7	352.0	29.0	4.7	96.0	0.0	0.7	352.0
2010-01-01 01:00:00	48.0	4.9	98.0	0.2	1.7	296.0	5.5	4.8	96.0	0.0	0.6	0.0	33.0	4.8	96.0	0.2	0.6	0.0
2010-01-01 02:00:00	53.0	4.9	98.0	0.4	1.7	274.0	5.5	4.7	96.0	0.4	0.7	0.0	34.0	4.7	96.0	0.0	0.7	0.0
2010-01-01 03:00:00	53.0	4.8	98.0	0.2	1.6	289.0	5.5	4.5	96.0	0.0	1.1	345.0	35.0	4.5	96.0	0.2	1.1	345.0
2010-01-01 04:00:00	53.0	4.8	98.0	0.4	0.6	0.0	5.5	4.9	96.0	0.0	1.5	131.0	36.0	4.9	96.0	0.0	1.5	131.0

Figure 9: Full final dataframe to use with neural networks.

4 Machine learning techniques

Following the data collection, storage and manipulation, we will proceed with the search for the main machine learning techniques; these will be subsequently adopted by exploiting the measurements collected in the previous phase.

4.1 Introduction to machine learning

The main objective of *machine learning* is to be able to train a machine so that it learns to perform certain operations, to obtain the desired results, without explicitly providing it with the algorithm to perform. The machine must be able to learn the links present in a set of data, developing a specific behavior; this learned behavior can then be exploited to make predictions on new data. Learning can take place in three possible ways:

- **Supervised learning**: training takes place by providing the input data and, for each one, the corresponding expected result. The goal is to identify a function that transforms the input data into the expected results, minimizing the error. At each training iteration, the prediction made by the model is compared with the expected result, then calculating the new parameters to reduce the gap between the two values;
- **Unsupervised learning**: this type of training takes place by providing only the input data, without the expected results. It will be up to the model to try to extract some kind of knowledge from the data that has been fed to it;
- **Reinforcement learning**: in this type of training, the model is placed in a particular environment. The objective of the model will be to maximize a sort of score, through the rewards provided by the environment, as an effect of particular interactions with it.

One of the main techniques for carrying out machine learning is the use of so-called *artificial neural networks*. These networks are inspired by the structure of the human brain: in fact, one of the main components of these networks are artificial neurons.

In short, an *artificial neuron* receives a set of inputs, adds them together by multiplying them by specific weights and finally applies a particular *activation function*, returning the calculated value as output. By connecting a certain number of these neurons together, it is possible to obtain a very complex network, similar to the human brain. In general, the behavior of such a network is defined by the input weights of each single neuron, which will have to be determined through the training process. Neurons are grouped into layers, receiving inputs from the previous layer and providing outputs to the next layer. The initial layer, connected to the input data, is called *input layer*, while the last one, which supplies the network output, is called *output layer*; among these there may be various *hidden layers*, which allow to increase the complexity and potential of the network. Since hidden layers can be present in very large numbers, the term *Deep Learning* was born, which alludes precisely to the very high depth of layers that these networks can have.

4.2 Terminology: features and labels

Here are some terms that are widely used in the *machine learning* field:

- *features*: are the values that each individual input data record contains. Each record, or example, will be provided to the network one at a time;
- *labels*: are the values that the model will have to try to predict. During the *training phase*, the labels corresponding to the input data (*labeled example*) will also be provided to the model; in this way it will be able to calculate the error between the desired label and the forecast, correcting itself accordingly. During the model evaluation phase (*inference*), only the input data (*unlabeled example*) is provided to the model and it will have to produce the corresponding labels by exploiting the knowledge accumulated during the training phase.

4.3 Linear regression

The *labels* that the model will have to predict could either assume only very specific values or any real value. In the first case it is necessary to solve a *classification* problem, while in the second a *regression* problem. In this paper we will focus on the latter.

Suppose we have a set of records with a single feature, and we need to determine the corresponding label. If, between the *feature* and the *label*, there is an almost linear relationship, then it will be sufficient to define a straight line. The function associated with this line will be able to predict, approximately, the labels to be associated with any value of the input feature.

The equation of the straight line will obviously be of the type:

$$y = b + wx \tag{1}$$

Where:

- **y** is the value you want to predict (*label*);
- **b** is the point of intersection with the ordinate axis, it corresponds to the *bias*;
- **w** is the angular coefficient of the straight line, it corresponds to the weight associated with feature x;
- **x** is the value of the *input feature.*

In order to identify this straight line it will therefore be necessary to determine the values corresponding to *b* and *w*. This is exactly the goal of the *training* phase. A very simple *neural network*, without activation function, capable of representing the line just described is shown in Figure 10.

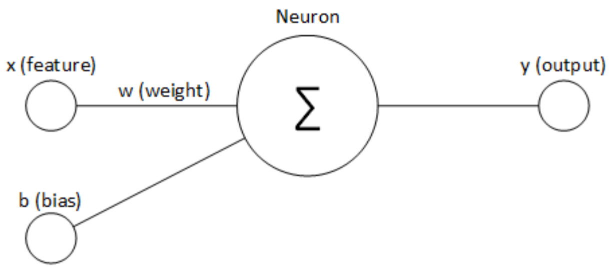


Figure 10: Simple neural network with a single neuron, one single input feature and no activation funtion applied to the output.

If there were more features, the equation (1) would become:

$$y = b + w_1x_1 + w_2x_2 + ... + w_nx_n \tag{2}$$

4.4 Calculation and error minimization

The goal of the model training phase is to identify the most appropriate *weights*, i.e. those that allow to obtain the best predictions. The *training* phase therefore takes place by comparing the network output (label forecast) with the expected label value, for each record among the input data; the various comparisons are then reduced to a single value (based on an appropriate metric), which corresponds to the global error that the network demonstrates on the set of input data. Having identified this error value, the network will have to correct the various weights in order to minimize it.

Mean absolute error (MAE)

A metric often used to calculate the error is the **mean absolute error**, defined as follows:

$$\frac{1}{N}\sum |y - y'| \tag{3}$$

Where:

- **N** is the number of records in the input data;
- **y** is the expected forecast;
- **y'** is the prediction of the network.

In other words, it is the average of the errors of each single record, calculated as the modulus of the difference between the expected value and the predicted value. The advantage of this function (the part within the summation) is that the error is easily quantifiable, but it has a point of discontinuity.

Gradient calculation

Following the determination of the error that the network demonstrates on the data, it is needed to find a technique to update the weights so that the error decrease. This can be accomplished by calculating the gradient, to locate in which direction the error decreases (*gradient descend*) [11]. Assume that the network has only one input *feature* and, consequently, only one weight; also suppose to initialize this weight w_1 randomly, and, calculating the error (*loss*), to be in the starting point of Figure 11.

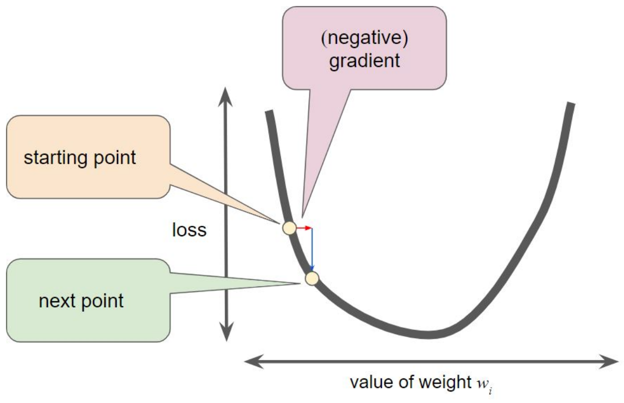


Figure 11: Graph describing the error variation when varying the weight w_1.

We now want to identify a new point, corresponding to a given value of w_1, such that the associated error is minimal.

A method that can be used to locate the minimum point is the calculation of the gradient, i.e. the derivative, in this case with respect to w_1 and at point p. The derivative mainly serves to identify in which direction to move, i.e. if increase or decrease w_1. In the example, since at the point p the function decreases, the derivative will be negative and w_1 will have to be incremented. In general if the derivative is negative w_1 must be increased while if positive it must be reduced; this allows to move w_1 towards the minimum of the function. However, it must also be established how much to increment or decrement w_1; this is defined through a parameter, called *learning rate* [12]. At each iteration, the value of w_1 will therefore be changed based on the following formula:

$$w_1 = w_1 - \alpha \frac{\partial f(w_1)}{\partial w_1} \tag{4}$$

where α is the *learning rate*. A learning rate that is too small increases the learning times training, as the weights vary too slowly with each step; a learning rate too large, on the other hand, could prevent the model from converging, since the point of minimum is constantly "skipped".

In the example there was a single weight; however, the same reasoning can be also adopted if there are more input features, and therefore more weights, considering the various partial derivatives with respect to each single weight $w_1, w_2, ..., w_n$.

Also, in the example, it was assumed to calculate the error on all data records of input; in a real case, where the input records might be present in number very high, it would be inconvenient to calculate the error for each of them. Thus, input records are usually split in *batches*, often randomly; for each batch then calculates the error and updates the weights. The *batch size* is then another adjustable parameter, which could affect the efficiency of the process training: too small could cause incorrect updating of the weights, too large could make training times intolerable. The batch size characterizes some training techniques [13]: *stochastic gradient descent (SGD)* where the batch size is 1 and *mini-batch stochastic gradient descent (mini-batch SGD)* where the size of the batch is usually between 10 and 1000.

4.5 Overfitting, sets of training, validation and tests

In the example mentioned in the previous sections, it was assumed that the *training* phase was being performed on all available input data. In fact, this approach can cause problems, as the model may overfit to the set of data provided to it, behaving negatively with new data. This phenomenon is called *overfitting* [14]: the network over-adapts to the characteristics of the data on which the training phase was done, losing the ability to generalize correctly on new examples.

One solution to solve this problem could be that to divide the input data into two distinct sets: *training set* and *validation set* [15]. The *training* set will be the one that will have the most records (for example you could divide respectively into 80%-20%). The examples contained in the *training* set must not appear in the *validation* set and vice versa; also the characteristics of the data present in the two sets should be similar. At each iteration of the training process, the error on the *training* set is then calculated, updating the weights accordingly. At the end of the weight update, the model is evaluated on the *validation* set and the procedure is repeated.

The best weight configuration will be the one that produces the minor error on the *validation* set. Proceeding in this way, however, there is a risk of overfitting on the *validation* set. To further reduce this probability, the input data can be further divided into three distinct sets: *training* set, *validation* set and *test* set [16]. In this case we proceed exactly as described above, but, at the end of the training process, the network is tested on the *test* set, to verify its performance. Using this technique it is easy to determine if the model is affected by overfitting on training data; in this case, in fact, it will be possible to notice, during the training phase, a decrease in the error on the *training* set, but a continuous increase of the error on the *validation* set, as shown in Figure 12.

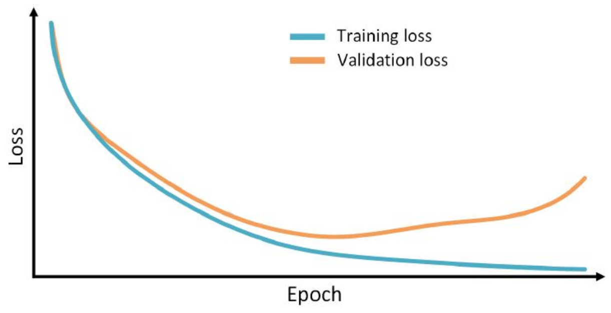


Figure 12: Overfitting example during the training phase.

However, it is necessary to specify that overfitting can only occur if the model is complex enough to fully adapt to characteristics of the *training* set. Another good method to contrast this kind of phenomenon is therefore to reduce the complexity of the model; this topic will be covered more fully in the *Regularization* section.

4.6 Feature engineering

To get the best results, you need to feed the network with the right data format. This implies the definition of an adaptation procedure of the data, which allows, starting from the data in their original format, to obtain the records with the features to provide to the model; this procedure, called *feature engineering* [17], could also prove to be very complex and laborious.

One of the initial things to do is the choice of *input features*. Use all the features present in the original data usually is not the best way to proceed; in fact the aim should be to minimize the number of features on which the model must depend, to reduce the likelihood of having to change it if there is a change in the characteristics of the input data. Furthermore, the various features should be independent from each other, to avoid that a subset of these essentially provide the same information. The values of any chosen feature should appear at least, individually, a certain number of times to allow the network to understand the relationship that this value has with the output [18]. *Magic numbers* are to be avoided: in their place it is necessary to insert an additional feature that indicates whether the datum it is present or not.

Numerical features present in the original data can be converted directly in the corresponding *features* to be provided to the network, converting each numerical value into a *float* data type [17]. It could be convenient though, especially if such feature take on very high values, scale them appropriately to lower values, for example between zero and one. This would allow the network to converge faster to value of the optimal weights, to prevent some weights from becoming *NaN* and to prevent features with higher values they take over the others [19].

4.7 Regularization

As already described, you might run into the problem while training your network of overfitting. The division of the input data into the three sets (*training* set, *validation* set and *test* set) allows to identify the *overfitting* on the *training* set and to mitigate it, considering as the best the weights configuration which produces the minor error on the *validation* set. However, there are other ways to avoid overfitting: one of these is the so-called *regularization*. Overfitting on the *training* set is in fact possible only if the model is sufficiently complex to being able to fully adapt to the characteristics of the training data. The idea is therefore that of reducing, during training, the complexity of the model, regularizing it [22]. The main goal of the training phase is to minimize the error:

$$minimize(loss) \tag{5}$$

Instead, adopting a regularization technique will not only minimize the error, but will also minimize the complexity of the model:

$$minimize(loss + complexity) \tag{6}$$

It is also very useful to be able to control the reactivity of the regularization. This control takes place through a parameter λ (*regularization rate*) [20], which is multiplied by the complexity; it defines the influence that the complexity of the model will have in the minimization process:

$$minimize(loss + \lambda complexity) \tag{7}$$

The λ value can heavily affect the quality of the results: a value too small could cause overfitting, while a valute too large could cause underfitting, i.e. the model is made too simple for the type of prediction to carry out. The various regularization techniques differ, mainly, for the ways in which they represent the complexity of the model.

4.8 Simple Neural Network

In the previously considered example of linear regression there was a single neuron, which operated by multiplying the inputs by a given weight and adding them together. The output was then a simple linear combination of the input. One might think of increasing the complexity of such a model by adding other neurons, which, usually are grouped in layers. Each layer receives inputs from the previous layer and provides the output to the next layer. The initial layer, connected to the input data, is called the *input layer*, while the last one, which supplies the network output, is called *output layer*; among these there can be various *hidden layers*, which allow to increase the complexity and the potential of the network. An example of a layered network is shown in Figure 13.

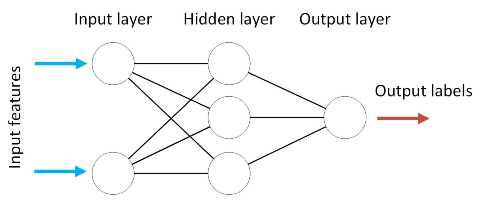


Figure 13: Example of a multilevel neural network.

Even in the case of Figure 13 however, using the simple neurons described so far, it obtains a linear combination of the input data as output [23]. If the goal was to describe a complex non-linear relationship between input and output, it would be necessary to add some component that models its non-linearity. An adoptable solution is *feature crossing* [24]: they are added to the network input other artificial features, obtained by applying non-linear functions to some of the original features. This technique can produce good results, but in complex cases it may be non-trivial to determine which artificial features to add. A more general solution is to define not-linear *activation functions* for each neuron [23]; this function will be applied to the output of the neuron before it is passed on to the next neuron. Usually the same *activation function* is applied to the neurons of the same layer. These functions are chosen based on the needs and allow to model a non-linearity within the network. By superimposing a series of non-linear layers, it is possible to obtain an extremely complex model, capable of understanding even very complicated relationships. During the training phase, a back-propagation technique is used to correct the weights: starting from the output nodes, the various gradients are calculated for each layer, updating the weights accordingly, until the input layer is reached.

4.9 Activation functions

Some of the commonly used *activation functions* [23] are, for example, the *sigmoid* function and the *ReLU* function. The *sigmoid* function generates values always between 0 and 1; it is defined in Eq. 8 and shown in Figure 14 as follows:

$$f(x) = \frac{1}{1+e^{-x}} \tag{8}$$

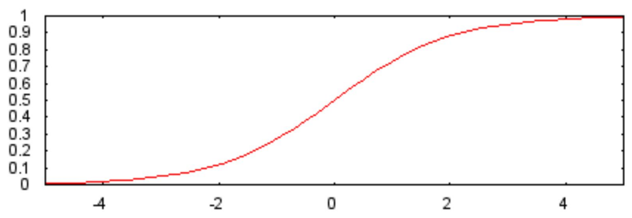


Figure 14: Sigmoid function graph (source [23]).

The *ReLU* function is instead defined in Eq. 9 and shown in Figure 15:

$$f(x) = max(0, x) \tag{9}$$

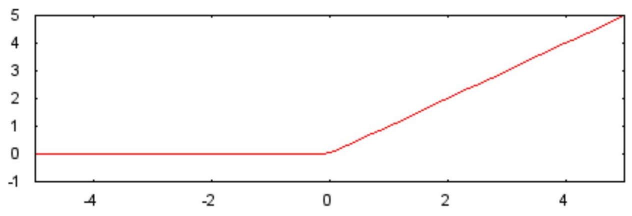


Figure 15: ReLU function graph (source [23]).

4.10 Recurrent neural network

The networks previously considered associate a given input to a given output. The order in which the inputs are presented to the network does not affect the result: each record is independent and the information to calculate the output must be contained entirely in it. But there are problems where the result can depend on the particular sequence in which the data is presented; in this case then the actual output will depend not only on the actual input record, but also on the past history. An example could be trying to predict the next word in a particular sentence. Obviously, in this case, it would not be entirely sufficient to know only the last word in order to be able to predict the next: you need to have some more information about the words that have occurred recently, to determine the context of the sentence. A solution could be to pass the entire sentence to a network of similar structure to those seen previously.

However, two problems arise: the first is that the sentences could be of variable size; the second is that a sentence too long might involve defining a model with a very large number of input features. The need therefore arises for a network capable of operating on an arbitrary sequence of data and to abstract and store information about the order in which the data occur. Such networks are called *recurrent neural networks*. They are very useful to make predictions where the sequence of the data is important, such as stock or weather forecasts. A recurrent neural network is formed by a series of *recurring cells*: a cell is comparable to a neuron of neural networks seen earlier, but has a higher complexity (in fact inside it there are other small neural networks capable of maintaining a memory).

4.11 Simple recurrent cell

A recurring cell can be thought of as a neural network, which provided an input returns an output. Among the inputs, however, there are not only the *input features*, but also the *previous state* of the network. This allows the network to maintain an internal memory, which stores some information relating to the data previously examined. The scheme of a simple recurring cell is shown in Figure 16.

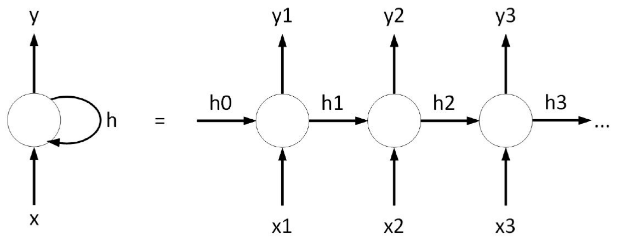


Figure 16: Simple cell of a recurrent neural network. Left side: compact notations. Right side: extended notation.

Although a network formed by cells of this type has a lot of potential, it suffers some problems related to memory persistence and gradient calculation. As for the neural networks seen previously, for updating the weights it is used a *back-propagation* technique. In this case, however, the output also depends from the previous state of the network, so the calculation of the gradient also propagates to the previous states; this process is called *back-propagation through time (BPTT)*. Since the previous states could be numerous, it could occur a vanishing gradient or exploding gradient problem: the first case occurs if the gradient reaches too small values, the second if it reaches too high values, becoming *NaN* (not a number).

This can happen since the calculation of the gradient can be decomposed into a product of other gradients inherent to the previous state, which in turn can be decomposed into products of gradients inherent to the previous states. If the values to be multiplied are less than zero, a few multiplications are enough to reach an extremely small value; vice versa, by multiplying many values greater than zero, the limits of a floating point could easily be exceeded. The *vanishing gradient* problem causes limited persistence of the memory of the network, since the data seen in a distant period will contribute minimally to network training, having a very low gradient.

4.12 LSTM and GRU cells

Problems related to simple repeating cells are solved by more complex cells: *LSTM (Long Short Term Memory)* and *GRU (Gated Recurrent Unit)*. These cells internally use the so-called *gates*, which allow to select which input or output signals to take into consideration by adjusting their intensity. GRU cells are slightly simpler than LSTM, but they get very similar results with lower training costs. The LSTM and GRU cells are interchangeable without problems; the choice between them can be determined on the basis of performance on the specific problem.

5 Project goal

The main objective will be to define a model capable of making predictions on a particular air pollutant. To make the predictions, the techniques described in Section 4 will be adopted.

The prediction will be performed through the use of some *artificial neural network*: the training of the network will take place by providing it with a series of measurements of the pollutant, plus some other features corresponding to the meteorological data, relating to a given time interval; starting from these data, it will therefore have to try to predict a future measurement of that particular pollutant, then comparing it with the real one (*supervised learning*), which will be present among the measurements stored in the *Pandas* dataframes.

The most suitable pollutant on which to make this prediction seems to be *nitrogen dioxide* NO_2, mainly because it is the compound with the highest number of associated stations and which has the greatest number of records. There are also many stations for PM_{10}, but the frequency with which the measurements of this pollutant are carried out is much lower (daily), and, for this reason, the number of records associated with it is much smaller; this scarcity of data would make training the neural network difficult or even impossible.

Measurements relating to *nitrogen dioxide* are present on an hourly basis. The goal set is to predict, starting from a set of records with measurements up to a given hour, the NO_2 pollution value after 24 hours; in this case, the forecasting capabilities of an artificial neural network could be severely challenged.

The place or places where the predictions will be made will correspond to the locations of the *nitrogen dioxide* monitoring stations: we will focus exclusively on the area of the province of Rimini, where there are three of them. The network will therefore return exactly three outputs, each corresponding to the pollution value expected for each station in Rimini. This approach was chosen because the spatial density of the various stations is very low. The geographic coordinates relating to each monitoring station will not be provided as input to the network, since they do not provide any significant information that could improve the forecast.

The final objective will be therefore to obtain a model, in the form of a neural network, capable of predicting the concentration level of *nitrogen dioxide* in 24 hours, for each of the three stations present in the province of Rimini.

The *input data* to be supplied to the network to make this forecast will be a set of records, relating to a given time interval of arbitrary length, containing the values of *nitrogen dioxide* concentration, temperature, humidity, precipitation, wind speed and wind direction for each station. To provide the network with structured data in this way it will be necessary to define an adequate interface and conversion procedure. The type and structure of the neural network to be used are not well known; it will therefore be necessary to carry out a series of experimental tests, in order to identify a sufficiently adequate model.

6 Design

Taking into account the objectives and conditions defined in Section 5, we will define the techniques adopted to prepare the data so that they can be used as input to an *artificial neural* *network.*

6.1 Artificial features: time information

To help the network in the forecasting operation, some *artificial features* obtained from temporal information will be added because the value of the pollution should depend partially on these. The *features* that will be added are:

- **Month** Integer value between 1 and 12. From the month it is possible to trace back to the season, and the season can determine different weather conditions, allowing the dispersion of pollutants with greater or lesser speed; moreover, the colder months will be characterized by more pollution, due to domestic heating systems;
- **Day of the week** Integer value between 1 and 7. The day of the week could provide an estimate of the pollution; in fact, working days could be characterized by greater air pollution;
- **Hour** Integer value between 0 and 23. The time could also provide an estimate of the pollution: probably, in peak hours, the air will be much more polluted than at night.

6.2 Label extraction

In Section 5, the goal was set to make a prediction 24 hours later on the most recent record in the sequence fed to the network. In order to carry out the *training* phase, the network must therefore have the real value with which to compare the prediction (being *supervised learning*).

It will therefore be necessary to generate a new *dataframe* which contains, for each record of the previously generated table (Fig. 9), the corresponding label to be provided. The label for each record will consist of three distinct values, since the network must predict the NO_2 concentration for each of the three stations in Rimini. To generate the table containing the labels, it is sufficient to extract the columns relating to the *nitrogen dioxide* measurements from the previously generated table (Fig. 9) and subsequently shift them by minus 24 hours; this is equivalent to eliminating the first 24 records from the label table and the last 24 records from the input data table (which would not have associated labels).

A simplified example of such a procedure, considering a forecast two hours into the future and two monitoring stations, is shown in Table 2.

Station	10000001		10000060	
Parameter	8	1003	8	1003
DATA_INIZIO				
2010-01-01 00:00	41	0.0	29	0.0
2010-01-01 01:00	48	0.2	33	0.2
2010-01-01 02:00	53	0.4	34	0.0
2010-01-01 03:00	53	0.2	35	0.2
2010-01-01 04:00	53	0.4	36	0.1
2010-01-01 05:00	52	0.0	36	0.0

Station	10000001	10000060
Parameter	8	8
DATA_INIZIO		
2010-01-01 02:00	53	34
2010-01-01 03:00	53	35
2010-01-01 04:00	53	36
2010-01-01 05:00	52	36
2010-01-01 06:00	null	null
2010-01-01 07:00	null	null

Table 2: Left: Input data to provide to the network (some attributes have been omitted for simplicity). Right: *labels* of the corresponding input data, in this example two hours in the future.

The correspondence between the input values in Table 2 (left) and the labels in Table 2 (right) is determined by the row number; when such tables are converted to multidimensional arrays, the match will be determined by their position in the array (*index*).

6.3 Sequence generation

In reality, the network input must not be a single record, but a contiguous sequence of them relating to a given time interval. Therefore, starting from the input data table (Table 2 left) and from the table with the *labels* (Table 2 right), sequences of records will be extracted and, to each one, the respective *label* to be expected will be associated; it will correspond to the *label* relating to the last record present in the sequence.

For example, considering Table 2 (left) and sequences of length three, it is possible to extract from it two sequences, shown in Table 3 (left) and in Table 4 (left); the relative labels are shown, respectively, in Table 3 (right) and in Table 4 (right).

Station	10000001		10000060	
Parameter	8	1003	8	1003
DATA_INIZIO				
2010-01-01 00:00	41	0.0	29	0.0
2010-01-01 01:00	48	0.2	33	0.2
2010-01-01 02:00	53	0.4	34	0.0

Station	10000001	10000060
Parameter	8	8
DATA_INIZIO		
2010-01-01 04:00	53	36

Table 3: Left: First sequence extracted from Table 2 (left). Right: *labels* associated to the first sequence, obtained from Table 2 (right).

Station	10000001		10000060	
Parameter	8	1003	8	1003
DATA_INIZIO				
2010-01-01 01:00	48	0.2	33	0.2
2010-01-01 02:00	53	0.4	34	0.0
2010-01-01 03:00	53	0.2	35	0.2

Station	10000001	10000060
Parameter	8	8
DATA_INIZIO		
2010-01-01 05:00	52	36

Table 4: Left: Second sequence extracted from Table 2 (left). Right: *labels* associated to the second sequence, obtained from Table 2 (right).

As can be seen, the sequences are partially overlapping, since some records are the same; despite this, two different and independent inputs are still considered. In the example, sequences of length three have been extracted, but this value can be adjusted as desired, bearing in mind that it could affect the quality of the results.

A reasonable value, which will be used in experiments, is 72 (namely 24 * 3); this means that the measurements of the last three days will be used to forecast pollution in 24 hours. From now on, a *sequence* will be considered a single example to be provided to the network, during the *training* or *evaluation* phase. To obtain better performances, as already described in Section 4, all data will also be scaled between 0 and 1, through the *MinMaxScaler* class made available by *Scikit-learn.*

6.4 Data splitting: training, validation and test sets

In general, in order to train a *neural network* it is necessary to divide the input data into *training*, *validation* and *test* sets. For this study it was decided to proceed with the subdivision of the generated *sequences*, so that all possible *sequences* are first extracted from the input data, subsequently these *sequences* will be divided among the three sets. This approach allows the *sequences* to be randomly shuffled, since the temporal sequentiality is contained within the single sequence. This approach makes it possible to obtain sets with characteristics that are more similar to each other, because it allows the sequences to be distributed randomly.

Actually, in the tests carried out in the next section, only the *training* and *validation* set sequences were randomly distributed, since, otherwise, it would not have been possible to generate a viewable graph of the predictions on the *test* set.

For the training phase it was also necessary to define the *batch size*, i.e. how many sequences to examine for updating the weights. This is also an adjustable value which can affect the quality of the results; a sufficiently large batch size could in fact allow updating the weights of the network in a better way. At the same time, a too large batch size could saturate the computer's memory, making the training phase effectively impossible. In the various experiments a *batch size* of 128 was used.

7 Experimental tests and model choice

After the manipulation of the data structure (to make them usable by a *neural network*) it will be possible to proceed with the definition of the model that will have to carry out the forecasting operation. However, the most suitable structure for this model is not well known a priori, and for this reason we will proceed with a series of experimental tests of the network typologies defined in Section 4.

Usually, you start with a very simple model and add complexity to it only if really necessary: a too complex model would increase the amount of resources needed to carry out the training phase and there could be the risk of falling into the *overfitting* problem. To compare the various models, the *MAE (mean absolute error)* demonstrated on the *test set* will be taken into consideration, because it generates easily comparable and interpretable values.

For simplicity, the model with the lower *MAE* will be considered the best in the various next experiments. Each experimentation will be characterized by a *training phase* of 20 *epochs*; each *epoch* is made up of 100 training phases (*steps*). Each training phase will update the analyzed *weights* by a number equal to the block size of sequences, which in this case is 128. At the end of each *epoch*, the model will be evaluated on the *validation* set, each time maintaining the configuration with the lower *MAE*. At the end of the 20 *epochs*, the model will be evaluated on the *test* set, calculating the *mean absolute error* on it. Furthermore, an *Adam optimizer* will be used in the training phase; the *learning rate* will be adjusted automatically during training by defining appropriate callbacks.

The following subdivision of the various sets was adopted in the various experiments: first the contiguous input records to be assigned to the *test* set were extracted; from the remaining ones all the possible sequences were generated, then subdividing them randomly between *training* and *validation* sets.

7.1 Linear neural network

The simplest *neural network* model is characterized by only three neurons, one for each output and without *activation function*, hence, linear. Before this level, it is necessary to add a *Flatten* layer, which takes care of concatenating the various records of the sequence: otherwise it would not have been possible to supply a *sequence* of records to a network of this type. In this case, each record in the sequence contains 21 features (3 NO_2 concentrations, 15 weather related, 3 artificial features) and the sequence consists of 72 records (records for 3 consecutive days). The *Flatten* layer concatenates these records, resulting in an input of 21 * 72 = 1512 features. This means that each of the three neurons will have 1512 input connections, each with an independent *weight*. The adjustable parameters (weights) are therefore 1512 * 3 = 4536 plus the *bias* relative to each neuron (1 * 3 = 3), for a total of 4539 adjustable parameters. The structure of the model is shown in Listing 7.1.

Listing 7.1: Structure of the linear neural network under test.

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten (Flatten)           (None, 1512)              0

 dense (Dense)               (None, 3)                 4539

=================================================================
Total params: 4,539
Trainable params: 4,539
Non-trainable params: 0
_________________________________________________________________
```

The result of the *training* phase is shown in Listing 7.2.

Listing 7.2: Model evaluation (error) of the linear neural network.

```
Last training epoch: 1s 10ms/step - loss: 0.0531 - val_loss: 0.0538
Best MAE on validation: 0.053839731961488724

MAE loss (test-set): 0.058183543384075165
MAE loss (test-set rescaled): 8.843898594379425
```

The *MAE loss* value corresponds to the *mean absolute error* calculated on the *test* set scaled between 0 and 1, while the *MAE loss rescaled* value corresponds to the error with respect to the real values of the *labels* (not scaled between 0 and 1): this means that the forecast is wrong, on average, by 8.84 $\mu g/m^3$.

This is the simplest model under test since it has low complexity (only three neurons). It can therefore be considered as a reference to compare with the other models. Below it is shown the graph that describes the variation of the *MAE* during the training phase (Figure 17) and an example of prediction on the *test set* (Figure 18). As can be seen from Figure 17, there is no overfitting problem and the predictions of Figure 18 are not always correct; this probably indicates an insufficient complexity of the model.

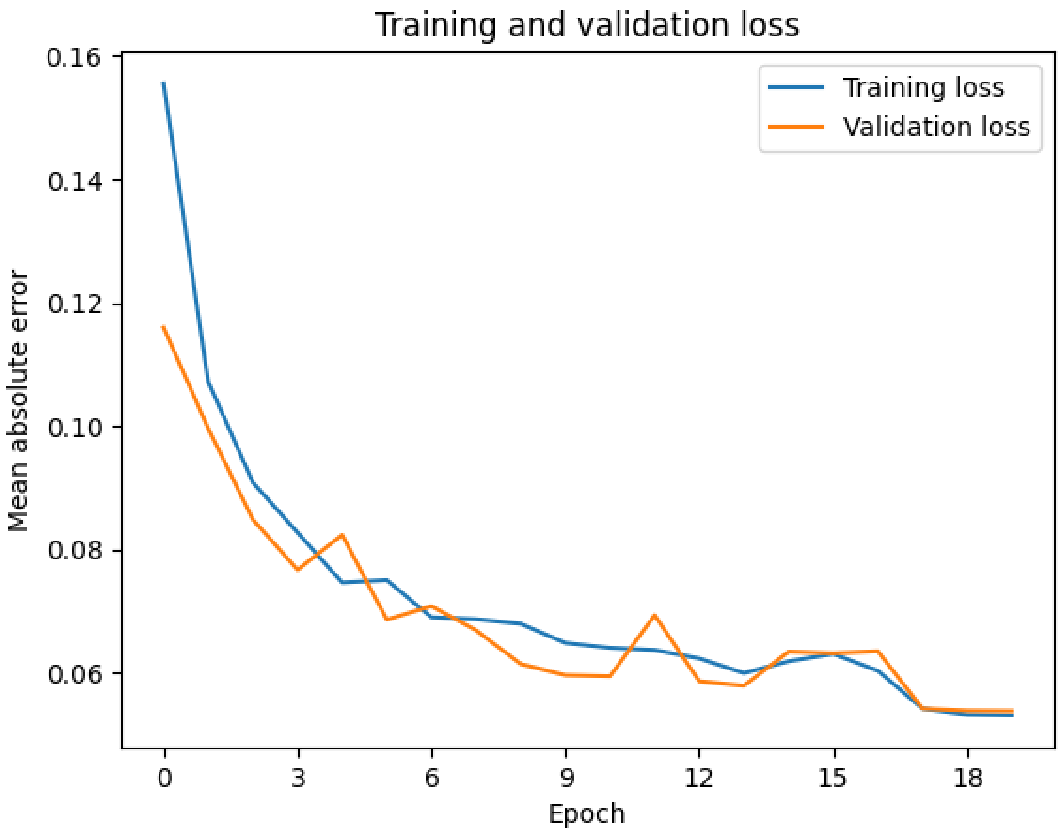


Figure 17: MAE error during *training* epochs.

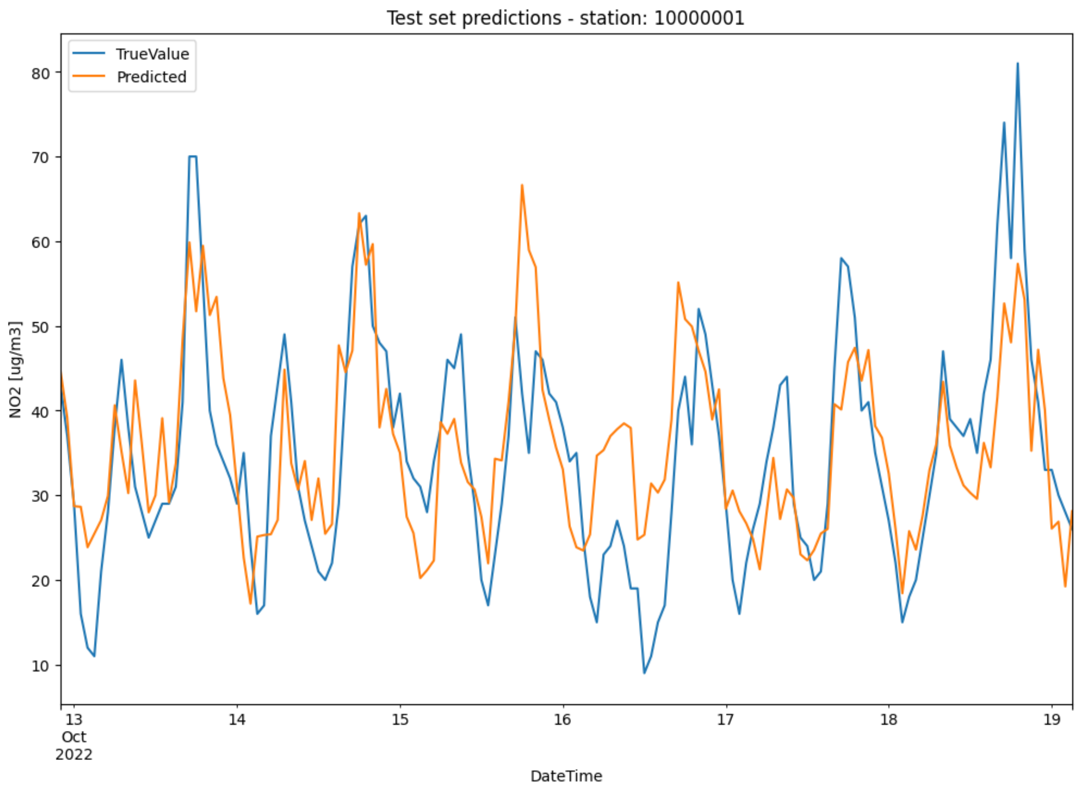


Figure 18: Prediction on part of the *test set* for one of the three monitoring stations (Flaminia).

7.2 Non linear multilevel neural network

Subsequently, a more complex network was defined, composed of two *hidden layers*, each with 32 neurons, plus the 3 neurons of the *output layer*. The *activation function* associated with the *hidden layers* is a *ReLu*, while a *sigmoid* has been associated with the *output layer*. The *activation function* of the *output layer* produces values between 0 and 1; this helps the network to obtain more accurate predictions, since both the input *features* and the *labels* were scaled between 0 and 1. Also in this case an initial *flatten layer* was necessary, for the reasons already described. The structure of the model is shown in Listing 7.3.

Listing 7.3: Structure of the non linear multilevel neural network under test.

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten (Flatten)           (None, 1512)              0

 dense (Dense)               (None, 32)                48416

 dense_1 (Dense)             (None, 32)                1056

 dense_2 (Dense)             (None, 3)                 99

=================================================================
Total params: 49,571
Trainable params: 49,571
Non-trainable params: 0
_________________________________________________________________
```

As can be seen from Listing 7.3, this model is much more complex than that shown in the previous section: about 10 times more (almost 50000 parameters). The result of the *training* phase is shown in Listing 7.4.

Listing 7.4: Model evaluation (error) of the non linear multilevel neural network.

```
Last training epoch: 1s 11ms/step - loss: 0.0474 - val_loss: 0.0481
Best MAE on validation: 0.048072218894958496

MAE loss (test-set): 0.052240461111068726
MAE loss (test-set rescaled): 7.940550088882446
```

In this case, the error is smaller than the linear model; this means that such a network can make better predictions. Below it is shown the graph describing the variation of the *MAE* during the *training* phase (Figure 19) and an example of a forecast on the *test set* (Figure 20). This model seems to produce smoother predictions, but is limited because it requires an initial *flatten layer*. This limits the maximum length of the input sequence, as the complexity of the model increases proportionally with it.

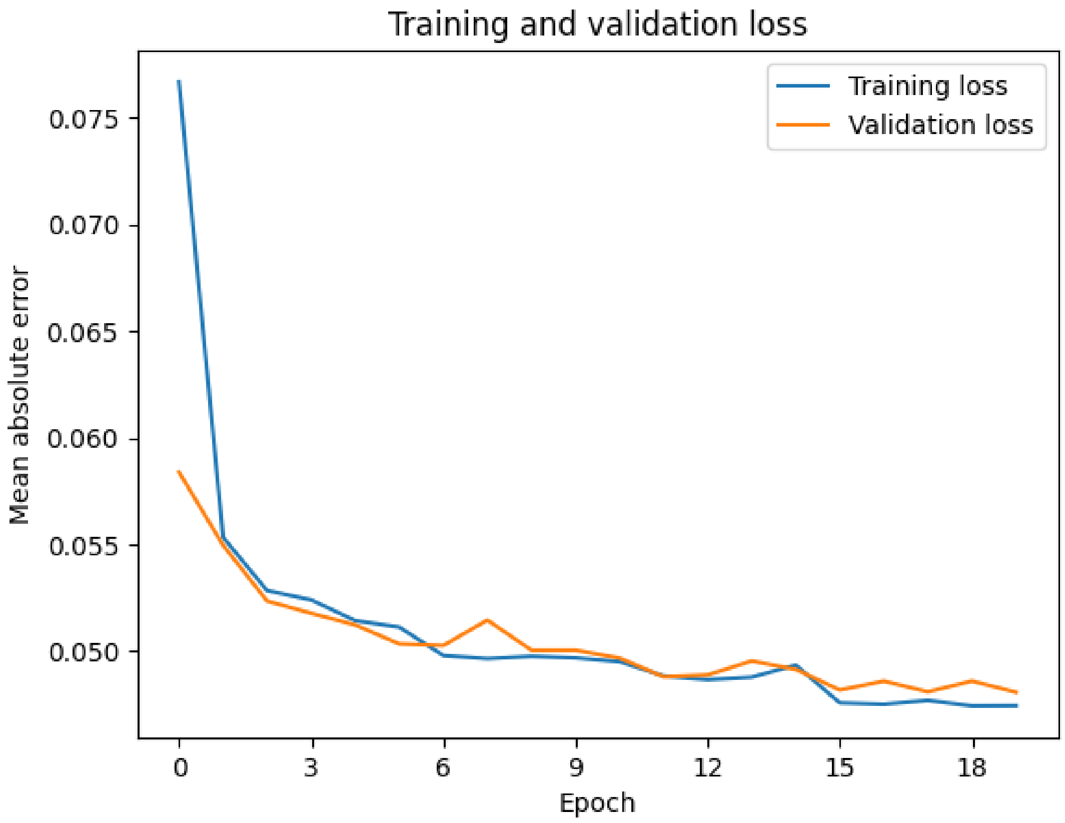


Figure 19: MAE error during *training* epochs.

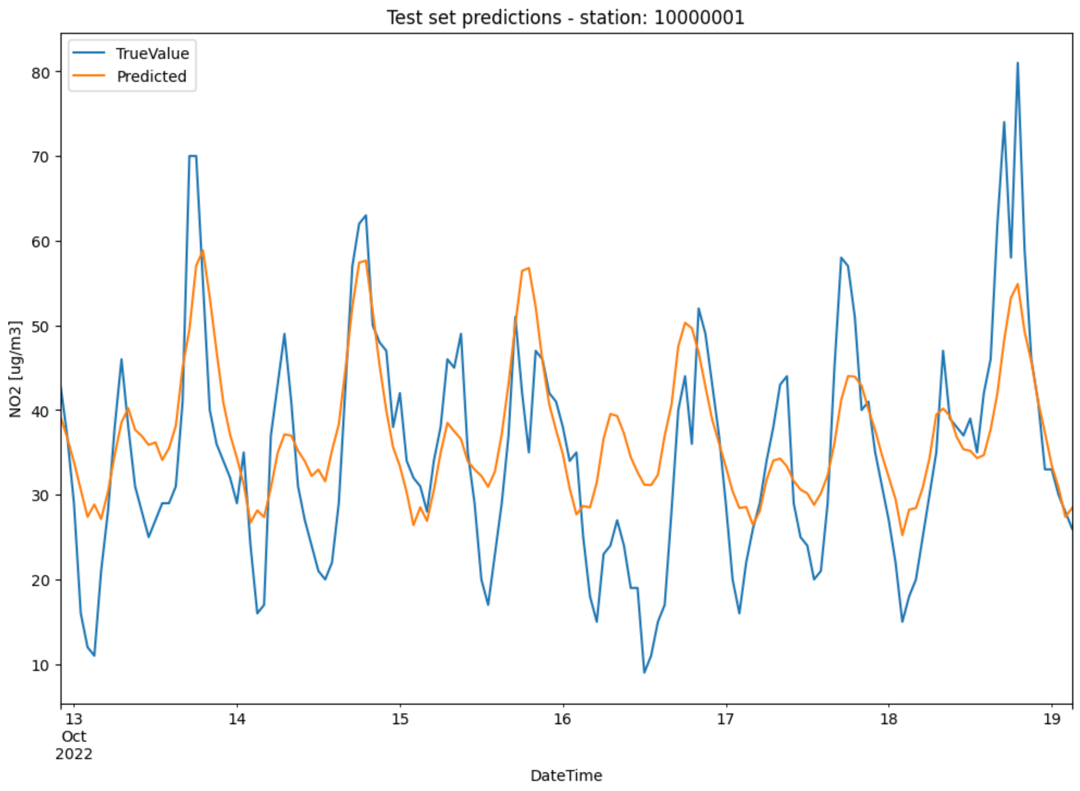


Figure 20: Prediction on part of the *test set* for one of the three monitoring stations (Flaminia).

7.3 Recurrent neural network

The most suitable type of network for this problem should be a *recurrent neural network*. For this test, a model with a *hidden layer* composed of 32 *GRU cells* and a normal *output layer* with three neurons was therefore defined to generate the three output values. In this case the *flatten layer* is not necessary, since *GRU cells* accept sequences of records as input. The structure of the model is shown in Listing 7.5.

Listing 7.5: Structure of the recurrent neural network under test.

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 gru (GRU)                   (None, 32)                5280

 dense (Dense)               (None, 3)                 99

=================================================================
Total params: 5,379
Trainable params: 5,379
Non-trainable params: 0
_________________________________________________________________
```

As can be seen from Listing 7.5, this model has a complexity equal to the first examined (about 5000 parameters). The result of the *training phase* is shown in Listing 7.6.

Listing 7.6: Model evaluation (error) of the recurrent neural network.

```
Last training epoch: 9s 87ms/step - loss: 0.0481 - val_loss: 0.0485
Best MAE on validation: 0.04818357899785042

MAE loss (test-set): 0.05240798741579056
MAE loss (test-set rescaled): 7.966014087200165
```

In this case, the error is larger than in the *multilevel neural network* examined previously, but it is still smaller than the linear neural network. Also, despite having the same number of parameters as the first linear neural network tested, it performed better. Considering that a model of this type could handle sequences of arbitrary length without problems, it is easy to affirm its superiority compared to the models seen previously, at least for this type of problem. However, it should be noted that the time required for the training phase was greater: almost 10 seconds per step, against the 1 second per step of the initial linear model. Below it is shown the graph describing the variation of the *MAE* during the *training phase* (Figure 21) and an example of a forecast on the *test set* (Figure 22). Again, there appear to be no overfitting issues; one could therefore think of further increasing the complexity of the model.

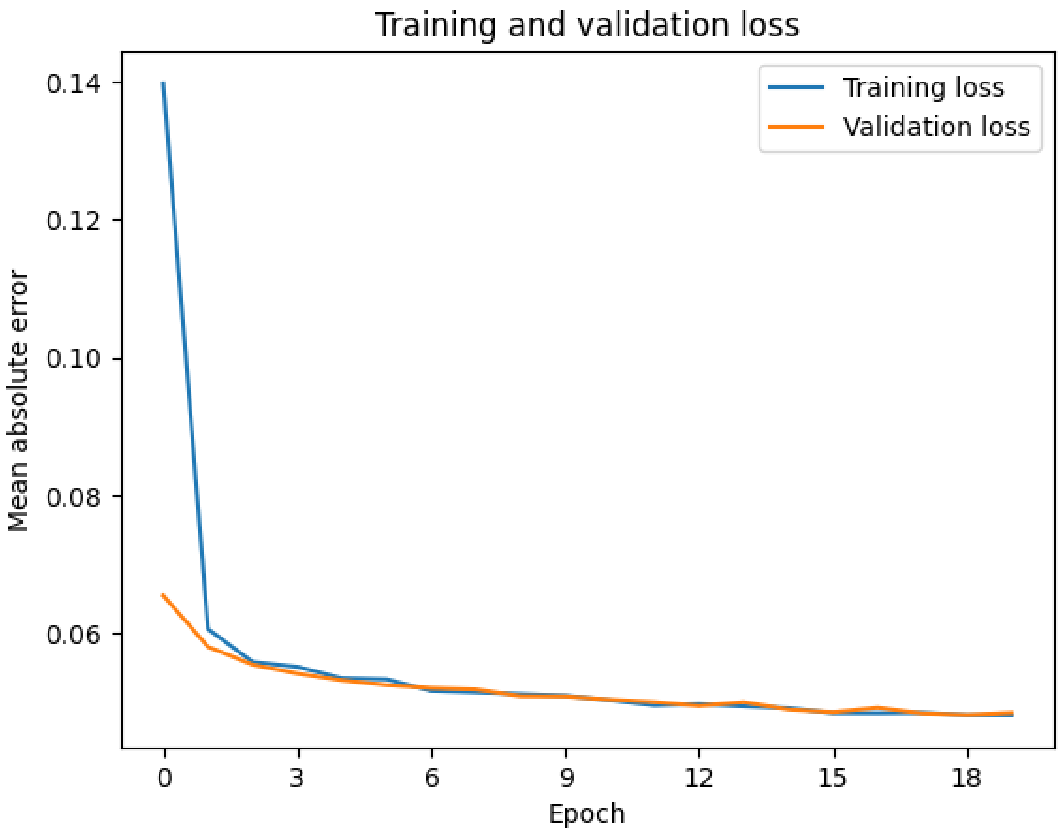


Figure 21: MAE error during *training* epochs.

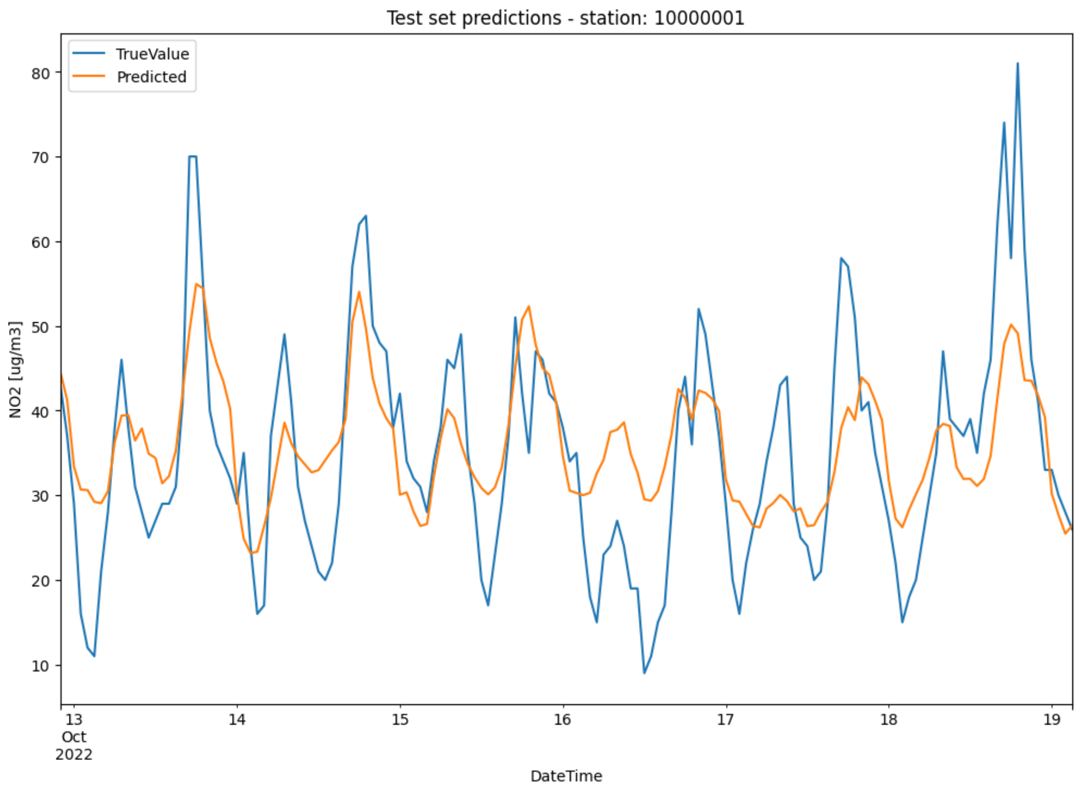


Figure 22: Prediction on part of the *test set* for one of the three monitoring stations (Flaminia).

7.4 Multilevel recurrent neural network

As long as no *overfitting* issues are encountered, it makes sense to increase the complexity of the model. By overlaying multiple *GRU* levels, a more complex model can be achieved. In this test, a network was built consisting of a first *hidden layer* with 64 *GRU cells* and a second one with 32 *GRU cells*, in addition to the normal *output* layer with three neurons. As with the previous case, no *flatten layer* is needed. The structure of the model is shown in Listing 7.7.

Listing 7.7: Structure of the multilevel recurrent neural network under test.

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 gru (GRU)                   (None, 72, 64)            16704

 gru_1 (GRU)                 (None, 32)                9408

 dense (Dense)               (None, 3)                 99

=================================================================
Total params: 26,211
Trainable params: 26,211
Non-trainable params: 0
_________________________________________________________________
```

As can be seen from Listing 7.7, this model has a complexity equal to half of the *multilevel neural network* seen above. The result of the *training phase* is shown in Listing 7.8.

Listing 7.8: Model evaluation (error) of the multilevel recurrent neural network.

```
Last training epoch: 18s 180ms/step - loss: 0.0475 - val_loss: 0.0475
Best MAE on validation: 0.04731962829828262

MAE loss (test-set): 0.05162404477596283
MAE loss (test-set rescaled): 7.84685480594635
```

In this case, the error demonstrated by the model on the *test* set is the smallest of those obtained so far. However, it should be noted that the increase in complexity has further lengthened the *training* times, almost doubling them compared to the previous test: from 10 seconds per step it has gone to 18 seconds per step. Below it is shown the graph describing the variation of the *MAE* during the *training* phase (Figure 23) and an example of a forecast on the *test set* (Figure 24). Also in this case there seems to be no *overfitting* problems; thus, the complexity of the model could have continued to be increased. However, to avoid having to manage the various problems related to a too heavy model (such as extended *training* times), it was decided to consider this configuration as the best compromise between model simplicity and forecast quality.

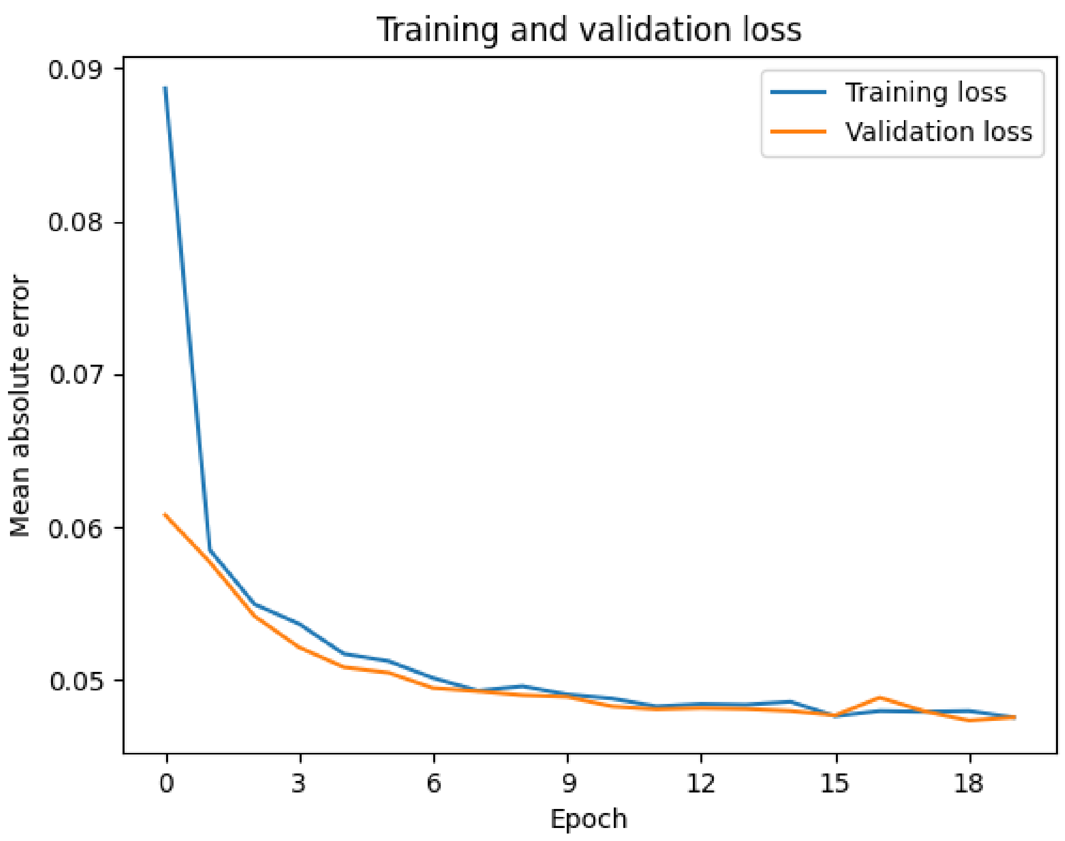


Figure 23: MAE error during *training* epochs.

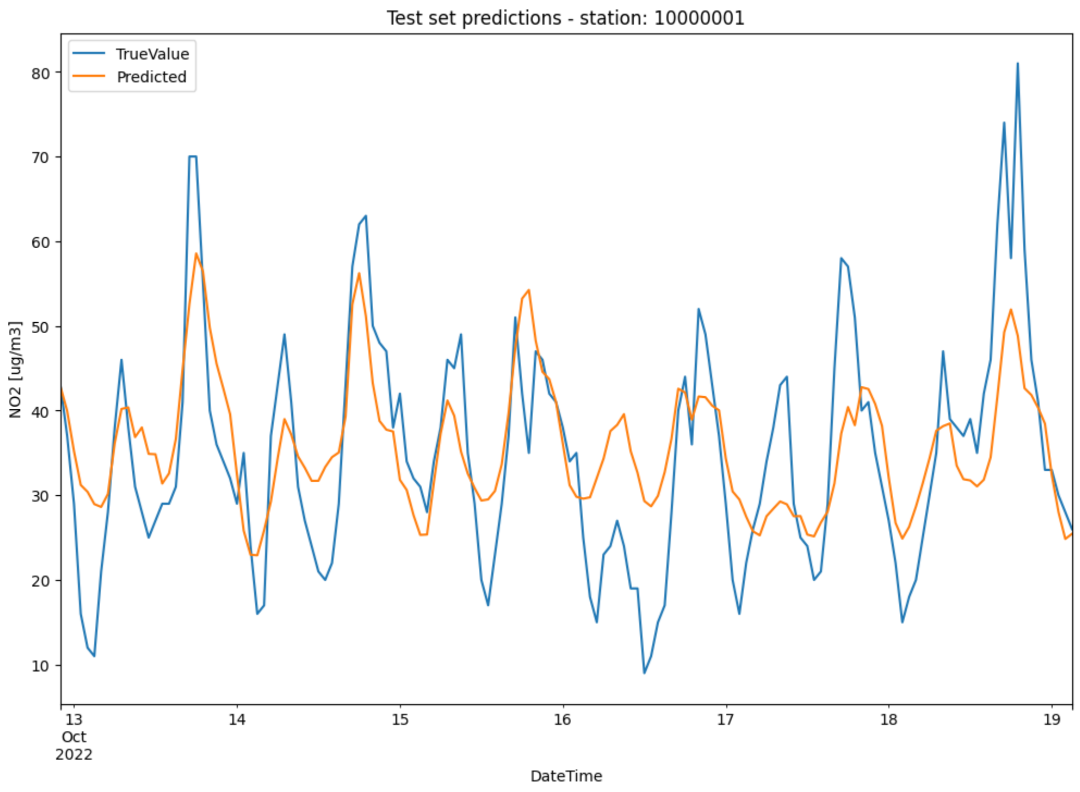


Figure 24: Prediction on part of the *test set* for one of the three monitoring stations (Flaminia).

7.5 Chosen model: output comparison

Since the *multilevel recurrent neural network* has been identified as the best model, an example of its entire output is shown above in Figure 24 and below in Figure 25 and 26. It consists of the forecasts of *nitrogen dioxide* concentrations for the three monitoring stations in the province of Rimini, namely stations number 10000001 (*Flaminia*), 10000060 (*San Clemente*) and 10000074 (*San Leo*).

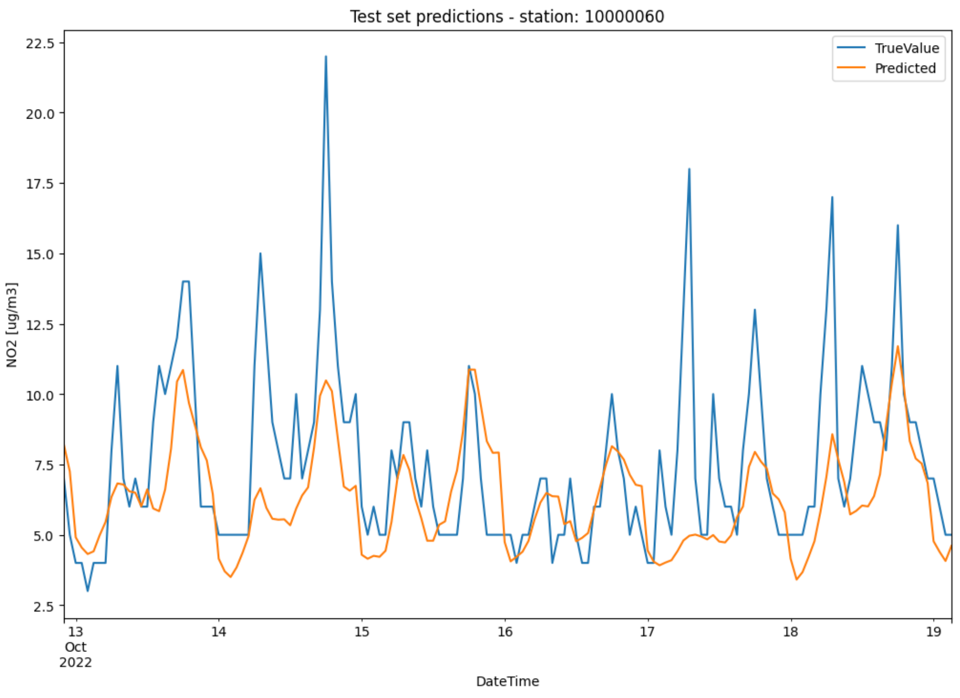


Figure 25: Prediction on part of the *test set* for one of the three monitoring stations (San Clemente).

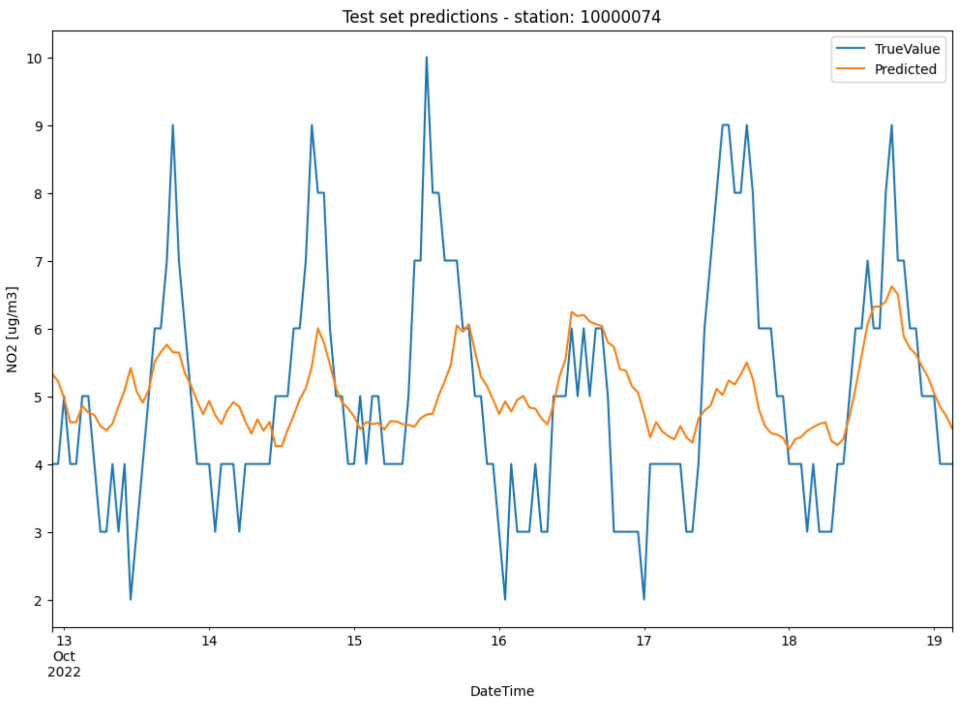


Figure 26: Prediction on part of the *test set* for one of the three monitoring stations (San Leo).

As can be seen, the forecasts trend is far more accurate for the *Flaminia* station (Figure 24) than for the other two. This may be due to two main factors:

- *weather stations association*: for the *Flaminia* station, the *Rimini urbana* meteorological station was associated (Figure 7), from which it was possible to find all five meteorological features. Those two stations (one of air quality, one of weather measurements) are very close to each other; the same cannot be said of *San Clemente* and *San Leo* air quality stations, to which the same weather station (*Mulazzano*) has been mainly associated, which however is located very far from *San Leo*. In fact, Mulazzano and San Leo are on opposite sides of the *Titano* hill. Hence, the non-correspondence of location between air quality monitoring stations and meteorological stations is underlined once again and this is likely to have caused poorer forecasts;

- *different type of monitoring station*: the *Flaminia* station is classified as *urban traffic* station, while *San Clemente* and *San Leo* ones are classified as *rural background* air quality stations. Consequently, the NO_2 concentrations recorded in the three stations can follow very different trends given the different measurement situations; moreover, in *rural background* stations the measured values are often very low causing an unevenly trend.

8 Conclusions

Starting from the data collected, a path was outlined that could lead, through the use of specific techniques and technologies, to the definition of a model capable of making air quality forecasts. The main ML techniques were then described on a theoretical level: neural networks. These techniques are based on the definition of a model, composed of a series of levels of interconnected neurons, capable of learning, through the administration of a set of examples, the relationships between expected inputs and outputs (in the case of supervised learning). This process is based on an error-minimisation operation between predicted and actual values, through the calculation of the gradient. Furthermore, since the network must learn from a set of meaningful examples, a preliminary data selection and manipulation operation is also required.

Analysing the data, it was realised that the number of air quality monitoring stations is not very high, while meteorological stations are present in greater numbers. Measurements are present on an *hourly* basis and the pollutant with the largest amount of data appears to be *nitrogen dioxide*. In order not to complicate the problem excessively it was decided to work on a restricted area, corresponding to the province of Rimini, which contains three monitoring stations with *nitrogen dioxide* measurements.

The final objective was to obtain a model, in the form of a neural network, capable of predicting the level of *nitrogen dioxide* concentration 24 hours in the future, for each of the three monitoring stations present in the province of Rimini.

An important part of achieving this goal was the manipulation of the collected data, in order to make it usable by a neural network. In fact, it was necessary to define a new data structure, so that each record, relating to a given hour, would contain for each station information relating to the concentration of *nitrogen dioxide*, temperature, humidity, precipitations, wind speed and wind direction plus the artificial features about time information; this information could in fact determine the dispersion speed of atmospheric pollutants, enabling better forecasts to be obtained. For the association of meteorological information, the meteorological station *closest* to the air quality monitoring station under consideration was chosen; any missing data were also completed by means of a *k-Nearest Neighbors algorithm*. *Sub-sequences* were then extracted from this data and each was assigned the *label* to be predicted.

By manipulating the data in this way, it was possible to feed it into a *neural network*. By carrying out various experiments, it was concluded that the most suitable model for this type of problem is a *multilevel recurrent neural network*. It has been shown that such a network under certain conditions can generate good predictions, this therefore justifies the adoption of these ML techniques, as they are indeed able to obtain appreciable results. With the definition of the final model and the analysis of its predictions, the intended goal is deemed to have been achieved.

In spite of this, some problems arose during the experiments, which revealed the main weakness of these techniques. In fact, the model is trained on the *training* set, it is validated at the end of each epoch on the *validation* set, to understand whether the model has actually improved compared to the previous epoch, and finally verified on the *test* set, to assess its final performance. This means that the final result of the training phase and the verdict on the performance evaluation depend, almost exclusively, on the characteristics of the three sets. If the data have any kind of anomaly, as is probably the case here, and consequently if the three sets do not have homogeneous characteristics, questionable results may be obtained.

However, although the data were affected by some anomalies, it was possible to obtain acceptable predictions. In fact, the objective was not to identify a method or configuration to obtain very good predictions, but rather to identify the main *machine learning* techniques and apply them to a real problem in order to obtain respectable but certainly not perfect results.

It must be kept in mind that in order to obtain an adequate quality of results, it would be necessary to increase the complexity of the model in relation to the number of stations considered; this may make a different approach preferable, defining, for example, several simpler models, and assigning to each one some subset of the stations to be considered. In general, the greatest effort required in applying these techniques consists in the initial analysis and manipulation of the data, selecting the most important features and defining the new data structure to be provided to the network.

In this paper, the main notions relating to *neural networks* and the main technologies available have been described, then put into practice on a real case in a simple way, in order to achieve the objectives set, without going into all the possible insights or experiments that can be carried out. During the steps necessary to achieve the objectives, certain strategies were adopted and certain choices were made, sometimes based on theoretical knowledge, sometimes on the results of experimental tests, and sometimes on intuition. It cannot therefore be guaranteed that the solutions proposed in this discussion were always the optimal choices.

The repercussion of varying these conditions on the results could be the subject of further study and experimentation. In any case, the results obtained, considering that the starting point was a simple collection of data, are certainly useful and fully reflect the potential of these techniques.

References

[1] Air quality Open Data Arpae URL: *https://dati.arpae.it/dataset/qualita-dell-aria-rete-di-monitoraggio* (visited on 10/04/2023).

[2] Dext3r Arpae - weather open data portal URL: *https://simc.arpae.it/dext3r/* (visited on 15/04/2023).

[3] "Analisi dati inquinamento atmosferico mediante machine learning" - Lorenzo Mondani URL: *https://amslaurea.unibo.it/id/eprint/16168* (visited on 27/03/2023).

[4] Consolidated text: "Directive 2008/50/EC of the European Parliament and of the Council of 21 May 2008 on ambient air quality and cleaner air for Europe" URL: *https://eur-lex.europa.eu/eli/dir/2008/50/2015-09-18* (visited on 27/03/2023).

[5] D.M. 30 marzo 2017 - "Procedure di garanzia di qualita' per verificare il rispetto della qualita' delle misure dell'aria ambiente, effettuate nelle stazioni delle reti di misura." (17A02825) URL: *https://www.gazzettaufficiale.it/eli/id/2017/04/26/17A02825/sg* (visited on 27/03/2023).

[6] Air Quality Network Arpae URL: *https://www.arpae.it/it/temi-ambientali/aria/dati-qualita-aria/rete-monitoraggio* (visited on 05/04/2023).

[7] Air quality website Arpae URL: *https://www.arpae.it/it/temi-ambientali/aria/dati-qualita-aria/stazioni-fisse* (visited on 10/04/2023).

[8] Nitrogen Dioxide legal limits - Arpae URL: *https://www.arpae.it/it/temi-ambientali/aria/scopri-di-piu/inquinanti-e-iqa/biossido-di-azoto* (visited on 10/04/2023).

[9] Scikit-learn KNN imputer documentation URL: *https://scikit-learn.org/stable/modules/generated/sklearn.impute.KNNImputer.html* (visited on 01/05/2023).

[10] Air quality data capture criteria for validation URL: *http://www.arpamoliseairquality.it/valutazione-della-qualita-dellaria/* (visited on 06/05/2023).

[11] Reducing loss: gradient descent URL: *https://developers.google.com/machine-learning/crash-course/reducing-loss/gradient-descent?hl=en* (visited on 06/05/2023).

[12] Reducing loss: learning rate URL: *https://developers.google.com/machine-learning/crash-course/reducing-loss/learning-rate?hl=en* (visited on 06/05/2023).

[13] Reducing loss: stochastic gradient descent URL: *https://developers.google.com/machine-learning/crash-course/reducing-loss/stochastic-gradient-descent?hl=en* (visited on 06/05/2023).

[14] Generalization: Peril of Overfitting URL: *https://developers.google.com/machine-learning/crash-course/generalization/peril-of-overfitting* (visited on 07/05/2023).

[15] Training and Test Sets: Splitting Data URL: *https://developers.google.com/machine-learning/crash-course/training-and-test-sets/splitting-data?hl=en* (visited on 07/05/2023).

[16] Validation Set: Another Partition URL: *https://developers.google.com/machine-learning/crash-course/validation/another-partition?hl=en* (visited on 07/05/2023).

[17] Representation: Feature Engineering URL: *https://developers.google.com/machine-learning/crash-course/representation/feature-engineering?hl=enn* (visited on 08/05/2023).

[18] Representation: Qualities of Good Features URL: *https://developers.google.com/machine-learning/crash-course/representation/qualities-of-good-features?hl=en* (visited on 08/05/2023).

[19] Representation: Cleaning Data URL: *https://developers.google.com/machine-learning/crash-course/representation/cleaning-data?hl=en* (visited on 08/05/2023).

[20] Regularization for Simplicity: Lambda URL: *https://developers.google.com/machine-learning/crash-course/regularization-for-simplicity/lambda?hl=en* (visited on 08/05/2023).

[21] Regularization for Simplicity: L_1 Regularization URL: *https://developers.google.com/machine-learning/crash-course/regularization-for-sparsity/l1-regularization?hl=en* (visited on 08/05/2023).

[22] Regularization for Simplicity: L_2 Regularization URL: *https://developers.google.com/machine-learning/crash-course/regularization-for-simplicity/l2-regularization?hl=en* (visited on 08/05/2023).

[23] Neural Networks: Structure URL: *https://developers.google.com/machine-learning/crash-course/introduction-to-neural-networks/anatomy?hl=en* (visited on 09/05/2023).

[24] Feature Crosses: Encoding Nonlinearity URL: *https://developers.google.com/machine-learning/crash-course/feature-crosses/encoding-nonlinearity?hl=en* (visited on 09/05/2023).