**ELECTRICITY PRICES PREDICTION**

The price of electricity on the European market is very volatile. This is due both to its mode of production by different sources, each with its own constraints (volume of production, dependence on the weather, or production inertia), and by the difficulty of its storage. Being able to predict the prices of the next day is an important issue, to allow the development of intelligent uses of electricity. In this article, we investigate the capabilities of different machine learning techniques to accurately predict electricity prices. Specifically, we extend current state-of-the-art approaches by considering previously unused predictive features such as price histories of neighboring countries. We show that these features significantly improve the quality of forecasts, even in the current period when sudden changes are occurring. We also develop an analysis of the contribution of the different features in model prediction using Shap values, in order to shed light on how models make their prediction and to build user confidence in models.

**Introduction**

The problem of Electricity Price Forecasting (EPF) is becoming more and more challenging to solve. The applications made possible by a price forecasting model are crucial for achieving the energy transition. They allow owners of renewable energy production means to make profit on the market by anticipating price movements and promote smart applications such as self-consumption [1] or car batteries optimization [2].

At the same time, there are numerous factors that need to be taken into account to understand electricity prices. For example, energy transition policies increase the proportion of renewable energy in total production [3] and introduce new market regulations such as taxation of carbon dioxide emissions. Moreover, cross countries interconnections are multiplying and some markets such as the EPEX SPOT1 set prices for all European countries, bringing the forecasting task to the scale of the continent.

Additionally, the pricing algorithms [4] used to balance generation and consumption can lead to price spikes, both negative and positive. These spikes can result in huge losses for unwary business owners and are difficult to handle by traditional forecasting models. Particularly, the current period is marked by repeated lockdowns that cause severe changes in the European market. The economic recovery following the COVID pandemic [5], [6] also causes prices to reach up to five times the usual season price, with an increased volatility, as shown in Fig. 1.

Meanwhile, Machine Learning (ML) models are increasingly effective in solving difficult problems [7], [8] and can represent complex situations [9], [10]. However, they are sometimes hard to reproduce, if the described methodology and parameters are not thoroughly reported. ML models are also known to lack explainability, be difficult to interpret and are often thought of as black box models. Data analysts generally decide whether to use them or not based on a single metric evaluated solely on one dataset.

Overall, the interest of researchers and business owners in EPF is growing [11], [12]. EachSection snippets

Electricity price forecasting

Electricity markets are subject to several constraints induced by the inherent nature of this energy which requires consumption and production to be permanently matched on a continental scale. To tackle this problem, markets use pricing algorithms. For European exchanges, the euphemia [4] algorithm maximizes social welfare by solving a mixed-integer quadratic programming optimization problem. Social welfare is defined as the sum of consumer surplus, supplier surplus and cross-border trade

**Machine learning for EPF**

Machine Learning (ML) is a branch of computer science proposing forecasting models by implementing efficient learning from data algorithms. This field has received a lot of attention in the past decades due to the abundance of available data and the growing computing power of machines. In the field of forecasting, ML models have been able to solve very complex problems in image processing [7], [59], [60] but also in multivariate Time Series regression [8], [13], [14], [61], [62], [63]. As we

**Datasets**

Many multivariate time series forecasting research articles [13], [14] recommend to evaluate models on several datasets as the behavior of a same algorithm can be very different depending on unknown characteristics of the dataset. The relative performances of several models can even vary and considering a large number of datasets makes it possible to have a more robust evaluation of the model performances. To assess the specific qualities of a model, it is therefore relevant to consider

**Evaluation of the models on the different datasets**

The objective of this section is to evaluate the different models of machine learning. First, we measure the impact of considering the additional features on the accuracy of predictions. We also evaluate the interest of simultaneously predicting the price of electricity in several countries. Then, we propose to study the models from an XAI point of view, to identify on which variables the predictions are based.

**Synthesis, discussion and future work**

In this section, we summarize our conclusions and observations from the results of our experiments. First, we see that including new features in the predictive dataset dramatically increases model performance. Among these added features, the most discriminating are the features without lag days: production and consumption forecasts, and Swiss prices. We believe that the Belgian dataset is more difficult to grasp as it lacks the forecasts for the period

and the Swiss prices for the period

.

**Credit authorship contribution statement**

Léonard Tschora: Conceptualization, Methodology, Implementation, Validation, Investigation of the results, Writing – original draft. Erwan Pierre: Investigation, Resources, Data curation, Writing, Funding acquisition. Marc Plantevit: Conceptualization, Investigation, Supervision, Writing, Funding acquisition. Céline Robardet: Conceptualization, Investigation, Supervision, Writing, Funding acquisition.

**Declaration of Competing Interest**

Leonard Tschora reports financial support was provided by National Association for Research and Technology.

**Acknowledgment**

This research has received funding from the ANRT (French National Association for Research and Technology) .

References (86)

AmabileL. et al.

Optimizing the self-consumption of residential photovoltaic energy and quantification of the impact of production forecast uncertainties

Adv Appl Energy

(2021)

MeiJ. et al.

Stochastic optimization of multi-energy system operation considering hydrogen-based vehicle applications

Adv Appl Energy

(2021)

SuvarnaM. et al.

A machine learning framework to quantify and assess the impact of COVID-19 on the power sector: An Indian context

Adv Appl Energy

(2022)

VlahogianniE.I. et al.

Short-term traffic forecasting: Where we are and where we’re going

Transp Res C

(2014)

LagoJ. et al.

Forecasting day-ahead electricity prices: A review of state-of-the-art algorithms, best practices and an open-access benchmark

Appl Energy

(2021)

PinoR. et al.

Forecasting next-day price of electricity in the spanish energy market using artificial neural networks

Eng Appl Artif Intell

(2008)

ShiW. et al.

An effective two-stage electricity price forecasting scheme

Electr Power Syst Res

(2021)

WangD. et al.

Multi-step ahead electricity price forecasting using a hybrid model based on two-layer decomposition technique and BP neural network optimized by firefly algorithm

Appl Energy

(2017)

ZielF. et al.

Forecasting day ahead electricity spot prices: The impact of the exaa to other European electricity markets

Energy Econ

(2015)

TanZ. et al.

Day-ahead electricity price forecasting using wavelet https://www.kaggle.com/code/dimitriosroussis/electricity-price-forecasting-with-dnns-eda?scriptVersionId=57748754&cellId=9(2010)

View more references

Cited by (45)

Harnessing eXplainable artificial intelligence for feature selection in time series energy forecasting: A comparative analysis of Grad-CAM and SHAP

2024, Applied Energy

Show abstract

Forecasting electricity prices from the state-of-the-art modeling technology and the price determinant perspectives

2024, Research in International Business and Finance

Show abstract

Event-driven forecasting of wholesale electricity price and frequency regulation price using machine learning algorithms

2023, Applied Energy

Show abstract

Profit Maximization of Retailers with Intermittent Renewable Sources and Energy Storage Systems in Deregulated Electricity Market with Modern Optimization TechniquesWe will drop all the columns that are constituted by zeroes and NaNs, as they are unusable. We will also remove the columns which will not be used at all in our analysis and which contain day-ahead forecasts for the total load, the solar energy and the wind energy.

# Drop unusable columns

df\_energy = df\_energy.drop(['generation fossil coal-derived gas','generation fossil oil shale',

'generation fossil peat', 'generation geothermal',

'generation hydro pumped storage aggregated', 'generation marine',

'generation wind offshore', 'forecast wind offshore eday ahead',

'total load forecast', 'forecast solar day ahead',

'forecast wind onshore day ahead'],

axis=1)

df\_energy.describe().round(2)

generation biomass generation fossil brown coal/lignite generation fossil gas generation fossil hard coal generation fossil oil generation hydro pumped storage consumption generation hydro run-of-river and poundage generation hydro water reservoir generation nuclear generation other generation other renewable generation solar generation waste generation wind onshore total load actual price day ahead price actual

count 35045.00 35046.00 35046.00 35046.00 35045.00 35045.00 35045.00 35046.00 35047.00 35046.00 35046.00 35046.00 35045.00 35046.00 35028.00 35064.00 35064.00

mean 383.51 448.06 5622.74 4256.07 298.32 475.58 972.12 2605.11 6263.91 60.23 85.64 1432.67 269.45 5464.48 28696.94 49.87 57.88

std 85.35 354.57 2201.83 1961.60 52.52 792.41 400.78 1835.20 839.67 20.24 14.08 1680.12 50.20 3213.69 4574.99 14.62 14.20

min 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 18041.00 2.06 9.33

25% 333.00 0.00 4126.00 2527.00 263.00 0.00 637.00 1077.25 5760.00 53.00 73.00 71.00 240.00 2933.00 24807.75 41.49 49.35

50% 367.00 509.00 4969.00 4474.00 300.00 68.00 906.00 2164.00 6566.00 57.00 88.00 616.00 279.00 4849.00 28901.00 50.52 58.02

75% 433.00 757.00 6429.00 5838.75 330.00 616.00 1250.00 3757.00 7025.00 80.00 97.00 2578.00 310.00 7398.00 32192.00 60.53 68.01

max 592.00 999.00 20034.00 8359.00 449.00 4523.00 2000.00 9728.00 7117.00 106.00 119.00 5792.00 357.00 17436.00 41015.00 101.99 116.80

df\_energy.info()

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 35064 entries, 0 to 35063

Data columns (total 18 columns):

time 35064 non-null object

generation biomass 35045 non-null float64

generation fossil brown coal/lignite 35046 non-null float64

generation fossil gas 35046 non-null float64

generation fossil hard coal 35046 non-null float64

generation fossil oil 35045 non-null float64

generation hydro pumped storage consumption 35045 non-null float64

generation hydro run-of-river and poundage 35045 non-null float64

generation hydro water reservoir 35046 non-null float64

generation nuclear 35047 non-null float64

generation other 35046 non-null float64

generation other renewable 35046 non-null float64

generation solar 35046 non-null float64

generation waste 35045 non-null float64

generation wind onshore 35046 non-null float64

total load actual 35028 non-null float64

price day ahead 35064 non-null float64

price actual 35064 non-null float64

dtypes: float64(17), object(1)

memory usage: 4.8+ MB

The 'time' column, which we also want to function as the index of the observations in a time-series, has not been parsed correctly and is recognized as an object.

# Convert time to datetime object and set it as index

df\_energy['time'] = pd.to\_datetime(df\_energy['time'], utc=True, infer\_datetime\_format=True)

df\_energy = df\_energy.set\_index('time')

# Find NaNs and duplicates in df\_energy

print('There are {} missing values or NaNs in df\_energy.'

.format(df\_energy.isnull().values.sum()))

temp\_energy = df\_energy.duplicated(keep='first').sum()

print('There are {} duplicate rows in df\_energy based on all columns.'

.format(temp\_energy))

There are 292 missing values or NaNs in df\_energy.

There are 0 duplicate rows in df\_energy based on all columns.

As we can see, df\_energy has no duplicate values. Nevertheless, it has some NaNs and thus, we have to investigate further. Since this is a time-series forecasting task, we cannot simply drop the rows with the missing values and it would be a better idea to fill the missing values using interpolation.

# Find the number of NaNs in each column

df\_energy.isnull().sum(axis=0)

generation biomass 19

generation fossil brown coal/lignite 18

generation fossil gas 18

generation fossil hard coal 18

generation fossil oil 19

generation hydro pumped storage consumption 19

generation hydro run-of-river and poundage 19

generation hydro water reservoir 18

generation nuclear 17

generation other 18

generation other renewable 18

generation solar 18

generation waste 19

generation wind onshore 18

total load actual 36

price day ahead 0

price actual 0

dtype: int64

Most null values can be found in the 'total load actual' column. Therefore, it is a good idea to visualize it and see what we can do. The good news is that there are no NaNs in the 'price actual' column, which we will use as the target variable in order to train our model. The similar numbers in null values in the columns which have to do with the type of energy generation probably indicate that they will also appear in the same rows. Let us first define a plot function which we will then use so as to visualize the 'total load actual' column, as well as other columns.

# Define a function to plot different types of time-series

def plot\_series(df=None, column=None, series=pd.Series([]),

label=None, ylabel=None, title=None, start=0, end=None):

"""

Plots a certain time-series which has either been loaded in a dataframe

and which constitutes one of its columns or it a custom pandas series

created by the user. The user can define either the 'df' and the 'column'

or the 'series' and additionally, can also define the 'label', the

'ylabel', the 'title', the 'start' and the 'end' of the plot.

"""

sns.set()

fig, ax = plt.subplots(figsize=(30, 12))

ax.set\_xlabel('Time', fontsize=16)

if column:

ax.plot(df[column][start:end], label=label)

ax.set\_ylabel(ylabel, fontsize=16)

if series.any():

ax.plot(series, label=label)

ax.set\_ylabel(ylabel, fontsize=16)

if label:

ax.legend(fontsize=16)

if title:

ax.set\_title(title, fontsize=24)

ax.grid(True)

return ax

# Zoom into the plot of the hourly (actual) total load

ax = plot\_series(df=df\_energy, column='total load actual', ylabel='Total Load (MWh)',Zoom into the plot of the hourly (actual) total load

ax = plot\_series(df=df\_energy, column='total load actual', ylabel='Total Load (MWh)',

title='Actual Total Load (First 2 weeks - Original)', end=24\*7\*2)

plt.show()

After zooming into the first 2 weeks of the 'total load actual' column, we can already see that there are null values for a few hours. However, the number of the missing values and the behavior of the series indicate that an interpolation would fill the NaNs quite well. Let us further investigate if the null values coincide across the different columns. Let us display the last five rows.

# Display the rows with null values

df\_energy[df\_energy.isnull().any(axis=1)].tail()

generation biomass generation fossil brown coal/lignite generation fossil gas generation fossil hard coal generation fossil oil generation hydro pumped storage consumption generation hydro run-of-river and poundage generation hydro water reservoir generation nuclear generation other generation other renewable generation solar generation waste generation wind onshore total load actual price day ahead price actual

time

2016-11-23 03:00:00+00:00 NaN 900.0 4838.0 4547.0 269.0 1413.0 795.0 435.0 5040.0 60.0 85.0 15.0 227.0 4598.0 23112.0 43.19 49.11

2017-11-14 11:00:00+00:00 0.0 0.0 10064.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 NaN 60.53 66.17

2017-11-14 18:00:00+00:00 0.0 0.0 12336.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 NaN 68.05 75.45

2018-06-11 16:00:00+00:00 331.0 506.0 7538.0 5360.0 300.0 1.0 1134.0 4258.0 5856.0 52.0 96.0 170.0 269.0 9165.0 NaN 69.87 64.93

2018-07-11 07:00:00+00:00 NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN 63.01 69.79

If we manually searched through all of them, we would confirm that the null values in the columns which have to do with the type of energy generation mostly coincide. The null values in 'actual total load' also coincide with the aforementioned columns, but also appear in other rows as well. In order to handle the null values in df\_energy, we will use a linear interpolation with a forward direction. Perhaps other kinds of interpolation would be better; nevertheless, we prefer to use the simplest model possible. Only a small part of our input data will be noisy and it will not affect performance noticeably.

# Fill null values using interpolation

df\_energy.interpolate(method='linear', limit\_direction='forward', inplace=True, axis=0)

# Display the number of non-zero values in each column

print('Non-zero values in each column:\n', df\_energy.astype(bool).sum(axis=0), sep='\n')

Non-zero values in each column:

generation biomass 35060

generation fossil brown coal/lignite 24540

generation fossil gas 35063

generation fossil hard coal 35061

generation fossil oil 35061

generation hydro pumped storage consumption 22450

generation hydro run-of-river and poundage 35061

generation hydro water reservoir 35061

generation nuclear 35061

generation other 35060

generation other renewable 35061

generation solar 35061

generation waste 35061

generation wind onshore 35061

total load actual 35064

price day ahead 35064

price actual 35064

dtype: int64

It look like df\_energy has been cleaned successfully and is ready for further use as input into our model. The 1-4 zeroes in the columns which have to do with energy generation by type should not concern us very much. The 'generation hydro pumped storage consumption' may look suspicious, but we should have in mind that this type of energy is only used for load balancing, being consumed when in peak energy demands.

1.2. Weather features dataset

df\_weather.head()

dt\_iso city\_name temp temp\_min temp\_max pressure humidity wind\_speed wind\_deg rain\_1h rain\_3h snow\_3h clouds\_all weather\_id weather\_main weather\_description weather\_icon

0 2015-01-01 00:00:00+01:00 Valencia 270.475 270.475 270.475 1001 77 1 62 0.0 0.0 0.0 0 800 clear sky is clear 01n

1 2015-01-01 01:00:00+01:00 Valencia 270.475 270.475 270.475 1001 77 1 62 0.0 0.0 0.0 0 800 clear sky is clear 01n

2 2015-01-01 02:00:00+01:00 Valencia 269.686 269.686 269.686 1002 78 0 23 0.0 0.0 0.0 0 800 clear sky is clear 01n

3 2015-01-01 03:00:00+01:00 Valencia 269.686 269.686 269.686 1002 78 0 23 0.0 0.0 0.0 0 800 clear sky is clear 01n

4 2015-01-01 04:00:00+01:00 Valencia 269.686 269.686 269.686 1002 78 0 23 0.0 0.0 0.0 0 800 clear sky is clear 01n

df\_weather.describe().round(2)

temp temp\_min temp\_max pressure humidity wind\_speed wind\_deg rain\_1h rain\_3h snow\_3h clouds\_all weather\_id

count 178396.00 178396.00 178396.00 178396.00 178396.00 178396.00 178396.00 178396.00 178396.00 178396.00 178396.00 178396.00

mean 289.62 288.33 291.09 1069.26 68.42 2.47 166.59 0.08 0.00 0.00 25.07 759.83

std 8.03 7.96 8.61 5969.63 21.90 2.10 116.61 0.40 0.01 0.22 30.77 108.73

min 262.24 262.24 262.24 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 200.00

25% 283.67 282.48 284.65 1013.00 53.00 1.00 55.00 0.00 0.00 0.00 0.00 800.00

50% 289.15 288.15 290.15 1018.00 72.00 2.00 177.00 0.00 0.00 0.00 20.00 800.00

75% 295.15 293.73 297.15 1022.00 87.00 4.00 270.00 0.00 0.00 0.00 40.00 801.00

max 315.60 315.15 321.15 1008371.00 100.00 133.00 360.00 12.00 2.32 21.50 100.00 804.00

Here, we can see that all columns of df\_weather have the same number of rows; we still have to check what is the case for each city individually, though. We should note that the temperatures are in Kelvin. The most important thing to notice is that there are some problems and outliers. In particular:

There is at least one outlier in the 'pressure' column as the maximum value is 1,008,371 hPa or approximately 100 MPa, which is roughly the pressure at the bottom of Mariana Trench about 11 km below ocean surface [2]. This cannot be the case here.

There is at least one outlier in the 'wind\_speed' column as the maximum value is 133 m/s. This measurement is close to the fastest wind speed ever recorded on Earth, caused by the 1999 Bridge Creek–Moore tornado [3], a F5 (the largest intensity of the Fujita scale) tornado [4]. A tornado of such intensity has not been recorded in Spain [5] and hopefully will not happen in the future as well.

The 'rain\_3h' column is supposed to provide information about the precipitation (i.e. rain) of the last 3 hours in mm. Since the 'rain\_1h' column is supposed to provide the same information but about just the last hour, it would be logical to assume that its mean would be less than that of 'rain\_3h'. However, this is not the case in the statistical description above. So, it would be a good idea to further examine those columns.

# Print the type of each variable in df\_weather

df\_weather.info()

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 178396 entries, 0 to 178395

Data columns (total 17 columns):

dt\_iso 178396 non-null object

city\_name 178396 non-null object

temp 178396 non-null float64

temp\_min 178396 non-null float64

temp\_max 178396 non-null float64

pressure 178396 non-null int64

humidity 178396 non-null int64

wind\_speed 178396 non-null int64

wind\_deg 178396 non-null int64

rain\_1h 178396 non-null float64

rain\_3h 178396 non-null float64

snow\_3h 178396 non-null float64

clouds\_all 178396 non-null int64

weather\_id 178396 non-null int64

weather\_main 178396 non-null object

weather\_description 178396 non-null object

weather\_icon 178396 non-null object

dtypes: float64(6), int64(6), object(5)

memory usage: 23.1+ MB

We have to change the type of some of the columns, so that all of them are float64. We also have to parse 'dt\_iso' correctly and actually rename it as 'time' so that it matches with the index of df\_energy.

def df\_convert\_dtypes(df, convert\_from, convert\_to):

cols = df.select\_dtypes(include=[convert\_from]).columns

for col in cols:

df[col] = df[col].values.astype(convert\_to)

return df

# Convert columns with int64 type values to float64 type

df\_weather = df\_convert\_dtypes(df\_weather, np.int64, np.float64)

# Convert dt\_iso to datetime type, rename it and set it as index

df\_weather['time'] = pd.to\_datetime(df\_weather['dt\_iso'], utc=True, infer\_datetime\_format=True)

df\_weather = df\_weather.drop(['dt\_iso'], axis=1)

df\_weather = df\_weather.set\_index('time')

We have to split the df\_weather dataset into 5 datasets, one for each different city (Madrid, Barcelona, Bilbao, Seville and Valencia). But first,Display average weather features grouped by each city

mean\_weather\_by\_city = df\_weather.groupby('city\_name').mean()

mean\_weather\_by\_city

temp temp\_min temp\_max pressure humidity wind\_speed wind\_deg rain\_1h rain\_3h snow\_3h clouds\_all weather\_id

city\_name

Barcelona 289.848248 288.594704 291.021987 1284.010486 73.994221 2.786588 187.188043 0.117079 0.000327 0.000000 23.229648 760.917465

Bilbao 286.378489 284.916661 288.036687 1017.567439 79.089455 1.957470 159.883536 0.123493 0.001034 0.023455 43.960697 723.943228

Madrid 288.061071 286.824877 289.155600 1011.838448 59.776932 2.441696 173.293159 0.055083 0.000129 0.000029 22.397028 762.260264

Seville 293.105431 291.184103 295.962431 1018.504711 64.140732 2.483787 151.757179 0.045392 0.000180 0.000000 14.748770 771.409849

Valencia 290.780780 290.222277 291.355025 1015.973794 65.145113 2.692815 160.753820 0.035924 0.000226 0.000154 20.820999 781.228283

# Find NaNs and duplicates in df\_weather

print('There are {} missing values or NaNs in df\_weather.'

.format(df\_weather.isnull().values.sum()))

temp\_weather = df\_weather.duplicated(keep='first').sum()

print('There are {} duplicate rows in df\_weather based on all columns.'

.format(temp\_weather))

There are 0 missing values or NaNs in df\_weather.

There are 8622 duplicate rows in df\_weather based on all columns.

It seems that df\_weather has a lot of duplicate values. However, the method above may also show us rows which have the exame same values. This is not what we are looking for. What we want to ensure, is that there are no duplicate index rows, i.e. that we do not have multiple rows for the exact same hour. Of course, we also have to make sure that these duplicates concern each individual city. Since, df\_weather contains information about 5 different cities, it is very useful to display the number of observations for each one and compare it with the size of df\_energy.

# Display the number of rows in each dataframe

print('There are {} observations in df\_energy.'.format(df\_energy.shape[0]))

cities = df\_weather['city\_name'].unique()

grouped\_weather = df\_weather.groupby('city\_name')

for city in cities:

print('There are {} observations in df\_weather'

.format(grouped\_weather.get\_group('{}'.format(city)).shape[0]),

'about city: {}.'.format(city))

There are 35064 observations in df\_energy.

There are 35145 observations in df\_weather about city: Valencia.

There are 36267 observations in df\_weather about city