Predicting 24-Hour Nationwide Electrical Energy Consumption Based on Regression Techniques

Anatoliy Doroshenko[1,2,†], Dmytro Zhora[1,†], Vladyslav Haidukevych[1,†], Yaroslav Haidukevych[1,†], and Olena Yatsenko[1,*,†]

1 Institute of Software Systems of the National Academy of Sciences of Ukraine, Glushkov ave. 40, build. 5, Kyiv, 03187, Ukraine

2 National Technical University "Ihor Sikorsky Kyiv Polytechnic Institute", Polytechnichna str. 41, build. 18, Kyiv, 03056, Ukraine

Abstract

This paper applies standard regression techniques to forecast the country-wide consumption of electrical energy. All considered machine learning algorithms are available as a part of the Scikit-learn library. Besides the fine-tuning of regression hyperparameters, several data preparation techniques are employed to improve the forecasting accuracy. It is demonstrated that forecasting for 24 hours ahead is possible with good accuracy and has practical significance.

Keywords

Electricity markets, forecasting, machine learning, regression

1. Introduction

For a long time, Ukraine had only one market for electrical energy. That was the market of bilateral agreements that wasn't flexible enough to balance the interests of consumers and suppliers of electricity. Such agreements could span weeks, months, or even years. On July 1st, 2019, Ukraine adopted the European model [1] that assumes the following four markets: bilateral, day-ahead, intraday, and balancing. Despite the electricity market models in Europe having some differences [2], this was also a significant step forward in liberalizing electricity trading between countries.

The bilateral market can be referenced also as a future or forward market. In Ukraine, as shown in Figure 1, the total amount of deals is recorded every hour. At the same time, some European markets allow 15-minute contracts. If we consider four electricity markets in the order they are mentioned above (from bilateral to balancing), the properties of these markets can be formulated as follows:

- the volume of the market decreases,
- the price of the electricity increases,
- the volatility of the volume increases.

The laws of physics apply to electrical circuits regardless of the scale. There are some electricity losses associated with resistance, but usually, they are negligible. If the amount of electrical energy

14th International Scientific and Practical Conference from Programming UkrPROG'2024, May 14-15, 2024, Kyiv, Ukraine

* Corresponding author.

† These authors contributed equally.

doroshenkoanatoliy2@gmail.com (A. Doroshenko); dmitry.zhora@gmx.com (D. Zhora); gaidukevichvlad@gmail.com (V. Haidukevych); yarmcfly@gmail.com (Y. Haidukevych); oayat@ukr.net (O. Yatsenko)

0000-0002-8435-1451 (A. Doroshenko); 0009-0006-6073-7751 (D. Zhora); 0000-0002-0614-6778 (V. Haidukevych); 0000-0002-6300-1778 (Y. Haidukevych); 0000-0002-4700-6704 (O. Yatsenko)

traded and transmitted is measured on substations, we can conclude that the amount of produced electricity is exactly equal to the amount of consumed electricity. That is, for the purpose of this paper we can use the following terms interchangeably: energy production, energy consumption, and market volume. When the country is considered an open system, the following equation applies.

$$production + import = consumption + export. \quad (1)$$

The dataset used in this research represents the time range from July 1st, 2020, to December 31st, 2021. For historical reasons, the time range from July 1st, 2019, to June 30th, 2020, did not contain bilateral market data [3]. The market volume data were provided by the Institute of Energy Modelling, Ukraine. Figure 2 shows the dynamics of all four market components in time.

	A	B	C	D	E	F
1	TradeDate	TradeHour	Bilateral	DayAhead	Intraday	Balancing
2	2020-07-01	0	11746.78	2494	1018.2	-2101
3	2020-07-01	1	11654.98	2697.1	660.3	-2249.5
4	2020-07-01	2	11606.28	2606.8	624	-2398.8
5	2020-07-01	3	11637.28	2507.7	614.7	-2681.5
6	2020-07-01	4	11614.58	2487.2	607.7	-2666.2
7	2020-07-01	5	11645.48	2629.3	605	-2832
8	2020-07-01	6	11696.58	2937	610.6	-2455.8
9	2020-07-01	7	12160.58	3110.5	690.3	-2275.5

Figure 1: Hourly data of electricity market volumes, in megawatt-hours (MWh)

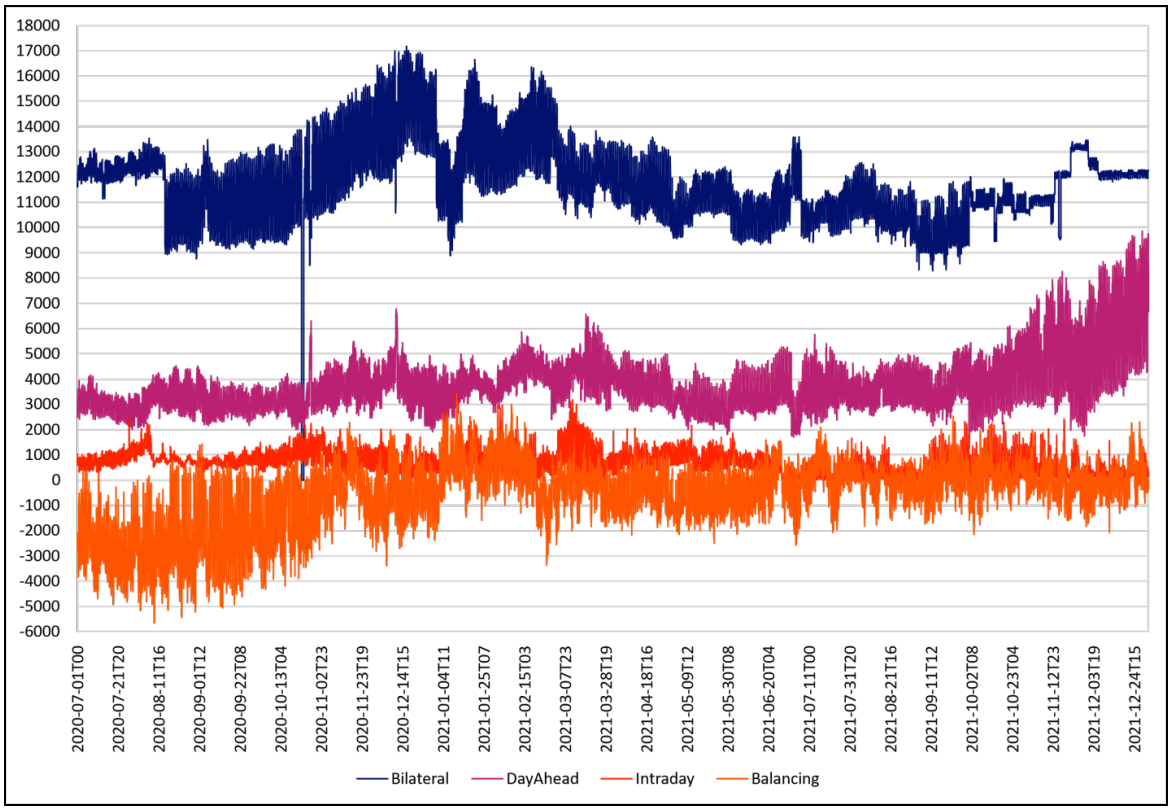

Figure 2: Market volume dependency on time, in megawatt-hours (MWh)

2. Volume Data Augmentation

It is often the case the modeled process is affected by other external factors not represented via input parameters from the original dataset. The outside temperature influences the consumption of electricity as more energy is needed in winter for heating and in summer for air-conditioning. Two columns with hourly data were added to the dataset representing the temperature for Ukraine and its capital, see the dependencies below in Figure 3. The location representing the country was selected as its linear geographic center with decimal GPS coordinates 48.379433N 31.165580E.

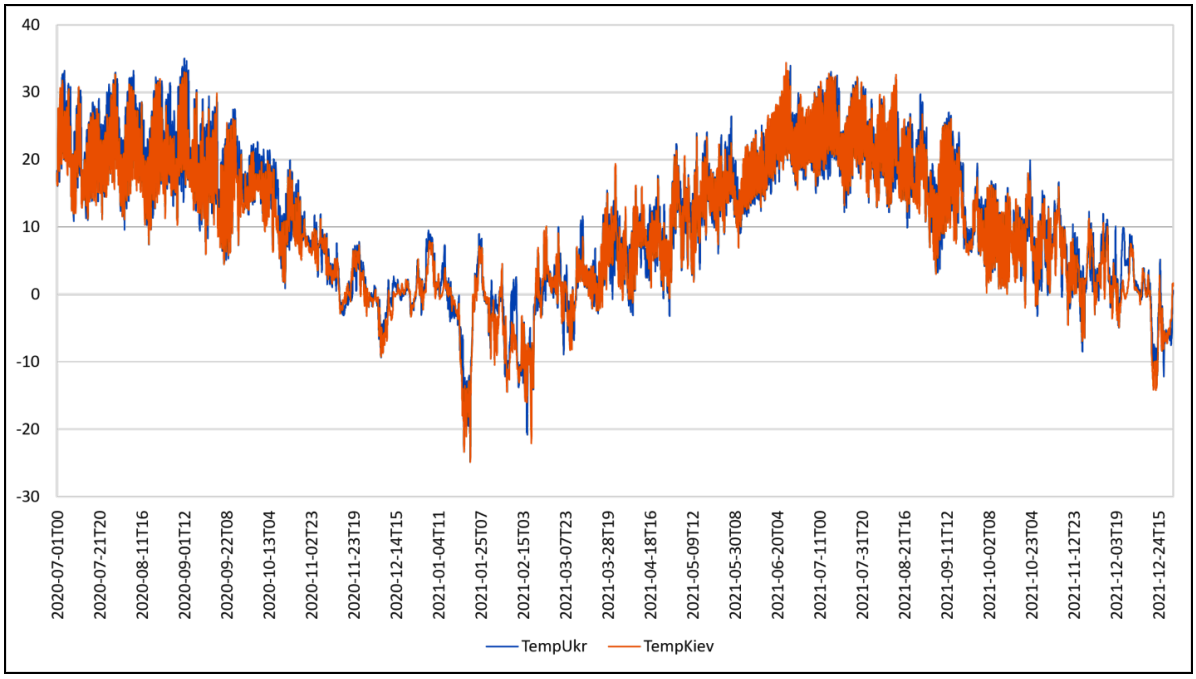

Figure 3: Dependency of outside temperature in Ukraine, hourly representation

Another important factor is the periodicity in the consumption of electrical energy. For example, at night people need less electricity than in the daytime. Similarly, on weekends the electricity consumption is lower than on weekdays. This paper considers four cycle types: daily, weekly, monthly, and yearly. One of the next sections will analyze whether these additions are helpful.

The problem is how to feed time representation to the machine learning algorithm in a way that similar moments in time would be interpreted as close by the algorithm. As shown in Figure 4, hour values 23 and 0 are close on the timescale, but they are distant in real-valued representation. One of the possible solutions to this problem is to calculate the sine and cosine of the cycle phase [4]. Figure 5 demonstrates how every hour in the daily cycle can be represented without gaps. In particular, close values on the timescale are represented by close values of sine and cosine functions.

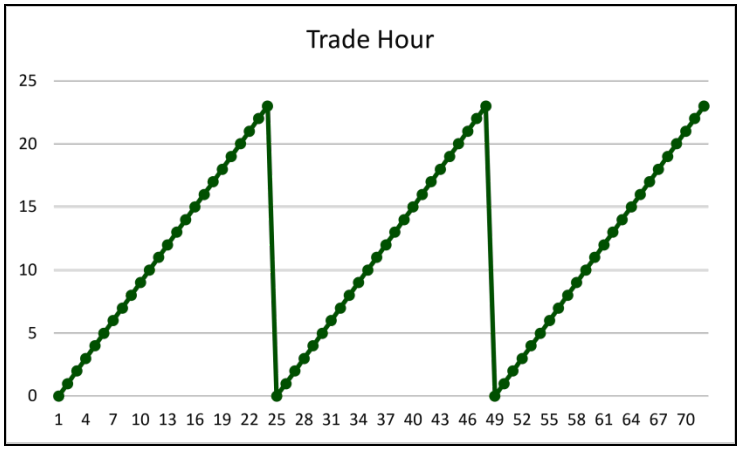

Figure 4: Raw hour data as can be submitted to the machine learning algorithm

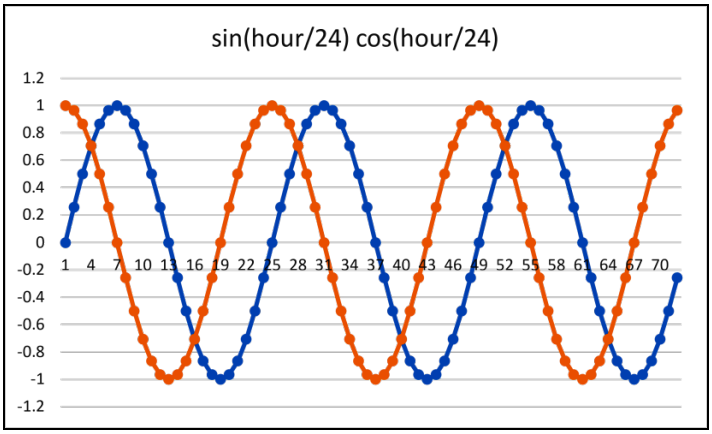

Figure 5: Sine and cosine time series for representation of temporal cycles

The augmented dataset is shown in Figure 6. The first two columns can be interpreted as composite primary key. In addition to the original 4 attribute columns with market volume data now we have 10 more columns. The temperature data were downloaded from the site https://openweathermap.org, the periodic columns were calculated using an algorithm written in Python.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1	TradeDate	TradeHour	Bilateral	DayAhead	Intraday	Balancing	TempUkr	TempKiev	SinDay	CosDay	SinWeek	CosWeek	SinMonth	CosMonth	SinYear	CosYear
2	2020-07-01	0	11746.78	2494	1018.2	-2101	18.38	17.99	0	1	0.9749	-0.2225	0	1	0.0172	-0.9999
3	2020-07-01	1	11654.98	2697.1	660.3	-2249.5	16.72	16.33	0.2588	0.9659	0.9659	-0.2588	0.0084	1	0.0165	-0.9999
4	2020-07-01	2	11606.28	2606.8	624	-2398.8	16.89	16.37	0.5	0.866	0.9556	-0.2948	0.0169	0.9999	0.0157	-0.9999
5	2020-07-01	3	11637.28	2507.7	614.7	-2681.5	16.99	16.52	0.7071	0.7071	0.9439	-0.3303	0.0253	0.9997	0.015	-0.9999
6	2020-07-01	4	11614.58	2487.2	607.7	-2666.2	16.94	16.49	0.866	0.5	0.9309	-0.3653	0.0338	0.9994	0.0143	-0.9999
7	2020-07-01	5	11645.48	2629.3	605	-2832	16.06	16.11	0.9659	0.2588	0.9166	-0.3999	0.0422	0.9991	0.0136	-0.9999
8	2020-07-01	6	11696.58	2937	610.6	-2455.8	16.06	16.3	1	0	0.901	-0.4339	0.0506	0.9987	0.0129	-0.9999
9	2020-07-01	7	12160.58	3110.5	690.3	-2275.5	16.12	16.26	0.9659	-0.2588	0.8841	-0.4673	0.0591	0.9983	0.0122	-0.9999

Figure 6: Augmented market volume dataset with temperature and periodic data

3. Resampling of Temporal Data

The usage of additional input parameters typically provides better regression results. If we need to forecast market volumes for 24 hours ahead then it makes sense to take into account the available data for the last 24 hours (at least). The machine learning algorithms and library functions expect that both input and output parameters are represented as one record. So, as a data preparation step, the data displayed in Figure 6 were resampled into the following columns, where M1 suffix means the parameter was taken one hour ago, P1 suffix means the parameter was taken one hour later, etc.

Primary key: TradeDate, TradeHour

Input columns: SinDay, CosDay, SinWeek, CosWeek, SinMonth, CosMonth, SinYear, CosYear, Bilateral, DayAhead, Intraday, Balancing, TempUkr, TempKiev, BilateralM1, DayAheadM1, IntradayM1, BalancingM1, TempUkrM1, TempKievM1, BilateralM2, DayAheadM2, IntradayM2, BalancingM2, TempUkrM2, TempKievM2, ..., BilateralM23, DayAheadM23, IntradayM23, BalancingM23, TempUkrM23, TempKievM23

Output columns: BilateralP1, DayAheadP1, IntradayP1, BalancingP1, BilateralP2, DayAheadP2, IntradayP2, BalancingP2, ..., BilateralP24, DayAheadP24, IntradayP24, BalancingP24

The obtained dataset had 13'129 records as the first 24 records and the last 24 records after resampling were not fully qualified. The dataset was split into training and testing parts using the standard library function train_test_split from sklearn.model_selection namespace [5]. The obtained datasets were saved into files, so different regression algorithms mentioned further in the paper were evaluated on the same data.

4. Model Evaluation Metrics

To measure the influence of input parameters, we used the nearest neighbors regression model represented by class KNeighborsRegressor from sklearn.neighbors namespace. This machine learning algorithm provides quite competitive results and has a small number of hyperparameters to optimize.

The Python code snippets that implement this functionality are provided in Appendix 1. The complexity of the algorithm is hidden behind fit and prediction methods. Other regression and classification algorithms also reuse these methods, so the substitution of one algorithm instead of another is relatively simple.

The metrics used to measure the discrepancy between the test set and forecasted data are given in Table 1. Here y_i is the output value from the i-th record in the testing dataset, f_i is the predicted value for the i-th record, $\overline{y}$ is the average output value over the test dataset. These formulas are considered in the context of one selected output column representing the market volume.

Table 1
The name and definition of standard metrics for regression task

Metric Name	Metric Formula	Formula Number
R2 score (or determination coefficient)	$R^2 = 1 - \frac{\sum_i (y_i - f_i)^2}{\sum_i (y_i - \overline{y})^2}$	(2)
Mean absolute percentage error	$MAPE = \frac{1}{n}\sum_{i=1}^{n}\left\lvert\frac{y_i - f_i}{y_i}\right\rvert$	(3)
Mean absolute error	$MAE = \frac{1}{n}\sum_{i=1}^{n}\lvert y_i - f_i\rvert$	(4)

5. Manual Feature Selection

Now we need to evaluate the effect of additional parameters and history length on prediction accuracy. Table 2 shows the accuracy improvements after adding temperature and periodic

parameters. It appears all additional parameters are useful, but the overall effect is rather minor. Here are the parameters for the starting model.

Input columns: Bilateral, DayAhead, Intraday, Balancing

Output columns: BilateralP24, DayAheadP24, IntradayP24, BalancingP24

Table 2
The R2 score obtained for different input parameter sets

	Bilateral	DayAhead	Intraday	Balancing
Starting Model	0.93291021	0.90164708	0.72947093	0.77183561
Temperature Data	0.93439629	0.90410349	0.73455079	0.77676317
Daily Cycle	0.93455756	0.90418401	0.73498679	0.77679022
Weekly Cycle	0.93465155	0.90445549	0.73560531	0.77761618
Monthly Cycle	0.93471811	0.90462339	0.73591691	0.77787893
Yearly Cycle	0.93479404	0.90470696	0.73575183	0.77860211

And the following is the intermediate input parameter set obtained.

Input columns: Bilateral, DayAhead, Intraday, Balancing, TempUkr, TempKiev, SinDay, CosDay, SinWeek, CosWeek, SinMonth, CosMonth, SinYear, CosYear

Figure 7 shows the improvements in forecasting results when more historical data is added to the input dataset. The full history for the last 24 hours provides better results. And now the full set of input parameters contains 106 entries that are listed below.

Input columns: Bilateral, DayAhead, Intraday, Balancing, BilateralM1, DayAheadM1, IntradayM1, BalancingM1, ..., BilateralM23, DayAheadM23, IntradayM23, BalancingM23, TempUkr, TempKiev, SinDay, CosDay, SinWeek, CosWeek, SinMonth, CosMonth, SinYear, CosYear

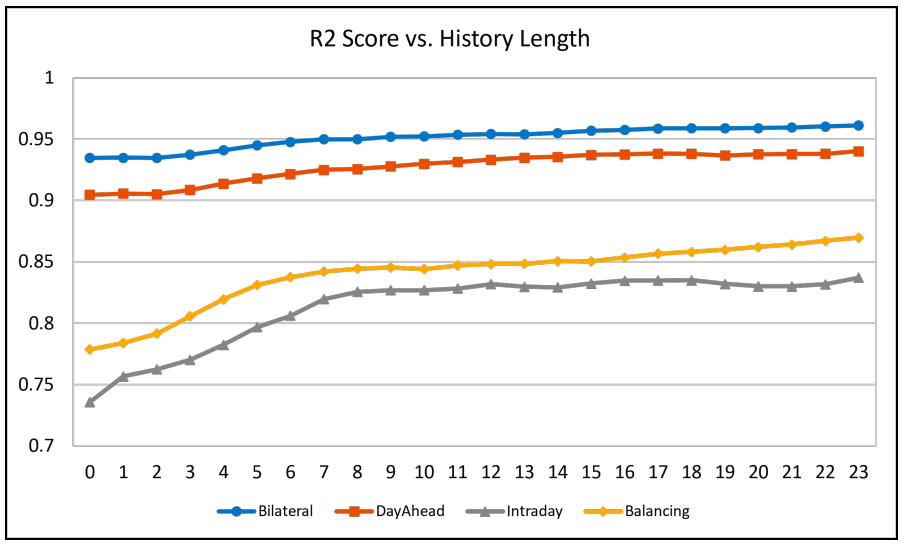

Figure 7: The dependency of the R2 score on the history length in hours

6. Automatic Feature Selection

The high dimensionality of input space is typically considered a problem, especially with noisy data. On the other hand, not all input parameters explored so far have equal contribution to the

quality of results. So, it would be helpful to try removing the parameters that provide less useful information than others.

It appears this is not complex with the class SelectFromModel from sklearn.feature_selection namespace [6]. This meta-transformer should be provided with an estimator object that, in turn, can calculate the array of feature importances. One of such classes is RandomForestRegressor which gets feature importances as a function of informational entropy. The Python code that implements this approach is demonstrated in Appendix 2. The constructor for the SelectFromModel class also takes the threshold parameter that allows to vary the number of features selected. The optimal results were obtained with 60 features taken out of 106, see the results in Table 3 and Appendix 3 for the feature list itself.

Table 3
The R2 score improvements obtained using input feature selection

	Bilateral	DayAhead	Intraday	Balancing
Full Set: 106 Features	0.96129509	0.94019898	0.83718184	0.86971889
60 Selected Features	0.96322701	0.94024491	0.85536199	0.87121345

7. Hourly Forecasting Results

So far, all the results were related to 24-hour forecasting. Figures 8 and 9 below show the R2 score and mean absolute percentage error for the range from 1 and up to 24 hours. The one-hour forecasting provides the best results. It is also worth noting that bilateral and day-ahead markets have much better predictability than balancing markets. As for the intraday market, it has a low mean absolute error just because the size of this market is small.

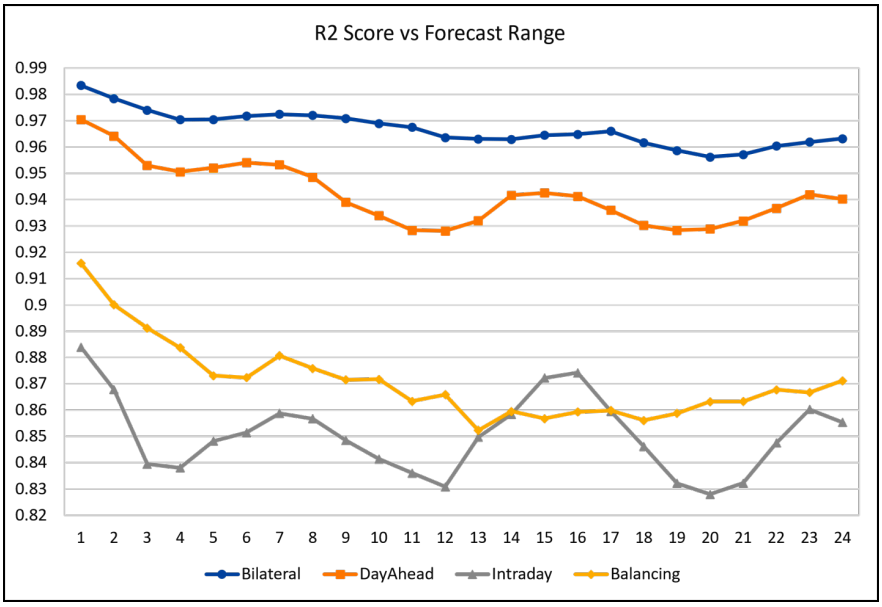

Figure 8: The dependency of the R2 score from the forecast range in hours

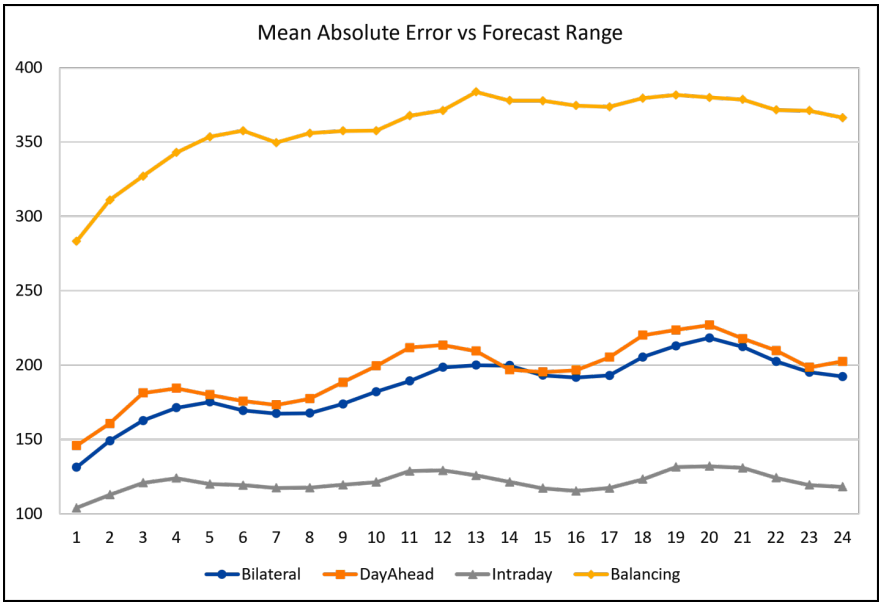

Figure 9: The dependency of mean absolute error from the forecast range (MWh)

8. Forecasting Error Distribution

The 24-hour prediction error for all four markets can be measured on the test set, which represents 20 % of the original dataset. For convenience in representation and analysis, the test set was sorted by real market volume. The predicted values are shown in Figures 10–17 with dots. The probability distribution of error is shown using histograms. An interesting finding is that forecasting error is not always Gaussian. In particular, this is the case for bilateral and intraday market volumes.

The curve representing balancing market volume in Figure 16 crosses the zero line, also it has more negative values than positive. This can be interpreted as that market players tend to overbuy electricity in other markets, so they need to sell more on average at the last moment. Let's note that this inefficiency can be mitigated by the usage of forecasting models.

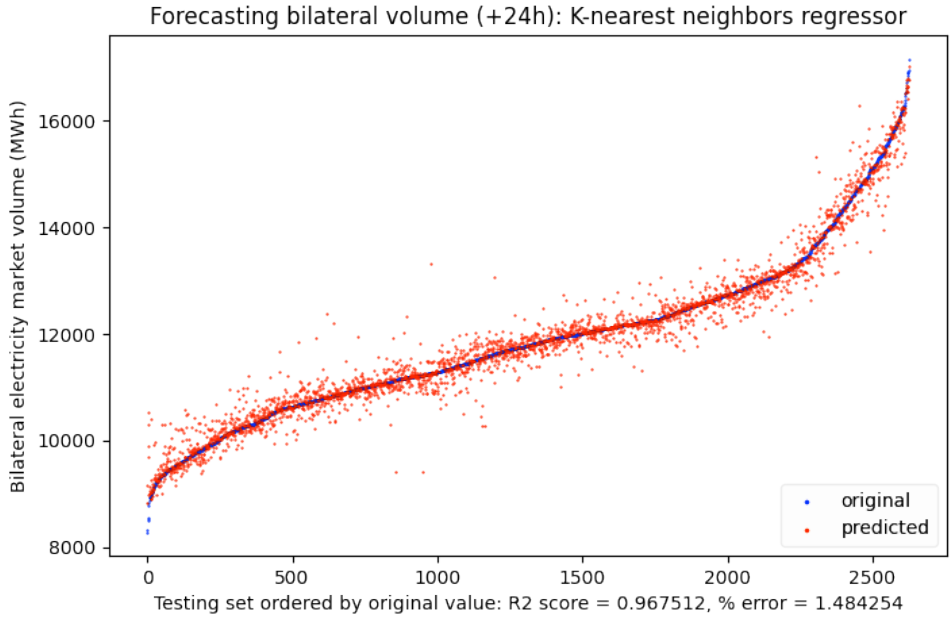

Figure 10: Prediction error for 24 hours ahead, bilateral market volume (MWh)

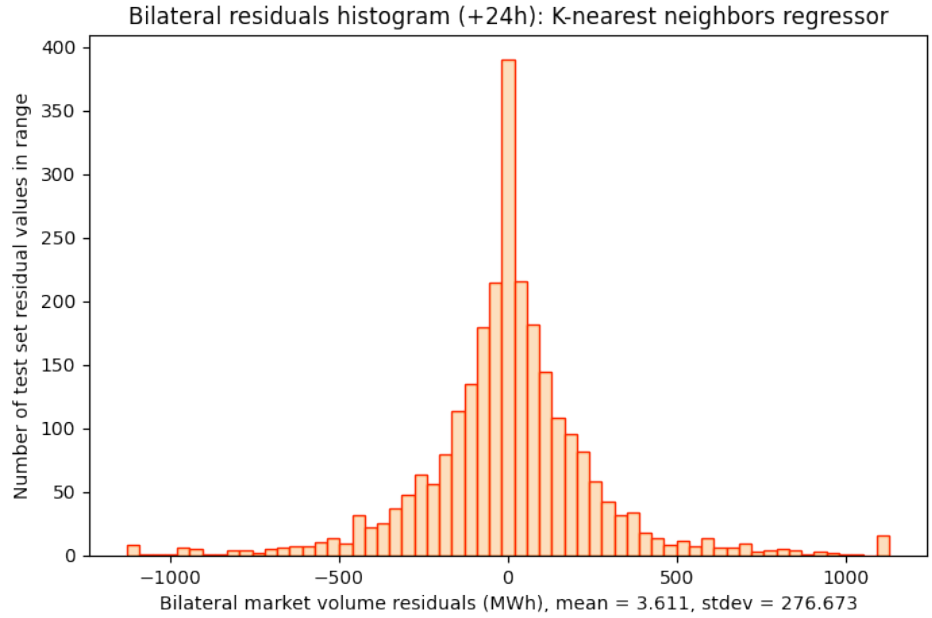

Figure 11: Residuals histogram for 24-hour forecasting, bilateral market volume (MWh)

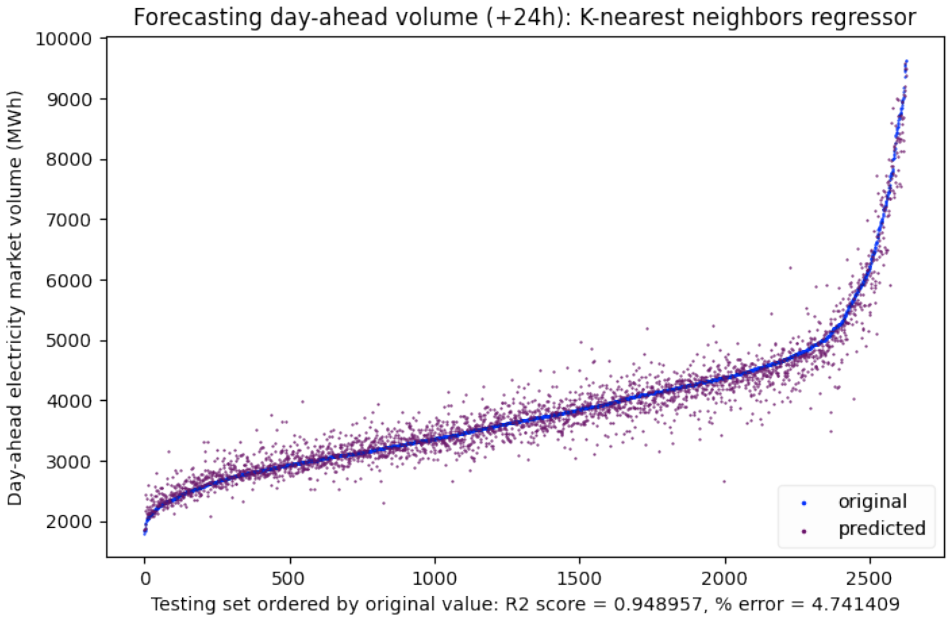

Figure 12: Prediction error for 24-hours ahead, day-ahead market volume (MWh)

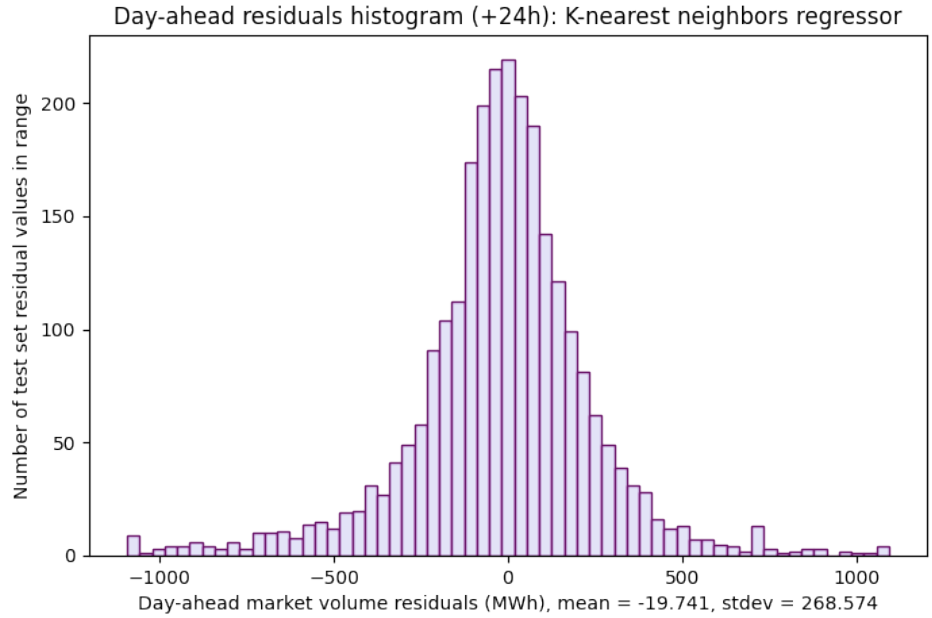

Figure 13: Residuals histogram for 24-hour forecasting, day-ahead market volume (MWh)

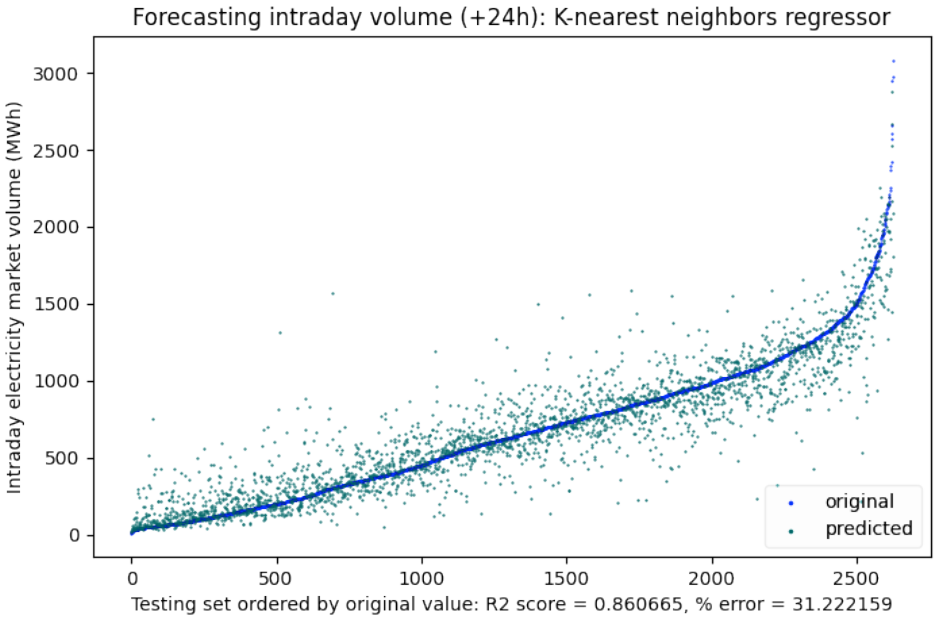

Figure 14: Prediction error for 24 hours ahead, intraday market volume (MWh)

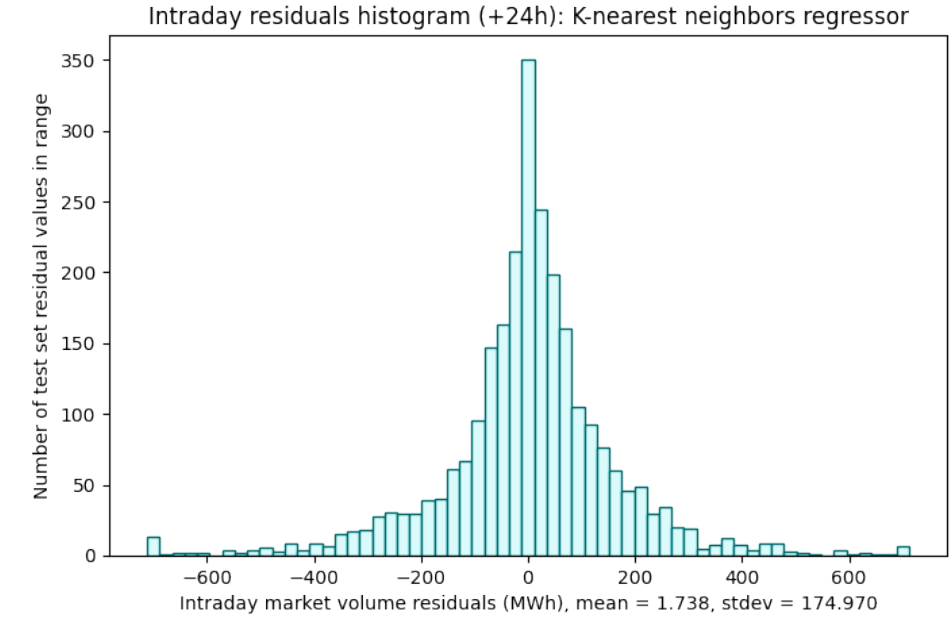

Figure 15: Residuals histogram for 24-hour forecasting, intraday market volume (MWh)

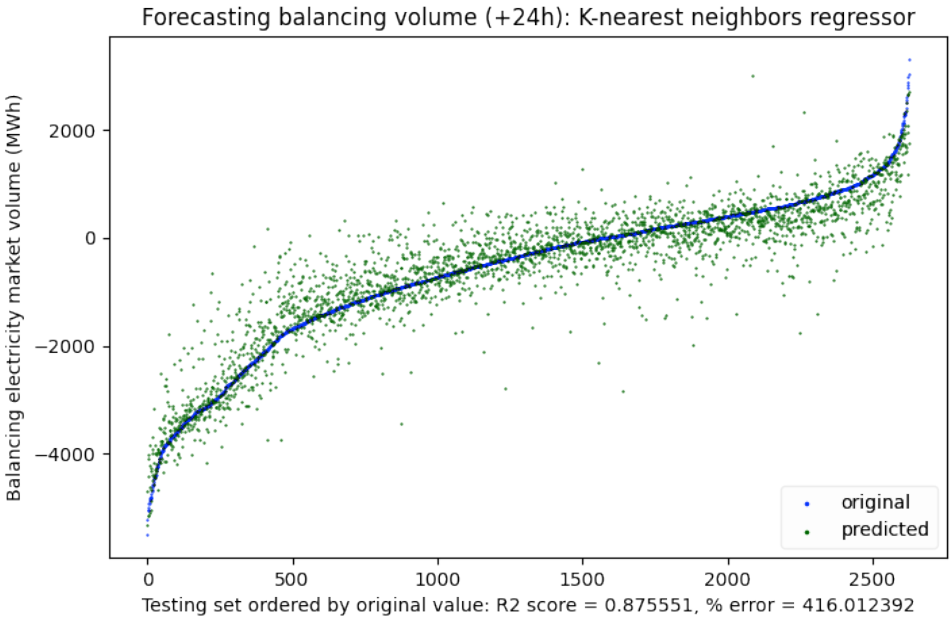

Figure 16: Prediction error for 24 hours ahead, balancing market volume (MWh)

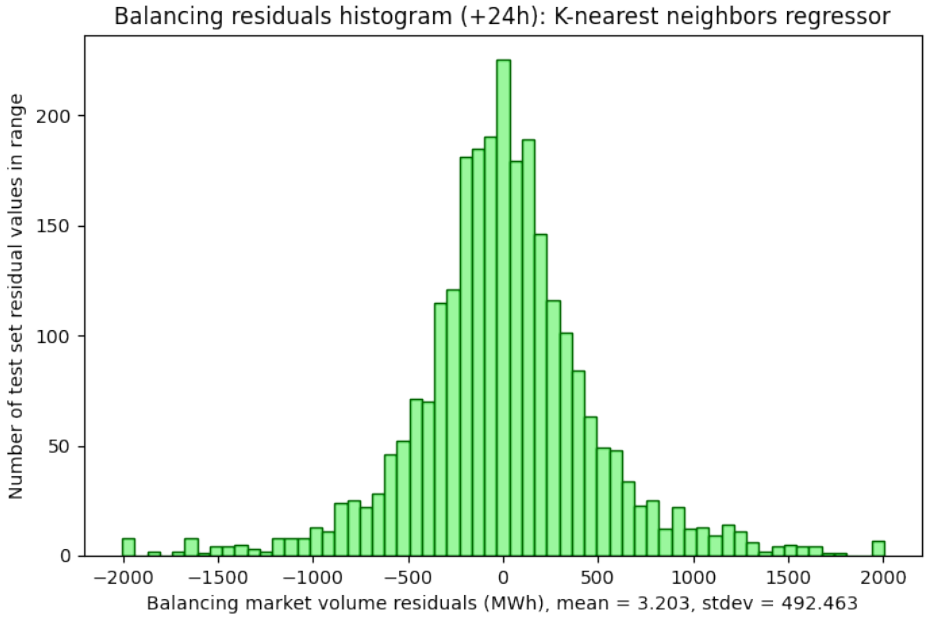

Figure 17: Residuals histogram for 24-hour forecasting, balancing market volume (MWh)

9. Comparison of Regression Algorithms

So far, all the results were obtained with the nearest neighbor regressor. And it makes sense to explore the performance of other algorithms on the same column configuration that is represented in Appendix 3. The output parameters were selected for 24-hour forecasting. The results shown in Table 4 and Table 5 include the comparison with classic instruments like multi-layer perceptron [7], support vector machine [8], and linear regression [9]. The constructors of Python objects representing regression algorithms with corresponding manually optimized hyperparameters are provided in Appendix 4.

Table 4
Comparison of R2 scores for regression algorithms on the testing dataset

Regression Algorithm	Bilateral	DayAhead	Intraday	Balancing
Histogram Gradient Boosting	**0.98734425**	**0.97273813**	**0.87836457**	**0.91963280**
Ada Boost Regressor	0.98008607	0.96134363	0.85172910	0.90325404
Gradient Boosting Regressor	0.97878979	0.96317970	0.84666374	0.90112536
Extra Trees Regressor	0.97461940	0.95963273	0.86484512	0.89815645
Nearest Neighbors Regressor	0.96751227	0.94895676	0.86066507	0.87555149
Random Forest Regressor	0.96680397	0.94718425	0.83167183	0.87304825
Support Vector Machine	0.93841639	0.90790177	0.78281964	0.78573216
Multi-Layer Perceptron (QNO)	0.93589612	0.90409299	0.75444413	0.79110787
Multi-Layer Perceptron (SGD)	0.93414003	0.90877942	0.77358025	0.81562885
Elastic Net Regressor	0.92924816	0.90300302	0.75547081	0.77908284
Linear Regression	0.92921485	0.90297901	0.75552627	0.77906737
Bayes Ridge Regressor	0.92502565	0.89258447	0.74195841	0.77884534

Table 5

Comparison of mean absolute percentage errors for regression algorithms

Regression Algorithm	Bilateral	DayAhead	Intraday	Balancing
Histogram Gradient Boosting	**0.00970813**	**0.03555078**	0.30680050	**3.41473968**
Ada Boost Regressor	0.01043662	0.03988947	**0.29964852**	3.70352770
Gradient Boosting Regressor	0.01167197	0.04196347	0.33108930	4.30691288
Extra Trees Regressor	0.01340342	0.04470638	0.39715721	3.68779356
Nearest Neighbors Regressor	0.01484254	0.04741409	0.31222159	4.16012392
Random Forest Regressor	0.01538395	0.05016324	0.44490383	4.21424456
Support Vector Machine	0.02049737	0.06506308	0.44628820	5.01011223
Multi-Layer Perceptron (QNO)	0.02201155	0.06895532	0.48465482	4.25173649
Multi-Layer Perceptron (SGD)	0.02328186	0.06766103	0.49768136	4.58588195
Elastic Net Regressor	0.02164488	0.06785682	0.46013951	5.91742745
Linear Regression	0.02167965	0.06799572	0.46071594	5.92935640
Bayes Ridge Regressor	0.02222508	0.06981429	0.50172640	5.90361401

It is worth noting that some algorithms do not natively support multi-output configuration, so it was needed to use the class MultiOutputRegressor to overcome this problem and cover four electrical energy markets with one machine learning model.

Tables 4–6 represent the following characteristics obtained for different machine learning models: R2 score, mean absolute percentage error, and mean absolute error. It appears, that for this specific task, the ensemble methods are much better than others, and the winning algorithm Histogram Gradient Boosting is one of them. Also, it is one of the fastest and it can flawlessly handle datasets with missing values. On the current dataset, the training phase takes about 20 seconds.

Two different training approaches were used for multi-layer perceptron: quasi-Newton optimizer (QNO) and stochastic gradient descent (SGD). The first algorithm uses analytic solution to weight optimization problem, while the second algorithm employs an iterative process to find the minimum of error function. In both cases the architecture was the same, the perceptron had four layers (that is two hidden layers). Empirically, this architecture was more successful than three or five-layer perceptrons. Meanwhile, all these configurations are universal approximators.

It makes sense to explain the high mean absolute percentage error for the balancing column in Table 5. This is not the error, also a couple of zero values in the input dataset were replaced to improve MAPE figures. As shown in Figure 16, most of the values for this column are located close to zero. So, the calculations according to formula (3) involve the division by a small value. In other words, the MAPE metric is just not that adequate for the balancing column.

Table 6

Comparison of mean absolute errors for regression algorithms (MWh)

Regression Algorithm	Bilateral	DayAhead	Intraday	Balancing
Histogram Gradient Boosting	**114.528749**	**136.017198**	107.623324	**287.495480**
Ada Boost Regressor	122.856359	151.671858	**107.042688**	308.036252
Gradient Boosting Regressor	137.181056	161.254944	119.274015	324.798985
Extra Trees Regressor	156.724684	165.609354	118.335948	320.904618
Nearest Neighbors Regressor	175.124238	183.691670	112.622368	344.727996
Random Forest Regressor	180.816224	187.724606	131.586872	360.659798
Support Vector Machine	239.541471	247.547199	150.333357	475.954169
Multi-Layer Perceptron (QNO)	257.264807	260.785606	159.517580	481.078516
Multi-Layer Perceptron (SGD)	270.583353	256.261356	157.399092	445.710338
Elastic Net Regressor	253.327126	259.857349	155.518920	487.916805
Linear Regression	253.691852	260.242048	155.743585	488.161734
Bayes Ridge Regressor	260.956241	269.083319	160.748064	487.498900

10.Conclusion

This paper demonstrates that proper forecasting model selection is a multi-stage process that may involve data selection, data preprocessing, data augmentation, selection of machine learning algorithm, optimization of hyperparameters, etc. While all computations for this work were done on a regular 8-core machine, the creation of the MLOps pipeline may require much more powerful computation resources.

The pre-trained model can be saved into a file for subsequent reuse in the production environment. There are two formats popular among Python developers: .joblib and .pickle. In addition, there is .onnx format that can be loaded not just in Python, but also in faster .NET or Java-based applications [10]. It is worth noting that Microsoft and other vendors invest significant resources into the development of multi-platform capabilities for machine learning [11].

There are dedicated solutions that can host the machine learning models using a microservice approach like Seldon Core [12]. According to this architecture, the serialized models are preloaded within docker containers and expose the HTTPS interface. So, the application can send input data vector as JSON document in REST API request. The HTTP response will contain the JSON document with predicted values.

The forecasting accuracy that was obtained for electrical energy markets is different. Nevertheless, the 1 % error for 24-hour forecasting of the bilateral market looks impressive. Such a forecast can be useful at the country scale to ensure required fuel supply, plan import/export operations, reduce electricity costs, etc. Similar research can be done for more specific datasets from commercial and state energy enterprises.

References

[1] A new model of the electricity market has been launched in Ukraine. URL: https://expro.com.ua/en/tidings/a-new-model-of-the-electricity-market-has-been-launched-in-ukraine.

[2] M. Osińska, M. Kyzym, V. Khaustova, O. Ilyash, T. Salashenko, Does the Ukrainian electricity market correspond to the European model?, Utilities Policy 79 (2022), 1–14. doi: 10.1016/j.jup.2022.101436.

[3] A. Doroshenko, D. Zhora, O. Savchuk, O. Yatsenko, Application of machine learning techniques for forecasting electricity generation and consumption in Ukraine, in: Proceedings of IT&I 2023, 2023, pp. 136–146. URL: https://ceur-ws.org/Vol-3624/Paper_12.pdf.

[4] Van Wyk, Encoding Cyclical Features for Deep Learning. URL: https://www.kaggle.com/code/avanwyk/encoding-cyclical-features-for-deep-learning.

[5] Scikit-learn: Machine Learning in Python. URL: https://scikit-learn.org/stable/

[6] Feature selection with Scikit-learn library. URL: https://scikit-learn.org/stable/modules/feature_selection.html.

[7] S. Haykin, Neural networks: a comprehensive foundation, Prentice Hall, Upper Saddle River, NJ, 1998.

[8] V. N. Vapnik, Statistical learning theory, Wiley, Hoboken, NJ, 1998.

[9] C. M. Bishop, Pattern recognition and machine learning, Springer, New York, NY, 2006. URL: https://www.microsoft.com/en-us/research/uploads/prod/2006/01/Bishop-Pattern-Recognition-and-Machine-Learning-2006.pdf.

[10] G. Novack, Deploy Sci-kit Learn models in .NET Core Applications. URL: https://towardsdatascience.com/deploy-sci-kit-learn-models-in-net-core-applications-90e24e572f64.

[11] X. Dupre, O. Grisel, Accelerate and simplify Scikit-learn model inference with ONNX Runtime. URL: https://cloudblogs.microsoft.com/opensource/2020/12/17/accelerate-simplify-scikit-learn-model-inference-onnx-runtime/

[12] V. Shanawad, Optimizing Custom Model Deployment with Seldon Core. URL: https://medium.com/@vinayakshanawad/serving-hugging-face-transformers-optimizing-custom-model-deployment-with-seldon-core-a593f6ea7549.

Appendix 1

The function that evaluates the input parameter set using nearest neighbors regressor:

```
def evaluate_input_features \
    (training_inputs, testing_inputs, training_outputs, testing_outputs):

    print("Evaluating the datasets using nearest neighbors regression model")
    evaluation_regressor = KNeighborsRegressor(n_neighbors = 5, weights =
        'distance', algorithm = 'auto', p = 1, metric = 'minkowski', n_jobs = 8)

    evaluation_regressor.fit(training_inputs, training_outputs)
    predicted_training_outputs = evaluation_regressor.predict(training_inputs)
    predicted_testing_outputs = evaluation_regressor.predict(testing_inputs)

    print_evaluation_metrics(training_outputs, testing_outputs, \
        predicted_training_outputs, predicted_testing_outputs)

def print_evaluation_metrics(training_outputs, testing_outputs, \
    predicted_training_outputs, predicted_testing_outputs):

    training_score = r2_score(training_outputs, predicted_training_outputs,
        multioutput = 'raw_values', force_finite = True)
    print(F"Training dataset R2 score(s): {training_score}")

    training_percentage_error = mean_absolute_percentage_error \
        (training_outputs, predicted_training_outputs, multioutput = 'raw_values')
    print(F"Training percentage error(s): {training_percentage_error}")

    train_standard_deviation = mean_squared_error \
        (training_outputs, predicted_training_outputs, \
        multioutput = "raw_values", squared = False)
    print("Training standard deviation(s):", train_standard_deviation)

    train_absolute_error = mean_absolute_error \
        (training_outputs, predicted_training_outputs, multioutput = 'raw_values')
    print("Training mean absolute error(s):", train_absolute_error)

    test_score = r2_score(testing_outputs, predicted_testing_outputs,
        multioutput = 'raw_values', force_finite = True)
    print(F"Testing dataset R2 score(s): {test_score}")

    test_percentage_error = mean_absolute_percentage_error \
        (testing_outputs, predicted_testing_outputs, multioutput = 'raw_values')
    print(F"Testing percentage error(s): {test_percentage_error}")

    test_standard_deviation = mean_squared_error \
        (testing_outputs, predicted_testing_outputs, \
        multioutput = "raw_values", squared = False)
    print("Testing standard deviation(s):", test_standard_deviation)

    test_absolute_error = mean_absolute_error \
        (testing_outputs, predicted_testing_outputs, multioutput = 'raw_values')
    print("Testing mean absolute error(s):", test_absolute_error)
```

Appendix 2

Identifying the features that provide higher information entropy:

```
random_forest = RandomForestRegressor \
    (n_estimators = 100, criterion = 'squared_error', ccp_alpha = 0.0)
random_forest.fit(dataset_inputs, dataset_outputs)
random_forest.feature_importances_

optimized_model = SelectFromModel(random_forest, \
    threshold = "0.12 * mean", prefit = True)
optimized_inputs = optimized_model.transform(dataset_inputs)

optimized_model.get_feature_names_out(input_names)
```

Appendix 3

Most informative input parameters selected with a random forest model:

```
selected_names = \
    ['Bilateral', 'DayAhead', 'Intraday', 'Balancing', 'BilateralM1',
    'DayAheadM1', 'IntradayM1', 'BalancingM1', 'DayAheadM2',
    'IntradayM2', 'BalancingM2', 'IntradayM6', 'BilateralM7',
    'DayAheadM7', 'IntradayM7', 'BalancingM7', 'BilateralM8',
    'DayAheadM8', 'IntradayM8', 'BilateralM9', 'DayAheadM9',
    'IntradayM9', 'BilateralM10', 'DayAheadM10', 'IntradayM10',
    'BilateralM13', 'BilateralM14', 'IntradayM14', 'BalancingM14',
    'BilateralM15', 'DayAheadM15', 'IntradayM15', 'BalancingM15',
    'BilateralM16', 'DayAheadM16', 'IntradayM16', 'BilateralM17',
    'DayAheadM17', 'IntradayM17', 'BilateralM18', 'DayAheadM18',
    'IntradayM18', 'IntradayM19', 'BilateralM21', 'BilateralM22',
    'DayAheadM22', 'IntradayM22', 'BilateralM23', 'DayAheadM23',
    'IntradayM23', 'BalancingM23', 'TempUkr', 'TempKiev', 'CosDay',
    'SinWeek', 'CosWeek', 'SinMonth', 'CosMonth', 'SinYear', 'CosYear']
```

Appendix 4

Regressor constructors with corresponding hyperparameters:

```
MultiOutputRegressor(HistGradientBoostingRegressor
    (loss = 'squared_error', learning_rate = 0.20, max_iter = 300,
    early_stopping = False, scoring = 'loss', random_state = 1))
MultiOutputRegressor(AdaBoostRegressor
    (estimator = DecisionTreeRegressor(criterion = 'squared_error',
    splitter = 'best', max_depth = None, min_samples_split = 2,
    max_features = None, random_state = 1), n_estimators = 10,
    learning_rate = 1.0, loss = 'square', random_state = 1))
MultiOutputRegressor(GradientBoostingRegressor
    (loss = 'squared_error', learning_rate = 0.39, n_estimators = 100,
    subsample = 1.0, criterion = 'squared_error', max_depth = 6,
    random_state = 1, max_leaf_nodes = None, ccp_alpha = 0.0))
ExtraTreesRegressor(n_estimators = 100,
    criterion = 'squared_error', max_depth = None, max_features = 1.0,
    bootstrap = False, n_jobs = 8, random_state = 1, ccp_alpha = 0.0)
KNeighborsRegressor(n_neighbors = 3, weights = 'distance',
    algorithm = 'auto', p = 1, metric = 'minkowski', n_jobs = 8)
RandomForestRegressor(n_estimators = 100,
    criterion = 'squared_error', max_features = 1.0, bootstrap = True,
    ccp_alpha = 0.0, n_jobs = 8, random_state = 1)
MultiOutputRegressor(NuSVR(nu = 0.4, C = 1000000.0,
    kernel = 'rbf', gamma = 'scale', shrinking = True, max_iter = -1))
MLPRegressor(hidden_layer_sizes = (200, 200,),
```

```
    activation = 'relu', solver = 'lbfgs', alpha = 0.0000, max_iter = 5000,
    random_state = 1)
MLPRegressor(hidden_layer_sizes = (200, 200,),
    activation = 'relu', solver = 'adam', alpha = 0.0002, max_iter = 200,
    batch_size = min(50, training_set_size), shuffle = True, random_state = 1,
    early_stopping = False)
ElasticNet(alpha = 1.0, l1_ratio = 1.0, fit_intercept = True,
    max_iter = 1000, positive = False, random_state = 1, selection = 'cyclic')
LinearRegression(fit_intercept = True, n_jobs = 8)
MultiOutputRegressor(BayesianRidge(max_iter = 300,
    tol = 0.001, alpha_init = None, lambda_init = 1.0, fit_intercept = True))
```