Supercomputer power consumption prediction using machine learning, nonlinear algorithms, and statistical methods

Jiří Tomčala
IT4Innovations, VSB - Technical University of Ostrava, 17.listopadu 2172/15, 70833 Ostrava-Poruba, Czech Republic
E-mail: jiri.tomcala@vsb.cz

Abstract. This work describes various methods of time series prediction. It illustrates the differences between machine learning methods, nonlinear algorithms, and statistical methods in their approach to prediction, and tries to explain in depth the principles of some of the most widely used representatives of these types of prediction methods. All of these methods are then tested on a time series from the real world: the course of power consumption of a supercomputer infrastructure. The reader is gradually acquainted with data analysis, preprocessing, the principle of the methods, and finally with the prediction itself. The main benefit of the work is the final comparison of the results of this testing in terms of the accuracy of the predictions, and the time needed to calculate them.

1. Introduction
Supercomputing infrastructure demands a great deal of energy, therefore it would be highly useful to be able to at least roughly predict its energy consumption. Users load it more or less randomly at more or less random times with more or less random calculations where more or less random errors can occur. So, the behavior of the users depends on many facts that may depend on each other non-linearly, and the number of such facts is so great that it is computationally intractable to model. This is essentially the definition of a complex system.

Hence, this infrastructure then becomes a complex system in terms of its consumption. The behavior of such systems seems to be chaotic at first glance, but in fact they often contain hidden deterministic patterns that make it possible to predict this behavior to a greater or lesser degree. However, it should be noted that this is something different than deterministic chaos [1], which in chaos theory is defined as the emergence of random, unpredictable behavior of a system that is, paradoxically, precisely determined by clear rules and clear initial conditions.

In the real world, however, the exact analytical description of the system is often not known and only measured data is available. They can be used to estimate their further development using different approaches. This work deals with the analysis of the principle and accuracy of machine learning, nonlinear, and statistical approach.

For the sake of completeness, it is worth mentioning that other examples of complex systems are systems from very diverse sources, such as an anthill, the nervous system, social networks, weather, transport, the stock exchange, etc.
2. Data analysis

This section will focus on the analyzed data in more detail. What exactly does this data contain? It is the course of the active power input of one substation, which supplies part of the Salomon cluster and also part of the Anselm cluster, both of which are located at IT4Innovations National Supercomputing Center [2]. At the request of the responsible employees of the Supercomputing Center, all original data from the infrastructure were normalized in a simple way, so that the total mean of the analyzed power consumption time series is 1 and its values are dimensionless numbers. This proportional adjustment in no way destroys the information hidden in the original course of power consumption, and the calculated prediction can be converted back to the original values if necessary.

User tasks take place on the computing nodes of these clusters, and it can be assumed that each such task has its typical time structure, which is inevitably reflected in the computing power requirements of these nodes and subsequently in their electricity consumption. These waveforms of the electricity consumption of the individual computing nodes are added together, thus creating a new time series of the total consumption, which then contains all the time structures of the running tasks. Since the number of these running tasks is finite and their time structure often shows repetitive patterns, it can be assumed that the resulting time series of electricity consumption is not purely chaotic, and that some repetitive patterns reappear in it. A simplified example of such pattern merging can be seen in Figure 1.

![Figure 1: A simplified example of pattern merging. Power consumption of 3 computational nodes on which different tasks are running at the same time. Regular patterns brought together give a regular pattern again, but more complicated. However, still with some (non-zero) degree of predictability.](image-url)
The presence of these patterns reduces the entropy value of the resulting time series, which according to [3] increases its predictability. Of course, these repetitive patterns have various time periods, and it may happen that the longest of them exceeds the length of the analyzed part of the time series, on the basis of which the prediction of further courses is made. If such a pattern has a more significant amplitude, then it will manifest itself in the resulting time series of electricity consumption as a trend, and can be removed during preprocessing.

At first glance, in the analyzed time series of power consumption in Figure 2, a slight ripple can be observed with an approximate period of 1440 minutes, which is 1 day. Consumption increases regularly in the morning, which is in line with the expected behavior of users; they start running new computing tasks at the beginning of the working day. This behavior of users thus creates the so-called seasonality of this analyzed time series. Before creating the prediction itself, it is advisable to remove the seasonality as a part of preprocessing.

Figure 2: An analyzed time series of power consumption. It is normalized measured input power from the IT4Innovations supercomputer infrastructure. The measured time period is from 1 AM, 2 November until 9 PM, 5 November, 2017.

3. Preprocessing
Performing preprocessing [4] increases the chances for success of any prediction. This includes checking the completeness of the data and possible imputation of missing values or removing erroneous data points. Furthermore, as already mentioned, the preprocessing also includes the removal of the trend and seasonality of the time series, the course of which is predicted.

Sometimes, before prediction, it is also good to find out whether the variance or covariance of a time series is a function of time, but in this case it will not be necessary. A quick look at the course of energy consumption in the time series in Figure 2 shows that both of these attributes are approximately constant.

At the end of the preprocessing, it is recommended to perform standardization and normalization, which in the case of this data from the Supercomputing Center has already been done at the very beginning, as mentioned in Section 2.

3.1. Data completeness
When checking the completeness of the power consumption data, less than 0.1% of missing values were found, and these were always isolated outages of one value. In order to preserve the
synchronicity of the whole time series, the missing places were not simply removed, but values based on neighboring values were inserted into their positions. Due to the very small number of these outages, a value corresponding to the ordinary arithmetic mean of the neighbouring values could be inserted into their place. This data was then analyzed for erroneous values and outliers, but none were found.

3.2. Trend stationarity
All the prediction methods used in this article will provide more accurate results if the time series is trend stationary. Simply put, this time series property tells us that the mean of time series values is not a function of time, and that it remains more or less constant. Of course, it depends on what length of time window is chosen for a possible trend stationarity test. To test our time series of power consumption, its entire course (as shown in Figure 2) was used as a time window.

The two most common tests were used to determine trend stationarity, namely the Augmented Dickey-Fuller test [5] and the Phillips-Perron test [6]. Both of these tests are available in the R [7] package \texttt{tseries} [8], which was also used to perform them. In both tests, the null hypothesis of trend non-stationarity of the analyzed time series is determined, and as an alternative hypothesis is its trend stationarity. After performing both tests, the results showed a p-value of less than 0.01, so both tests rejected the null hypothesis and thus it can be said with a high degree of certainty that our analyzed time series is a trend stationary.

3.3. Seasonality
By removing seasonality, the original patterns from which the resulting time series is composed stand out more (see Figure 1). Seasonality is actually just another regular pattern that comes into play, and by removing it the prediction methods will more easily detect unknown underlying patterns, allowing for more accurate future forecasts.

First, it is necessary to determine the course of the seasonal component in the time series. This can be done, for example, with the \texttt{decompose} function in an R language environment. Assume that the normalized time series of power consumption is stored in \texttt{powerTS}. Then simply subtract this seasonal component \texttt{powerTS\_seasonal} from the original time series, as shown in the following code.

```r
code
powerTS\_tsForm <- ts(powerTS,frequency=1440)
powerTS\_seasonal <- decompose(powerTS\_tsForm)$seasonal
powerTS\_seasonless <- powerTS - powerTS\_seasonal
```

Another possibility is to convert a part of the analyzed time series of the assumed seasonality period length into the frequency spectrum, remove high and medium frequencies from this spectrum, and then convert it back to the time domain. This filters out fast and medium-fast patterns caused by computational tasks, leaving only a slow component that should roughly correspond to the course of the season, in this case the course of the daily cycle. This smoothed waveform is then extended to the entire length of the original analyzed time series and subtracted from it. The following code shows a practical procedure in the environment of the R language using direct and inverse fast Fourier transform (FFT).

```r
code
powerTS\_season <- powerTS[1:1440]
freq\_powerTS\_season <- mean(powerTS\_season)
freq\_powerTS\_season_smooth <- fft(powerTS\_season)
freq\_powerTS\_season_smooth[-(1:8)] <- 0
```

The resulting time series and the seasonal component calculated by this procedure are shown in Figure 3.
Figure 3: Removing seasonality using the `decompose()` function.

```r
powerTS_season_smooth <- fft(freq_powerTS_season_smooth, inverse=TRUE) / length(freq_powerTS_season_smooth)
powerTS_season_smooth <- as.numeric(powerTS_season_smooth)
powerTS_smooth <- rep(powerTS_season_smooth, len=length(powerTS))
powerTS_seasonless <- powerTS - powerTS_smooth
```

Figure 4 shows the seasonal component as well as the resulting filtered time series created by the above procedure.

As can be seen from Figure 3, the seasonality detected by the `decompose()` function exhibits large and rapid fluctuations, which are then reflected in the resulting time series. For the purposes of future prediction this is very undesirable, and therefore in this case it is better to choose the second procedure, which uses the fast Fourier transform. Thus, the resulting seasonless time series `powerTS_seasonless` from Figure 4 will be used for the following prediction calculations.
4. Statistical methods
When it is necessary to make a forecast for the near future for several time series at once, it is advisable to use simple statistical methods such as the Simple Moving Average (SMA) or Exponential Smoothing (ETS).

The SMA’s prediction is the arithmetic or weighted average of a number of consecutive points of the time series that is predicted. The ETS method is similar to the SMA method except that the newer data points are given more weight. These methods give quick results, but may fail in more complex time series. Therefore, a much more advanced method is currently used in real statistical prediction: an Auto-Regressive Integrated Moving Average.

4.1. Auto-Regressive Integrated Moving Average (ARIMA)
On the basis of the model created by the ARIMA method, J. Durban made the first predictions as early as 1960. The ARIMA method creates a class of time series models that consist of three parts: AR, I, MA.
The Auto-Regressive part (AR) means that it can use a linear combination of previous values to express the following value. The order of this auto-regressive component is called $p$, and expresses how many time steps back this part of the model counts.

Integrated part (I) means that the data values have been replaced with the difference between their values and the previous values. This differencing can be performed more than once. The order of the integrating component is denoted $d$, and signifies how many times the differencing is applied. This process is mainly used to remove the trend from the time series. Because during preprocessing the power consumption time series was found to not contain a trend, the value of this parameter $d$ should be 0.

The Moving Average part (MA) indicates that part of the error (residue) of the time series can be explained as a linear combination of past errors. The order of the MA component is denoted $q$, and (similarly to the AR parameter $p$) it expresses from how many previous time intervals past errors occur in the model.

The overall ARIMA prediction equation is as follows:

$$\hat{y}_t = \sum_{i=1}^{p} \alpha_i \nabla^d x_{t-i} - \sum_{i=1}^{q} \beta_i \varepsilon_{t-i} - \sum_{i=1}^{d} (-1)^i \binom{d}{i} x_{t-i}$$  \hspace{1cm} (1)

where $x = (\ldots, x_{t-3}, x_{t-2}, x_{t-1})$ is a time series that is predicted, the $\alpha_i$ are the parameters of the AR part of the model, the $\nabla^d$ is the backward difference operator of the order $d$ which is the parameter of the I part, the $\beta_i$ are the parameters of the MA part, the $\varepsilon_{t-i}$ are past forecast errors, and the $\hat{y}_t$ is the prediction itself.

It is worth mentioning that the for non-zero parameter $d$ the differencing is performed before the prediction and then, after the prediction, the reverse process (undifferencing) occurs.

The most important thing is to correctly identify the $p$, $d$, and $q$ parameters. The $\alpha_i$ and $\beta_i$ values are then estimated by the model that is created by the software used, in our case the R package `forecast` [9]. Book [10] is recommended for a deeper study of this issue.

### 4.1.1. ARIMA parameters tuning

The key source for determining optimal values of these parameters is the autocorrelation function (ACF) and the partial autocorrelation function (PACF) of the time series that is predicted. The R package `forecast` was used to calculate the course of these functions and they are shown in Figure 5.

When assessing the course of the autocorrelation function (ACF), the rule is that if its values are positive for more than 10 lags, then the time series needs to be differenced to be more stationary. At the same time, if the ACF value for the lag 1 is too negative, then the time series is probably over-differenced.

As already mentioned, based on the data analysis it is assumed that the parameter $d$ is zero, but it can be seen from Figure 5 that according to the above rule it is necessary to difference our time series, thus $d = 1$.

Figure 6 shows our time series after differencing. The ACF and PACF of this time series is shown in Figure 7 and it is clear from their course that no further differencing will be needed. There is even a possibility that the resulting time series is over-differenced because the ACF value for the lag 1 is quite negative. In this case, it is useful to compare the standard deviation of the time series before and after differencing and then choose the level of differencing that provides the time series with the smaller one. In our case, the processed time series has after differencing half the standard deviation of the time series before differencing and therefore the value $d = 1$ remains unchanged.

The $p$ and $q$ parameters can also be determined from the ACF and PACF values. If the ACF values for an increasing number of lags gradually decrease to zero, then the number of significant peaks in the course of the PACF gives the value of the parameter $p$. Likewise, if the
Figure 5: The autocorrelation function (ACF) and the partial autocorrelation function (PACF) of the power consumption time series with seasonality removed.

PACF values for an increasing number of lags gradually decrease to zero, then the number of significant peaks in the course of the ACF gives the value of the parameter $q$.

The gradual decrease of the ACF and PACF to zero can clearly be seen from Figure 7, so this necessary condition for determining the parameters $p$ and $q$ is met. Figure 8 shows the first 37 lags of the ACF and PACF. It is therefore possible based on the above-mentioned rule to estimate from this picture that the approximate value of parameter $p = 8$ and $q = 6$.

The R package `forecast` also contains the `auto.arima()` function, which can estimate the $p$, $d$, and $q$ parameters. For our time series, this function estimated the following parameter values: $p = 0$, $d = 1$, $q = 2$. When experimenting with both models, it was found that the ARIMA(8,1,6) model is more accurate than the ARIMA(0,1,2) model estimated by the above-mentioned `auto.arima()` function. However, it is also slower, so both models have been included in the final comparison.
Figure 7: The autocorrelation function (ACF) and the partial autocorrelation function (PACF) of the differenced power consumption time series. No further differencing is needed.

Figure 8: First 37 lags of the autocorrelation function (ACF) and the partial autocorrelation function (PACF) of the differenced power consumption time series.

5. Nonlinear algorithms

The idea of nonlinear modelling is simple. All that is required is to find a suitable functional form that is flexible enough to model the course of the time series that is predicted. One of the most commonly used approaches is to take a system of basis functions and, using an optimization process, find their linear combination, which with the only slightest error approximates the current time series development.

While these basis functions may be non-linear, this optimization process ultimately leads to a set of linear equations, which greatly facilitates the search for the optimal parameters of the above-mentioned linear combination. Book [11] deals with nonlinear algorithms and nonlinear time series analysis in more detail.

The following subsections briefly introduce the most common variants of this nonlinear
modeling. For the final comparison of the prediction methods’ results, the zeroth algorithm (the last one) was chosen, and tuning of its parameters was performed.

5.1. Polynomials
The prediction is realized by constructing a multivariate polynomial that has the best fit to the previous time series data points. The advantage of polynomials is that many ways of working with them have already been developed. The degree of the polynomial should be determined with a view to avoiding under-fitting, which may occur if the degree of the polynomial is too low. On the other hand, too high a polynomial degree could cause over-fitting.

5.2. Radial basis functions (RBF)
The prediction is created by approximating the previous time series course by the sum of scalar radial basis functions whose only parameter is their shift on the timeline.

The resulting approximation then reads

\[ F(x) = \alpha_0 + \sum_{i=1}^{k} \alpha_i \Phi(\|x - y_i\|) , \tag{2} \]

where \( \Phi \) is a radial basis function and \( y_i \) is its shift on the timeline.

The timeshifts \( y_i \) should be fixed forwards and reasonably well distributed. Then the search for an approximation \( F \) leads again to a set of linear equations whose solution determines the values of the coefficients \( \alpha_i \). The Gauss function is often used as the radial basis function.

5.3. Zeroth algorithm
The name of this algorithm is derived from the fact that it is based on a zeroth-order approximation of the dynamics.

The prediction of the continuation of the time series is calculated as the arithmetic mean of the continuations of all sub-sequences in the previous development that are in the predetermined neighborhood of the last sub-sequence in the phase space of these sub-sequences.

This complicated sentence can be simply expressed analytically as

\[ \hat{y}_t = \frac{1}{|U_\varepsilon(x_{last})|} \sum_{x_k \in U_\varepsilon(x_{last})} x_{t-k} , \tag{3} \]

where \( x_{last} \) is the last sub-sequence, \( x_k \) is a sub-sequence in the previous development of the time series that is predicted, and \( |U_\varepsilon(x_{last})| \) means number of sub-sequences in the previous development that are in the neighborhood \( U_\varepsilon \) of the last sub-sequence. For the sake of simplicity, this neighborhood can be defined by the maximum norm, but it is possible to use any other norm. \( U_\varepsilon \) then contains all sub-sequences \( x_k \) that satisfy the condition \( \|x_{last} - x_k\| \leq \varepsilon \). The sum is then performed over all \( x_{t-k} \), which are continuations of all these sub-sequences. Since this sum is then divided by the number of these sub-sequences, the resulting prediction \( \hat{y}_t \) is equal to the arithmetic mean of these continuations.

The sub-sequences parameters \( m \) and \( \tau \) determine their embedding dimension and the delay time. Their effect on the shape of the sub-sequences can clearly be seen from expressions for \( x_{last} \) and \( x_k \) in Eq. (3).
5.3.1. Zeroth algorithm parameters tuning

The parameters that can affect the accuracy of this algorithm are $m$, $\tau$, and the radius of the neighborhood $\varepsilon$.

The parameter $m$ is essentially the length of the sub-sequences analyzed. It is obvious that the longer the previous development of the time series that is taken into account, the more accurate the forecast will be. On the other hand, if all the available previous data form one long sub-sequence, then based on this one observation, only by great coincidence will an accurate prediction be made. Part of tuning this algorithm will therefore be to determine the optimal ratio of the number of sub-sequences to their length.

The delay time $\tau > 1$ allows a larger period of previous data to be taken into account when making a prediction with the same sub-sequence length. It is actually a downsampling and it is suitable in the case of oversampled input data, where a large part of the values does not bring any new information. However, this is not the case with our energy consumption time series and therefore this parameter will be set to $\tau = 1$ during the whole tuning of this algorithm.

As mentioned above, the parameter $\varepsilon$ defines the size of the neighborhood in which similar sub-sequences are searched. If the $\varepsilon$ is too small, then the small number of sub-sequences found will cause the calculated prediction to be too specialized and take into account only a small portion of the previous data. On the other hand, using a too large $\varepsilon$ gives a prediction that is too general, and its value will essentially be a simple arithmetic mean of the previous data.

During the experiments with various values of $m$ and $\varepsilon$ was measured the results, which are shown in Figure 9.

Figure 9: Accuracy of the zeroth method depending on the values of the parameters $m$ and $\varepsilon$.

For clarity, this dependence is shown in Figure 10 again, but only for the optimal value of the parameter $m$.

Figure 10: Dependence of the zeroth method accuracy on the parameter $\varepsilon$ at the optimal value $m = 31$.

Although optimal setting may vary slightly over the course of the data, a constant setting of $m = 31$, $\tau = 1$, and $\varepsilon = 0.151$ will be used for the final comparison.
6. Machine learning methods

The principle of machine learning methods is well known, however, for completeness, their common basic features are presented here. Before making any prediction by machine learning methods, the existing observations are transformed into a training set. Then, using various procedures, a mathematical model that can estimate the continuation of the currently measured new data is created.

The training set can be constructed in various ways. Probably the simplest one is as follows. Let there be a measured time series $\mathbf{x} = (\ldots, x_{t-3}, x_{t-2}, x_{t-1})$. Then its sub-sequences can be considered as an individual observations of some development in this time series. The last value of these sub-sequences can then be considered as the result of the development of the measured value in the sub-sequence. The observations found in this way can be compiled into a table, and if the same sub-sequence structure is selected, a training set is created that can then be used to teach the machine learning model.

An example of how to build a simple training set is given in Table 1.

| observation | 1 | 2 | 3 | 4 | 5 | ... |
|-------------|---|---|---|---|---|-----|
| predictor 2 | $x_{t-3}$ | $x_{t-4}$ | $x_{t-5}$ | $x_{t-6}$ | $x_{t-7}$ | ... |
| predictor 1 | $x_{t-2}$ | $x_{t-3}$ | $x_{t-4}$ | $x_{t-5}$ | $x_{t-6}$ | ... |
| target      | $x_{t-1}$ | $x_{t-2}$ | $x_{t-3}$ | $x_{t-4}$ | $x_{t-5}$ | ... |

The training set from Table 1 would then be in the form

$$\mathbf{X} = \{(\mathbf{x}^{(1)}, y^{(1)}), (\mathbf{x}^{(2)}, y^{(2)}), (\mathbf{x}^{(3)}, y^{(3)}), \ldots\},$$

where

$$\mathbf{x}^{(1)} = (x_{t-3}, x_{t-2}), \quad y^{(1)} = x_{t-1},$$

$$\mathbf{x}^{(2)} = (x_{t-4}, x_{t-3}), \quad y^{(2)} = x_{t-2},$$

$$\mathbf{x}^{(3)} = (x_{t-5}, x_{t-4}), \quad y^{(3)} = x_{t-3},$$

$$\ldots$$

A common problem with these machine learning models is under-fitting, when the training set does not contain enough information and the model describes the reality too generally, or over-fitting, when on contrary the training set contains too much information and the model is then not able to generalize at all. As in the previous methods, the optimal settings will also be sought here by tuning the parameters.

There are many types of machine learning model; the following subsections describe the most important of them and also the tuning of their parameters when applying them to the above-mentioned power consumption time series.

6.1. k-nearest neighbors (KNN)

The training set is the model itself. This algorithm is searching for the k nearest observations (neighbors) in the training set and the prediction is calculated as the arithmetic mean of their targets.
Figure 11: The principle of the KNN method. Each black point represents one observation from the training set. The red dot represents the last sub-sequence whose continuation this method estimates from the arithmetic mean of the 5 nearest observation targets. In this example, the Euclidean norm was used to calculate the distance between the observations, and therefore the area in which the 5 nearest observations lie is in the form of a circle.

The resulting prediction of this method could be mathematically expressed as

\[ \hat{y}_t(X, x, k) = \frac{1}{k} \sum_{i=1}^{k} z_i, \tag{5}\]

\[ z_i = \{ y^{(\xi)} : \|x^{(0)} - x^{(\xi)}\| \leq \|x^{(0)} - x^{(\zeta)}\|; \ \xi \in K, \ \zeta \notin K, \ |K| = k, \ (x^{(K)}, y^{(K)}) \subset X\}, \]

where \( x \) is a time series that is predicted, \( x^{(0)} \) is the last sub-sequence \((\ldots, x_{t-2}, x_{t-1})\), \( K \) is a set containing indexes of the \( k \) nearest sub-sequences, and \( X \) is the training set.

6.1.1. KNN parameters tuning As can be seen from Eq. (5), the number of neighbors \( k \) is essentially the only parameter that can be changed when predicting by this method. It would be good to try different values of \( k \) and choose the best value according to the achieved accuracy of the given time series prediction. This is exactly the way used by the R package caret [12] when determining the prediction by this KNN method. However, due to the shortening of the prediction calculation time, the parameter \( k \) will be set to the optimal constant value during the final comparison.

Prediction accuracy of this method can also be affected by the number of predictors. Therefore, a set of calculations was performed on part of the analyzed data and the results are shown in Figure [12]. The lowest RMSE was measured at 41 predictors, and this number of predictors will also be used in the calculations for the final comparison.

For completeness, Figure [13] shows the dependence of the accuracy of this method on the number of neighbors \( k \), which is evaluated by the software used.

This particular graph holds only for a constant number of predictors \( N = 41 \). However, its shape varies for different numbers of predictors, so it cannot generally be said that the smallest
Figure 12: Illustration of the dependence of the accuracy of KNN prediction of the course of power consumption on the number of predictors.

Figure 13: Illustration of the dependence of the KNN prediction accuracy on the number of neighbors $k$ while the number of predictors $N = 41$.

$k$ is always the best choice. But for this case, the optimal value of the parameter $k = 5$, and this value will also be used in the final comparison.

6.2. Decision tree (DT)
Another prediction machine learning method is a decision tree, the principle of which is completely different. Based on observations from the training set, this method creates a decision-making structure (tree) that assigns the corresponding targets to all of these observations. The prediction is then determined by applying this decision tree to new observations. When, as in this case of power consumption, the predicted value and predictors are continuous, these decision trees are called regression trees. A simple example of such a tree is shown in Figure 14. Of course, the more branched the tree is, the more accurate the regression and hence the prediction.

The decision tree method was not included in the final comparison, but instead the more advanced random forest method, which is based on the decision tree and which is described in the following section, was included. For this reason, the tuning was performed only for the random forest method.

6.3. Random forest (RF)
The random forest method was introduced in 1995 by Tin-Kam Ho [13]. As a model, this method creates a multitude of decision trees with randomly selected predictors. More aptly, this method then should be called the forest of random trees. The principle of this method is that each of these random decision trees makes a prediction and the resulting prediction of the whole forest is, in this regression case, the mean of the individual trees predictions. This is illustrated in Fig. 15.
6.3.1. RF parameters tuning  The main parameters of the RF method are the number of trees (ntree), the number of randomly selected predictors in each tree (mtry), the maximum depth of trees, the minimum number of samples required for tree branching, and the minimum number of samples for each tree leaf.

The first two mentioned parameters ntree and mtry have the greatest influence on the accuracy of the prediction, and therefore the attention will be focused only on them during tuning. The remaining parameters will be estimated by the software used, which in this case will also be a caret package.

When experimenting, it is advisable to use the property of the used software caret, which for a given value ntree calculates the accuracy of the prediction for different values of mtry, and for the resulting prediction then uses the most accurate variant.

To determine the optimal value of ntree, a series of predictions with a constant number of predictors was performed. The resulting dependence of the accuracy of this method on the number of trees is shown in Figure 16.

It can be seen from Figure 16 that the accuracy of this method does not improve much from a certain value of ntree, and since the demand on computational resources for this prediction calculation increases with the increasing number of trees, it is appropriate in this case to choose the optimal value of ntree = 13.

Although the caret software finds the optimal value of mtry for a given ntree, it is interesting to see Figure 17 which shows how the accuracy of this method changes with the increasing value of mtry.
Figure 16: Illustration of the dependence of the power consumption RF prediction accuracy on the number of trees ($ntree$).

Figure 17: Illustration of the dependence of the power consumption RF prediction accuracy on the number of randomly selected predictors ($mtry$) while $ntree = 13$.

It is clear from Figure [17] that the optimal value of the parameter in this case is $mtry = 19$. Since this course has a similar shape also for other values of $ntree$, it can be said in general that the optimal number of randomly selected predictors should be about half of the total number of predictors.

6.4. Artificial neural networks (ANN)

This method is inspired by real neural networks. Learning in this network consists of adjusting the weights of the connections between the individual neurons so that all observations from the training set give the corresponding targets at the output of the neural network. Then, as with the decision tree, the prediction is determined by applying this model to new observations. The ANN structure is clearly shown in Fig. [18].

Given that the paradigm of artificial neural networks is generally well known and is described in many publications, such as [14], it makes no sense to describe it further here. It is more interesting to focus on determining the optimal parameters of this method for our time series of the power consumption.

6.4.1. ANN parameters tuning

A key part of the prediction using the artificial neural network method is the learning process of this network, in which the optimal weights of connections between neurons are set by means of forward and backward propagation. The rate of change of these weights is corrected by the parameter learning rate, which, however, is changed during the learning of the network, and therefore its setting was left to the used software caret. The parameters that are important for accuracy (and also for the speed) of prediction are mainly the count of hidden layers and the quantity of neurons they contain.

Due to the high degree of complexity of the supercomputer power consumption time series, the maximum number of layers was used, which is 3 in the case of the software used. In addition,
when creating a model from only two or one hidden layers with the same total number of neurons, the prediction calculation took significantly longer.

In order to find the optimal number of neurons in each layer, a series of experiments was performed in which the number of neurons in one layer was gradually increased, while the number of neurons in the remaining layers was kept constant. The result of one series can be seen in Figure 19, where the number of neurons in the second layer was changed.

It can be seen that as the number of neurons increases above the number of neurons in the other layers, the accuracy of the prediction no longer improves. The same property was also observed during similar experiments with the first and third layer. Thus, the most optimal distribution of neurons into layers is one in which these layers contain approximately the same number of neurons.

During further experimentation, this same number of neurons was gradually increased in all layers, and as can be seen from Figure 20, its increase above approximately 15 neurons no longer affected the accuracy of the prediction. So, for the final comparison, a network of neurons will be used, consisting of 3 hidden layers, each containing 15 neurons.

6.5. Extreme gradient boosting (XGBoost)
This method forms an ensemble of some (weaker) models, typically decision trees, and calculates a prediction error for each model. This creates an error function over the model space whose global minimum is searched iteratively by using optimization methods, such as the steepest
(gradient) descent. It also includes using quantile binning in the decision tree training, adding a regularization term to the objective to improve generalization, and supporting multiple forms of the error function. Fig. 21 is a simplified illustration of the principle of minimizing the prediction error depending on combinations of the use of various prediction models.

Interestingly, although it is a fairly new algorithm (it was designed in 2016 by Tianqi Chen and Carlos Guestrin in [15]), it is currently one of the most successful.

6.5.1. XGBoost parameters tuning

There are many parameters for this method that could be tuned, however, the most important of them are number of boosting iterations $nrounds$ (sometimes also referred to as the number of trees), maximum tree depth, learning rate $\eta$ (sometimes also referred to as the shrinkage), regularization parameter $\gamma$, and the minimum sum of instance weight (sometimes also referred to as the minimum leaf weight).

From experiments with the time series of energy consumption using the software caret, it was found that in terms of the accuracy of this method, it is best to set the maximum tree depth to 1 and leave the regularization parameter $\gamma$ at the default value of 0.

The learning rate $\eta$ is the shrinkage that is performed at each step. To get the most out of XGBoost, the learning rate $\eta$ must be set as low as possible. However, as the learning rate $\eta$ decreases, it is required to perform many more steps ($nrounds$) to reach the optimum. Therefore, these two parameters are somehow connected and Figure 22 shows how their values affect the accuracy of the XGBoost method prediction applied to the analyzed power consumption time series.
The minimum sum of instance weight defines the minimum sum of weights of all observations required in a leaf of the tree. Setting this parameter correctly is one of the many ways to control over-fitting. Higher values prevent the model from learning properties that could be highly specific to the particular sample selected for the tree. Figure 23 shows the dependence of the accuracy of a series of predictions on the value of this parameter when setting constant optimal values of other parameters from the previous measurement.

The above figures show the optimal values of the XGBoost method parameters $nrounds = 22$, $\eta = 0.23$, minimum sum of instance weight 20, maximum tree depth 1, and $\gamma = 0$, which will also be used in the final comparison of the accuracy of individual methods.

7. Final comparison

During the experiments, RMSE (Root Mean Square Error) will be used again as the main measure of the accuracy. RMSE has the advantage of giving an idea of the accuracy of the predictions over a period of time.

To make the final comparison as true as possible, the same number of samples (300) with the same number of predictors (41) will be used for all methods. In this way, a time window with a width of 340 minutes will be created that will contain data on the basis of which each of the methods will try to predict the value of the power consumption in the 341st minute. So, 5160 predictions will be made by each method. It should be emphasized that a new model from the previous 340 minutes of data will always be created before each prediction. This will ensure that all predictions of all methods are always made from the same and most recent data.

By moving this time window over the entire analyzed power consumption time series, a time course of prediction error will be generated for each method, from which the total RMSE will then be calculated. To smoothly display the development of this prediction error, the course
of the RMSE value will also be calculated for a floating window with a size of 300 minutes. In addition, for each method, the run-time required to calculate all these predictions will also be measured.

As already mentioned above, machine learning predictions will be calculated using R software package caret [12], statistical ARIMA predictions using R software package forecast [9], and the non-linear zeroth algorithm using a simple program written in R specifically for this paper.

8. Results

All results presented here were calculated on the Salomon cluster, which is part of the IT4Innovations infrastructure, located at the National Supercomputing Center of the Czech Republic.

During the calculations of the predictions of the whole time series using an artificial neural network, it was found that even if the optimal values of the parameters were used, the total time of their calculation exceeds 15 hours. This was far more than all other methods needed and therefore another variant of the artificial neural network with an increased value of the threshold parameter was added to the final comparison.

The threshold parameter determines the maximum possible error of an artificial neural network at which this network is considered to be correctly learned. By increasing this parameter, the time required for training an artificial neural network can be significantly reduced at the cost of its lower accuracy.

The results obtained for the default threshold value of 0.01 were marked as ANN 1 and for the increased threshold value of 0.02 were marked as ANN 2.

Figure 24: The course of accuracy of all methods in predicting the entire supercomputer power consumption time series. The accuracy is represented here by the RMSE of a floating window with a length of 300 minutes.
Table 2: Summary of overall results, sorted by total RMSE and also by the total run-time required to calculate the predictions of the entire supercomputer power consumption time series.

| Method      | Total RMSE [-] | Method      | Total run-time [s] |
|-------------|----------------|-------------|--------------------|
| ARIMA(8,1,6) | 0.02722        | Zeroth      | 23                 |
| ARIMA(0,1,2) | 0.02738        | ARIMA(0,1,2)| 58                 |
| XGB         | 0.02773        | KNN         | 3240               |
| RF          | 0.02836        | XGB         | 4515               |
| Zeroth      | 0.03231        | ARIMA(8,1,6)| 4714               |
| KNN         | 0.03350        | RF          | 7250               |
| ANN 1       | 0.03414        | ANN 2       | 25501              |
| ANN 2       | 0.03841        | ANN 1       | 56549              |

Figure 25: Graphic presentation of data from Table 2. This style of presentation has the advantage that it is possible to see at a glance which methods are more successful than others. The closer the method is to the lower left corner, the faster and more accurate this method is.

9. Conclusions and future work
From the final summary of the overall results presented in Table 2 and Figures 24 and 25, it is clear that these results divided the prediction methods used into two groups.

The first group includes more accurate methods such as extreme gradient boosting, random forest and both ARIMAs, and the second group includes less accurate methods such as the zeroth algorithm, \( k \)-nearest neighbors and both artificial neural network variants.

This can be clearly seen in Figure 25, where all the methods from the first group are at the bottom of the figure, while the methods from the second group are scattered in the upper half.

In terms of speed, the zeroth algorithm and ARIMA\((0,1,2)\) proved to be the best methods. On the other hand, despite the modification, both artificial neural network variants lag significantly behind the other methods in this respect. Ultimately, the time needed to sufficiently teach an artificial neural network the previous course of the time series is too long. In general, all machine learning methods need some time to train their model, however, here the artificial neural networks take significantly longer than the other methods used. Therefore, this machine learning method is not suitable for predicting the development of supercomputer power.
consumption time series or similar.

Overall, ARIMA(0,1,2) appears to be the most successful method in this case. Interestingly, its variant ARIMA(8,1,6), which is only slightly more accurate, is almost 10 times slower. For all prediction methods, when a good accuracy is achieved, each further refinement is paid for dearly by a significant extension of the computational time.

For future work, it would be interesting to create a new prediction method that would combine the principles of several existing methods so that it will be fast and at the same time sufficiently accurate. If such a method were to be developed, it could become a serious competitor to the prediction methods used in this paper.

Acknowledgments
This work was supported by The Ministry of Education, Youth and Sports from the Large Infrastructures for Research, Experimental Development, and Innovations project “e-INFRA CZ – LM2018140” and by SGC grant No. SP2020/137 “Dynamic system theory and its application in engineering”, VSB - Technical University of Ostrava, Czech Republic.

References
[1] Schuster H G and Just W 2005 Deterministic Chaos: An Introduction Wiley-VCH
[2] IT4Innovations 2021 Anselm, Salomon, DGX-2, and Barbora supercomputer clusters located at IT4Innovations, National Supercomputing Center VSB - Technical University of Ostrava, Czech Republic https://www.it4i.cz/en (accessed 6 Sep 2021)
[3] Tomčala J 2020 Predictability and Entropy of Supercomputer Infrastructure Consumption Chaos and Complex Systems, Springer Proceedings in Complexity ed Stavrinides S and Ozer M (Springer, Cham) pp 59–66
[4] García S and Luengo J and Herrera F 2014 Data Preprocessing in Data Mining Springer
[5] Dickey D A and Fuller W A 1979 Distribution of the Estimators for Autoregressive Time Series with a Unit Root J. Am. Stat. Assoc. 74 pp 427–431
[6] Phillips P C B and Perron P 1988 Testing for a Unit Root in Time Series Regression Biometrika 75 (2) pp 335–346
[7] R Core Team 2021 R: A Language and Environment for Statistical Computing R Foundation for Statistical Computing Available online on https://www.R-project.org (accessed 6 Sep 2021)
[8] Trapletti A and Hornik K 2021 tseries: Time Series Analysis and Computational Finance R package Available online on https://CRAN.R-project.org/package=tseries (accessed 6 Sep 2021)
[9] Hyndman R and Athanasopoulos G and Bergmeir C and Caceres G and Chhay L and O’Hara-Wild M and Petropoulos F and Razbash S and Wang E and Yasmeen F 2021 forecast: Forecasting functions for time series and linear models R package Available online on https://pkg.robjhyndman.com/forecast (accessed 6 Sep 2021)
[10] Brockwell P J and Davis R A 1996 Introduction to time series and forecasting Springer
[11] Kantz H and Schreiber T 2003 Nonlinear Time Series Analysis Cambridge University Press
[12] Kuhn M 2021 caret: Classification and Regression Training R package Available online on https://CRAN.R-project.org/package=caret (accessed 6 Sep 2021)
[13] Ho T K 1995 Random decision forests KDD ’16: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining pp 278–282
[14] Graupe D 2019 Principles Of Artificial Neural Networks: Basic Designs To Deep Learning World Scientific
[15] Chen T and Guestrin C 2016 XGBoost: A Scalable Tree Boosting System Proceedings of 3rd International Conference on Document Analysis and Recognition pp 785–794