Predicting Ions Concentration in Water Streams

Sinsoillier Mike Junior, Du Couédic De Kergoualer Sophie Zhuo Ran, Berezantev Mihaela
Department of Computer Science, EPFL, Switzerland

***Abstract*—This paper presents two models for the prediction of ion concentrations in water streams using quasi-continuous, low-cost data measures from in-situ probes. The aim of this paper is to show the complete procedure needed to obtain the best possible results for this gap-filling task. This project was supervised by Paolo Benettin from the Laboratory of ecohydrology - ECHO at EPFL.**

I. Introduction

The quality of water highly depends on its ionic composition (e.g. Ca, SO_4, NO_3, etc.). In order to be able to analyse it, researchers need high-frequency samples which are often too expensive to obtain. There are, however, other water characteristics that can be measured at high-frequency and at a reasonable cost. The latter can be used to infer the ion concentrations. Table I presents these components that will be used as features and Table II displays the output values.

Feature	Unit
precipitation	$mm/10min$
water_temperature	$°C$
water_electrical_conductivity	$\mu S/cm$
flow	$mm/10min$
turbidity	$FTUeq$
NO3-Neq	mg/l
TOCeq	mg/l
DOCeq	mg/l
dissolved_Oxygen	ppm
temperature_DO	$°C$
conductivity	$\mu S/cm$
temperature_EC	$°C$
pH	$-$
ORP	mV
dQ/dt	C/s

TABLE I
FEATURES THAT INFLUENCE THE ION CONCENTRATION IN WATER STREAMS

Label	Unit
Calcium - Ca	mg/l
Sulfate - SO_4	mg/l
Nitrate - NO_3	mg/l
Magnesium - Mg	mg/l
Chloride - Cl	mg/l
Sodium - Na	mg/l
Potassium - K	mg/l

TABLE II
IONS MEASURED THROUGH ION CHROMATOGRAPHY THAT INFLUENCE THE QUALITY OF WATER STREAMS

II. Data Pre-processing

The data set was cleaned and different feature transformations methods were applied before feeding the data to the models.

A. Data Cleaning

A first step is to ensure that the data is coherent and the values plausible. In particular, the removal of extremely high or low values is crucial, as those values can highly bias the predictions.

A preliminary exploration of the data revealed the presence of duplicated and conflicting elements which were removed. Some outliers were also detected and eliminated, the corresponding values were twenty times higher than the other values and also not plausible from a chemical point of view.

B. Feature creation

The various elements measured for the data set are naturally dependent on weather and seasons, and therefore some periodicity was expected to be found in the data. We employed fast Fourier transform (FFT) to visualize the different frequencies of each element. For most of them, the analysis revealed a yearly or a daily periodicity. Figure 1 illustrates the fast Fourier transform of the water temperature which presented both periodicities.

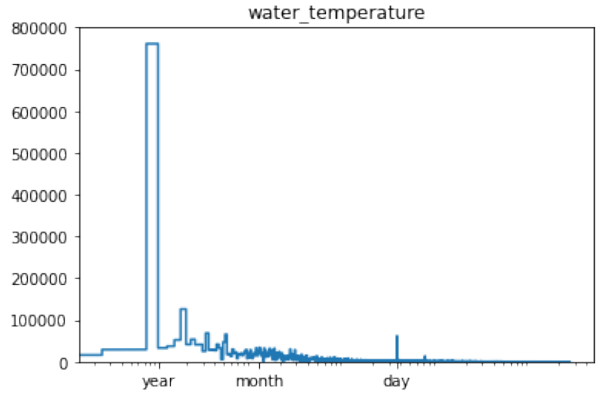

Fig. 1. fast Fourier transform of water temperature

Based on these results, we created two additional features to represent the time of the year and the time of the day at which each sample was taken. To do so, the initial time of each sample is transformed into seconds then divided by the number of seconds in a day and in a year respectively and finally applied a sinus function.

$$t_{day} = \sin\left(\frac{seconds \cdot 2\pi}{24 \cdot 60 \cdot 60}\right) \quad (1)$$

$$t_{year} = \sin\left(\frac{seconds \cdot 2\pi}{24 \cdot 60 \cdot 60 \cdot 365}\right) \quad (2)$$

C. Transformation of heavy-tailed features

The distribution of the following features were highly skewed: water temperature, turbidity, TOCeq. To address this issue, a general solution is the log-transformation:

$$feature' = \log_{10}(feature) \tag{3}$$

For the flow, the log-transformation was not sufficient and we additionally applied a square root transformation:

$$flow' = \log_{10}(100 \cdot \sqrt{flow}) \tag{4}$$

D. Standardization

Machine learning models tend to perform better when the data is in a uniform format. We therefore standardized the features to have zero mean and a standard deviation of 1.

III. Models

The continuous data available suggested that we were up to a regression task. Therefore, we have first considered a Gradient Boosting Regressor because they are easily trained and perform well. Also, the two available data sets have a relatively high number of samples (see Table III), so a Neural Network has also been implemented.

Data set	Sample Frequency	Number of samples
ion concentration	every hour	19'050
probe data	every 10 min	210'085

TABLE III
Available datasets

A. Gradient Boosting Regressor

A Gradient Boosting Regressor combines multiple weak learners, in this case decision trees, to create a model with a lower error than individual learners. It does so by boosting, i.e. at each step, it uses information from the previously built predictors to improve the model's performance. The loss function to be optimized is Mean Squared Error: a differentiable function that is a good fit for a regression task.

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y_i})^2 \tag{5}$$

The whole process can be divided in the following steps.

- Make an intial prediction with

$$argmin_{\hat{y_i}} \sum_{i=1}^{N} L(y_i, \hat{y_i}) \tag{6}$$

  This can be solved by gradient descent.
- Next, we calculate the error of each sample by using the previously built learner. Below is derivative of the loss function with respect to the predicted value.

$$R_{i,m} = -[\frac{\delta L(y_i, F(x_i))}{\delta F(x_i)}]_{F(x)=F_{m-1}(x)} \tag{7}$$

Ion	learning_rate	n_estimators
Ca	0.25	1000
SO_4	0.1	1000
NO_3	0.25	1000
Mg	0.25	1000
Cl	0.25	1000
Na	0.25	1000
K	0.25	1000

TABLE IV
Hyper-parameter Selection

  where i = 1, ... , N, m is the current tree, and the subscript denotes that the values of the previous tree are used to predict the errors of the current tree.
- The next step is to fit a decision tree on the errors instead of the target variable because GBR uses previously built learners to optimize the algorithm at each step.
- The output value for each leaf is computed.
- Using the initial predictions and all built trees we make new predictions for all the samples.

$$F_m(x) = F_{m-1}(x) + \alpha * \gamma_{j,m} \tag{8}$$

  where α is the learning_rate and $\gamma_{j,m}$ is the prediction of the sample's leaf in the current tree.
- Repeat the previous 3 steps until the desired number of trees are built or until adding new trees does not improve the fit.

1) Model Implementation

We will use the Scikit-learn Python library which implements a class named Gradient Boosting Regressor. A different model for each ion is needed and each of them has different hyper-parameters:

- *learning_rate*: size of the gradient steps when fitting the regression trees. It shrinks the contribution of each tree by learning_rate.
- *n_estimators*: the number of boosting staged to perform, i.e. the number of regression trees.

2) Hyper-parameter Selection

We used a 10-fold cross validation with mean RMSEs in order to select the best suited hyper-parameters of each model. The values considered were [125, 250, 500, 1000] for the n_estimators and [1, 0.5, 0.25, 0.1, 0.05] for the learning_rate. The choices can found in Table IV.

B. Recursive Neural Network

Recurrent neural networks, also known as RNNs, are a class of neural networks that allow previous outputs to be used as inputs while having hidden states.There exist many architectures of RNNs but the one of interest for us is the Long Short-Term Memory (LSTM) architecture and especially vanilla LSTM which is the most popular.

1) Vanilla Long Short-Term Memory

A vanilla LSTM unit is composed of a cell, an input gate, an output gate and a forget gate. The cell remembers values over arbitrary time intervals and the three gates regulate the

flow of information associated with the cell. Fig. 2 displays the architecture of a vanilla LSTM block, which involves the gates, the input signal $x^{(t)}$, the output $y^{(t)}$, the activation functions and the peephole connections which enable to let the gate layers look at the cell state. The output of the block is recurrently connected back to the block input and all of the gates.

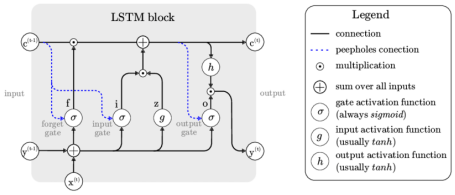

Fig. 2. Architecture of a typical vanilla LSTM block

Aiming to clarify how the LSTM model works, let us assume a network comprised of N processing blocks and M inputs. The forward pass in this recurrent neural system is described below.

Block Input. This step is devoted to updating the block input component, which combines the current input $x^{(t)}$ and the output of that LSTM unit $y^{(t-1)}$ in the last iteration. This can be done as depicted below:

$$z^{(t)} = g(W_z x^{(t)} + R_z y^{(t-1)} + b_z) \tag{9}$$

where W_z and R_z are the weights associated with $x^{(t)}$ and $y^{(t-1)}$, respectively, while b_z stands for bias weight vector.

Input gate. In this step, we update the input gate that combines the current input $x^{(t)}$, the output of that LSTM unit $y^{(t-1)}$ and the cell value $c^{(t-1)}$ in the last iteration. The following equation shows this procedure:

$$i^{(t)} = \sigma(W_i x^{(t)} + R_i y^{(t-1)} + p_i \odot c^{(t-1)} + b_i) \tag{10}$$

where $\odot$ denotes point-wise multiplication of two vectors, W_i, R_i and p_i are the weights associated with $x^{(t)}$, $y^{(t-1)}$ and $c^{(t-1)}$, respectively, while b_i represents for the bias vector associated with this component.

In the previous steps, the LSTM layer determines which information should be retained in the network's cell states $c^{(t)}$. This included the selection of the candidate values $z^{(t)}$ that could potentially be added to the cell states, and the activation values $i(t)$ of the input gates.

Forget gate. In this step, the LSTM unit determines which information should be removed from its previous cell states $c^{(t-1)}$. Therefore, the activation values $f^{(t)}$ of the forget gates at time step t are calculated based on the current input $x^{(t)}$, the outputs $y^{(t1)}$ and the state $c^{(t1)}$ of the memory cells at the previous time step $t-1$, the peephole connections, and the bias terms b_f of the forget gates. This can be done as follows:

$$f^{(t)} = \sigma(W_f x^{(t)} + R_f y^{(t-1)} + p_f \odot c^{(t-1)} + b_f) \tag{11}$$

where W_f, R_f and p_f are the weights associated with $x^{(t)}$, $y^{(t-1)}$ and $c^{(t-1)}$, respectively, while b_f represents for the bias vector associated with this component.

Cell. This step computes the cell value, which combines the block input $z^{(t)}$, $i^{(t)}$ and forget gate $f^{(t)}$ values, with the previous cell value. This can be done as depicted below:

$$c^{(t)} = z^{(t)} \odot i^{(t)} + z^{(t-1)} \odot f^{(t)} \tag{12}$$

Output gate. This step calculates the output gate, which combines the current input $x^{(t)}$, the output of that LSTM unit $y^{(t-1)}$ and the cell value $c^{(t-1)}$ in the last iteration. This can be done as depicted below:

$$o^{(t)} = \sigma(W_o x^{(t)} + R_o y^{(t-1)} + p_o \odot c^{(t-1)} + b_o) \tag{13}$$

where W_o, R_o and p_o are the weights associated with $x^{(t)}$, $y^{(t-1)}$ and $c^{(t-1)}$, respectively, while b_o represents for the bias vector associated with this component.

Block output. Finally, we calculate the block output, which combines the current cell value $c^{(t)}$ with the current output gate value as follows:

$$y^{(t)} = g(c^{(t)}) \odot o^{(t)} \tag{14}$$

In the above steps, σ,g and h denote point-wise non-linear activation functions. The logistic sigmoid $\sigma(x) = \frac{1}{1+e^{1-x}}$ is used as a gate activation function, while the hyperbolic tangent $g(x) = h(x) = \tanh(x)$ is often used as the block input and output activation function.

2) Training of the model

Backpropagation Through Time is used to compute the weights that connect the different components in the network. Therefore, during the backward pass, the cell state $c^{(t)}$ receives gradients from $y^{(t)}$ as well as the next cell state $c^{(t+1)}$.Those gradients are accumulated before being backpropagated to the current layer.

In the last iteration T, the change $\delta_y^{(T)}$ with the network error $\partial E/y^{(T)}$ such that E does the loss function. Otherwise, $\delta_y^{(t)}$ is the is the vector of delta values passed down $\Delta^{(t)}$ from the above layer including the recurrent dependencies. This can be done as follows:

$$\delta_y^{(t)} = \Delta^{(t)} + R_z^T \delta_z^{(t+1)} + R_i^T \delta_i^{(t+1)} + R_f^T \delta_f^{(t+1)} + R_O^T \delta_O^{(t+1)} \tag{15}$$

In a second step, the change in the parameters associated with the gates and the memory cell are calculated as:

IV. Results

A. Gradient Boosting Regressor

Table V shows some measures obtained after training the BR and using it on the testing set.

B. Recursive Neural Network

Table VI shows some measures obtained after training the RNN and using it on the testing set using 50 epochs.

V. Discussion

The recurrent neural network was very promising as it particularly well suited for time series data. However, it didn't achieve the expected performances and was surpassed by the Gradient Boosting Regressor. The RNNs are complex networks and a deeper understanding would probably let us to explore new directions and lot of parameters could have been

Ion	RMSE	R^2	Max Error [mg/l]	MAE [mg/l]
Ca	1.518	0.976	21.294	0.920
SO_4	0.667	0.985	8.575	0.441
NO_3	0.104	0.940	1.937	0.053
Mg	0.105	0.979	1.006	0.067
Cl	0.038	0.915	0.679	0.019
Na	0.095	0.988	1.321	0.062
K	0.039	0.953	0.563	0.024

TABLE V
GRADIENT BOOSTING REGRESSOR RESULTS

Ion	RMSE	R^2	Max Error [mg/l]	MAE [mg/l]
Ca	2.954	0.908	27.105	2.028
SO_4	1.338	0.941	10.974	0.887
NO_3	0.200	0.777	1.572	0.137
K	0.066	0.861	0.574	0.045

TABLE VI
RECURENT NEURAL NETWORKS REGRESSOR RESULTS

optimized using cross-validation. Another possible limitation was the limited quantity of ion concentration data. Despite the large size of the probe dataset which allows to make a high number of predictions, the ion concentration data represented a bottleneck for the training ouf our models.

literature

REFERENCES

[Neural Networs] Van Houdt, Greg Mosquera, Carlos Nápoles, Gonzalo. (2020). A Review on the Long Short-Term Memory Model. Artificial Intelligence Review.

[Boosting Regressor] Demystifying decision trees, random forests gradient boosting. (2021). Retrieved 23 December 2021, from Here

APPENDIX

For each ion we present the most important features for the model (needed for obtaining the ion measures in the future), a representation of the true values and the predicted values on the testing set (all the points should be as close to the diagonal as possible) and finally we plot the true and predicted values for a single week in order to see the difference between them.

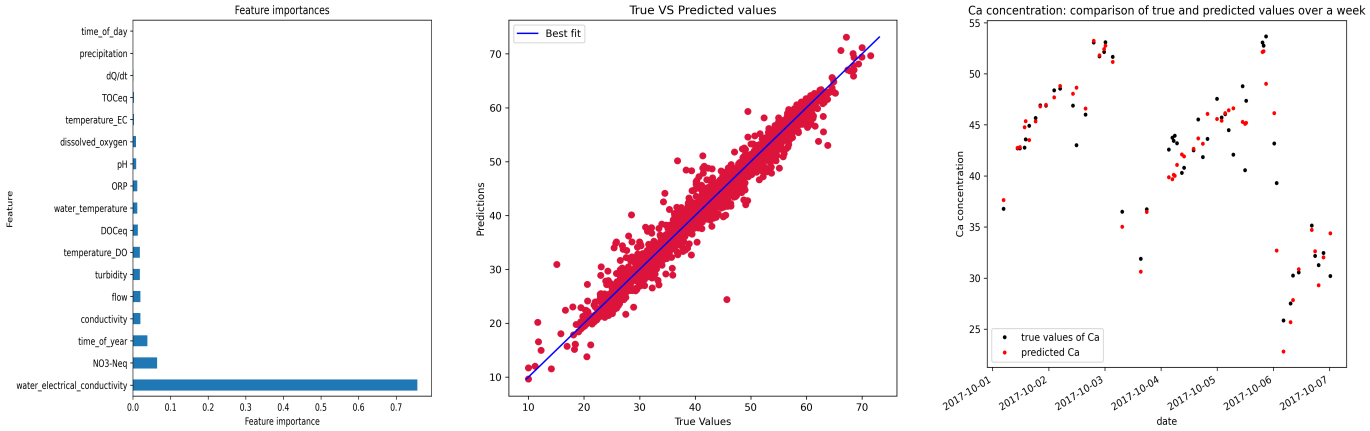

Fig. 3. Calcium

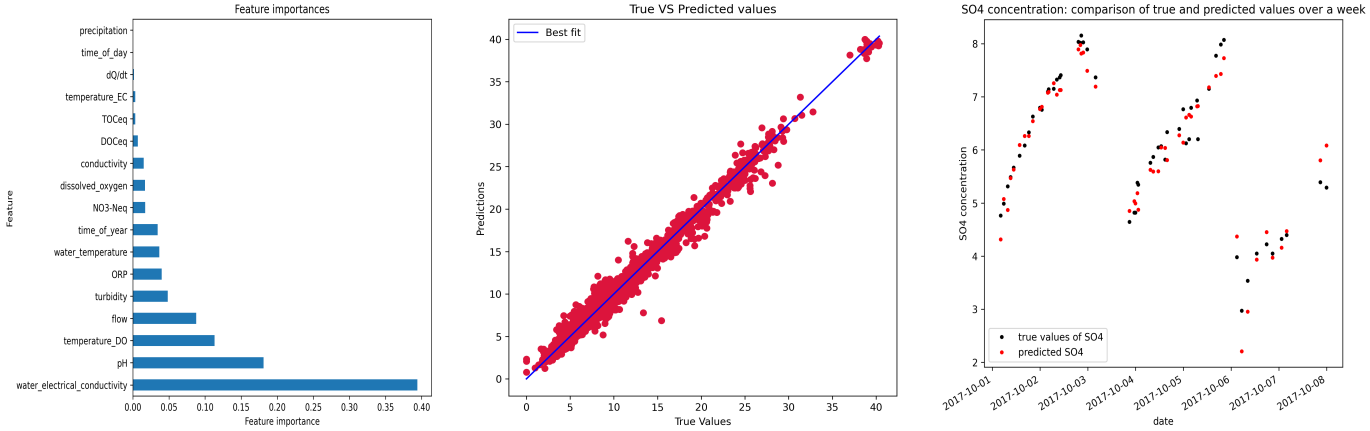

Fig. 4. Sulfate

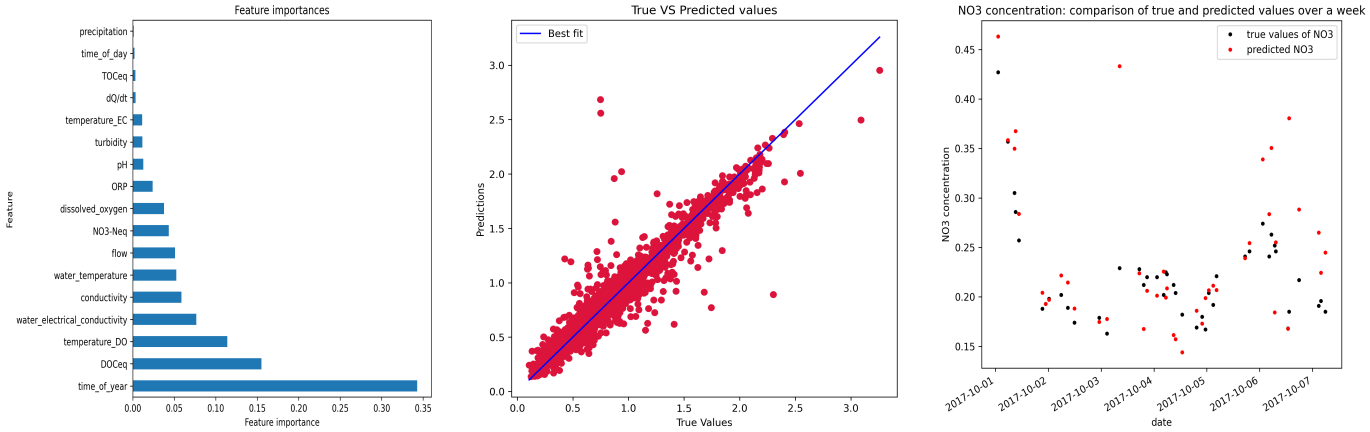

Fig. 5. Nitrate

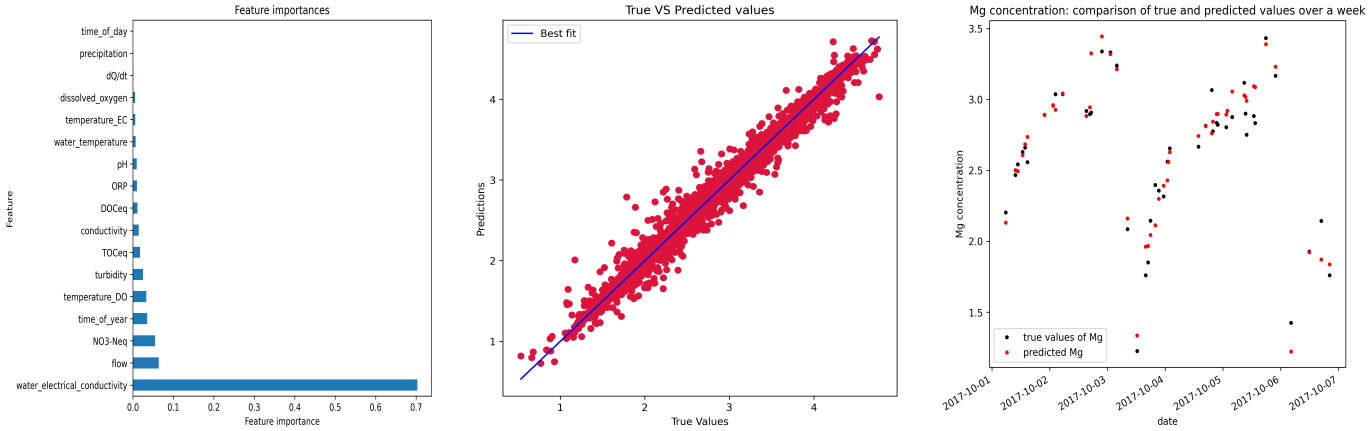

Fig. 6. Magnesium

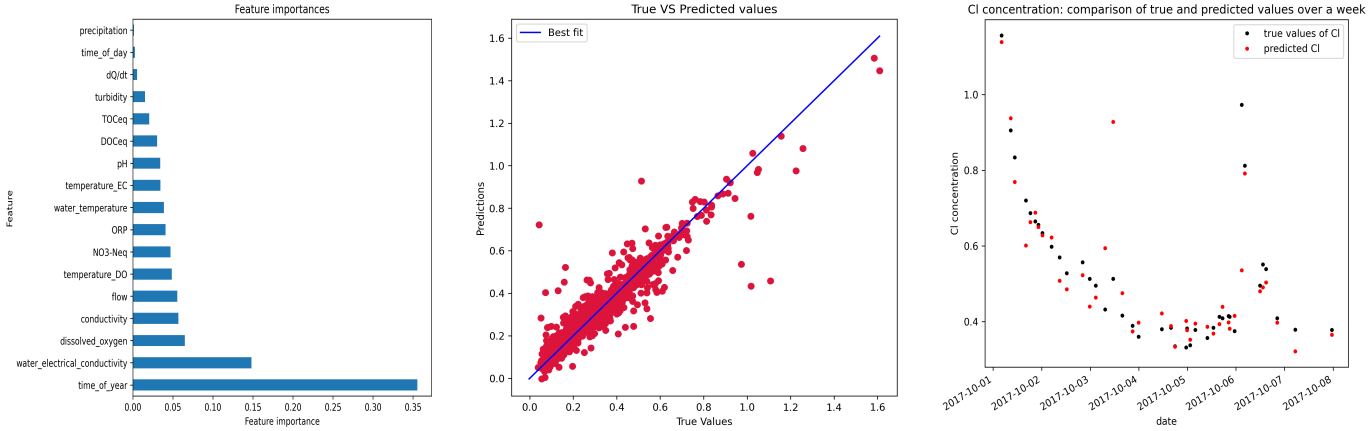

Fig. 7. Chloride

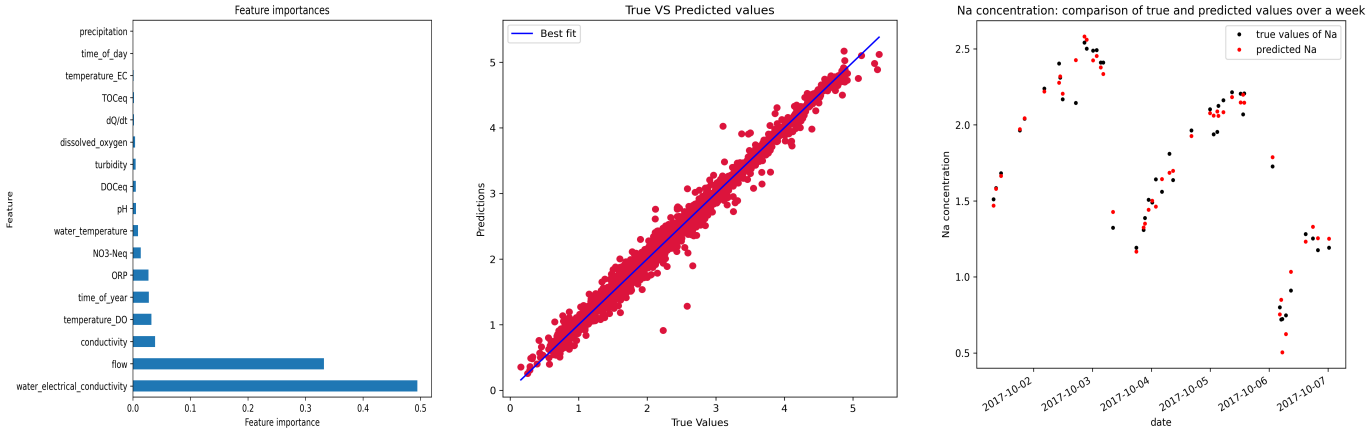

Fig. 8. Sodium

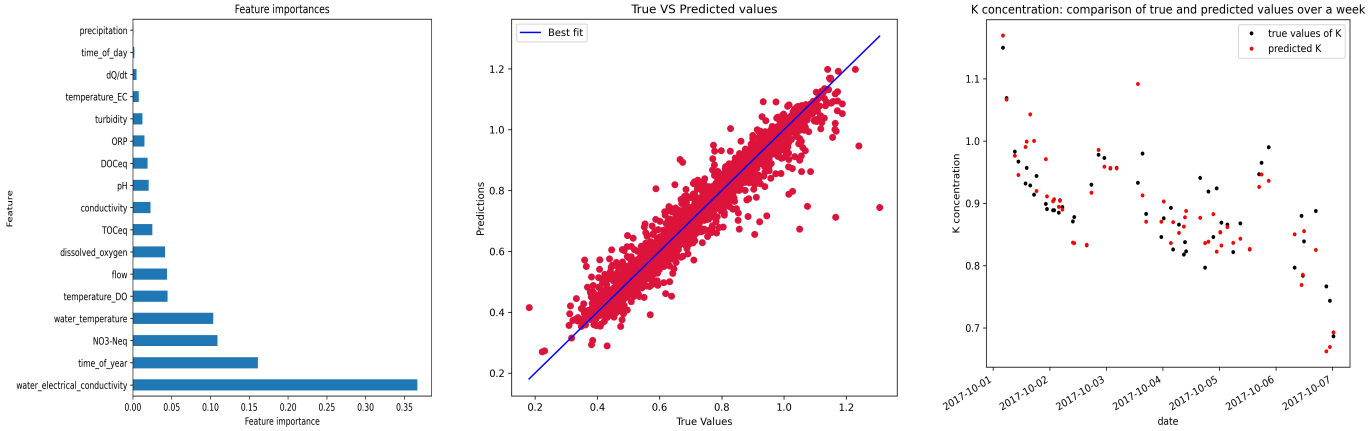

Fig. 9. Potassium