Data curation and analytical framework flow

The developed analytical framework is based on a dataset that included inputs generated, as a result of analytical procedures (real data obtained from *in-situ* sampling and laboratory analysis of both water and sediment samples from the sample points presented in figure 1).

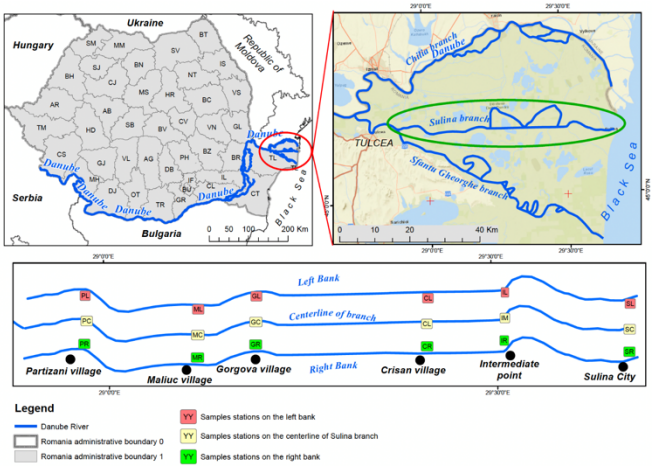

Fig. 1. Water and sediments sample points from the studied area

Therefore, the resulted numerical dataset was used as emphasised in figure 2.

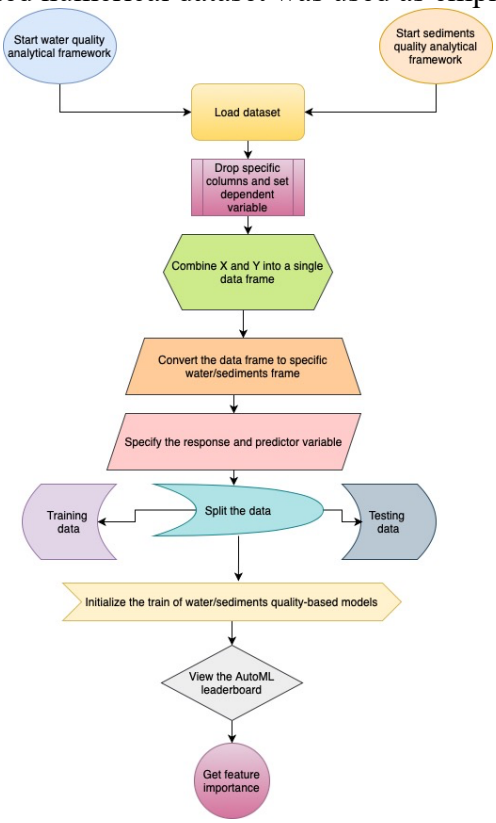

Figure 2. The analytical framework flow

It can be observed that the autoML script was used in order to identify various ML-based algorithms' performance (revealed by the accuracy metrics) in predicting the feature importance of the independent variables. Various scenarios were used, targeting to create the

baseline for determining Fe, Zn, Cd, Cr, and Ni from both water and sediments, in the studied area. The results have targeted to identify the main groups of ML algorithms, that gather different secondary algorithms, characterised by the variations of hyperparameters, that are associated with the highest predictive metrics. Also, for the identified main ML algorithms, an average value of feature importance associated with each of the independent variables will result and, further on, emphasize the power of the predictors in predicting the main dependent variable.

Main ML algorithms used for training and validation

The following main ML algorithms were used:

A. *Decision Random Forest (DRF) Machine Learning Algorithm* **-** an ensemble learning method that constructs multiple decision trees during training and outputs the mean prediction (regression) or mode of the classes (classification) of the individual trees. This method is widely used due to its robustness, accuracy, and ease of use. It is a versatile algorithm that can handle both classification and regression tasks effectively. DRF leverages the power of multiple decision trees to improve the model's performance. Each tree is trained on a random subset of the data, ensuring diversity and reducing the risk of overfitting. By combining the predictions from multiple trees, the algorithm achieves higher accuracy and stability compared to individual decision trees. The algorithm uses bagging, where each decision tree is trained on a bootstrapped sample of the training data. This technique helps in reducing variance and improving the stability and accuracy of the model. Bagging ensures that each tree is different, which enhances the overall performance of the ensemble. During the construction of each tree, DRF randomly selects a subset of features to consider for splitting at each node. This randomness helps in creating a diverse set of trees, further enhancing the model's robustness. Random feature selection prevents the model from relying too heavily on any single feature, improving generalization. In classification tasks, DRF uses majority voting to determine the final output. For regression tasks, it averages the predictions of individual trees. This ensemble approach mitigates the impact of noisy or inaccurate individual trees, leading to more reliable predictions. By combining multiple trees that are trained on different parts of the dataset with different features, DRF effectively reduces overfitting, leading to a more generalized model. The use of ensemble methods and randomization techniques makes DRF robust against overfitting, even with complex datasets. DRF can provide insights into feature importance by measuring how much each feature decreases the impurity in a tree. This can be valuable for understanding the underlying patterns in the data and for feature selection. Feature importance scores help in identifying the most influential features in the dataset, guiding further data analysis and model improvement.

Several hyperparameters can be tuned to optimize the performance of the DRF algorithm:

- Number of Trees (n_estimators): The number of trees in the forest. More trees can improve performance but also increase computational cost.
- Maximum Depth (max_depth): The maximum depth of each tree. Limiting depth helps prevent overfitting.
- Minimum Samples Split (min_samples_split): The minimum number of samples required to split an internal node. Higher values prevent overfitting.
- Minimum Samples Leaf (min_samples_leaf): The minimum number of samples required to be at a leaf node. Setting this parameter can smooth the model.
- Maximum Features (max_features): The number of features to consider when looking for the best split. It introduces randomness to make individual trees more diverse.
- Bootstrap: Whether bootstrap samples are used when building trees. If False, the whole dataset is used to build each tree.

The DRF algorithm is a powerful tool in machine learning, capable of handling both classification and regression tasks effectively. Its ensemble nature, combined with techniques like bagging and random feature selection, makes it robust against overfitting and highly

accurate. By tuning various hyperparameters, the performance of the DRF can be optimized for specific datasets and tasks, making it a versatile choice for many applications.

B. *Deep Learning Algorithm* **-** uses neural networks with many layers (hence the term 'deep'). These algorithms are designed to recognize patterns, learn from data, and make decisions with minimal human intervention. Deep learning has revolutionized fields such as computer vision, natural language processing, and speech recognition. Its ability to process and learn from large datasets has led to significant advancements in AI capabilities. At the core of deep learning are neural networks, which are inspired by the structure and function of the human brain. A neural network consists of an input layer, multiple hidden layers, and an output layer. Each layer contains neurons (nodes) that are connected with weights, allowing the network to learn complex representations of data. The architecture of these networks enables them to capture hierarchical patterns, making them effective for a wide range of tasks.The training process involves feeding data into the neural network, where it undergoes forward propagation and backpropagation. In forward propagation, the input data passes through the network, and predictions are made. During backpropagation, the network adjusts the weights based on the error of the predictions, iteratively improving its accuracy. This process is repeated over many iterations, or epochs, to minimize the loss function and enhance the model's performance. Activation functions introduce non-linearity into the neural network, enabling it to learn and represent complex patterns. Common activation functions include ReLU (Rectified Linear Unit), sigmoid, and tanh. These functions help the network capture intricate relationships within the data, enhancing its predictive power. Choosing the right activation function is crucial for the network's ability to converge and learn effectively.

There are various types of neural networks, each suited for different tasks. Convolutional Neural Networks (CNNs) are primarily used for image recognition and processing. Recurrent Neural Networks (RNNs) are effective for sequential data, such as time series or natural language. Other types include Generative Adversarial Networks (GANs) for generating new data and Transformer networks for handling sequential data efficiently.

Deep learning models are prone to overfitting due to their high capacity to learn from data. Techniques such as dropout, regularization, and early stopping are used to prevent overfitting. These methods help the model generalize better to new, unseen data, improving its performance. Implementing these techniques effectively can significantly enhance the model's ability to perform well on real-world data.

Transfer learning involves using a pre-trained model on a new, but related task. This approach leverages the knowledge gained from the original task, requiring less data and computation for the new task. Transfer learning is particularly useful when there is limited data available for training a deep learning model. It allows for faster development and deployment of models by building on existing, well-trained architectures.

Several hyperparameters can be tuned to optimize the performance of deep learning models:

- Learning Rate: Controls the step size during gradient descent. A smaller learning rate ensures convergence but requires more time.
- Batch Size: The number of training examples used in one iteration. Smaller batch sizes provide more updates but are noisier.
- Number of Epochs: The number of times the entire training dataset passes through the network. More epochs can improve learning but increase the risk of overfitting.
- Dropout Rate: The fraction of neurons randomly dropped during training to prevent overfitting.
- Number of Layers and Neurons: The depth and width of the neural network. More layers and neurons can capture complex patterns but increase computational cost.

Deep learning algorithms have transformed the landscape of artificial intelligence, enabling breakthroughs in various domains. By leveraging large datasets and powerful computational resources, deep learning models can achieve remarkable accuracy and efficiency. Tuning hyperparameters and employing techniques to prevent overfitting are crucial for optimizing the performance of deep learning models. As the field continues to evolve, deep learning remains at the forefront of AI research and application, driving innovation and discovery.

C. *Gradient Boosting Machine (GBM) Algorithm* - builds an ensemble of weak prediction models, typically decision trees, in a sequential manner. Each subsequent model attempts to correct the errors made by the previous models, resulting in a highly accurate predictive model. GBM leverages the power of ensemble learning by combining multiple weak learners to form a strong learner. Each weak learner is a simple model, such as a decision tree with limited depth. By sequentially adding these weak learners, GBM improves the overall model performance significantly. The training process in GBM is sequential, where each model is trained to correct the errors of the preceding models. This is achieved by fitting the new model to the residuals (errors) of the previous model. As a result, each subsequent model focuses on the areas where the previous models performed poorly, enhancing the overall accuracy.

GBM optimizes a specified loss function, such as mean squared error for regression or log-loss for classification. The algorithm minimizes this loss function by adding models that reduce the residuals of the previous models. This iterative process continues until the model performance converges or reaches a predefined number of iterations.

The learning rate, also known as shrinkage, controls the contribution of each weak learner to the final model. A lower learning rate requires more trees to achieve the same reduction in error, but it often results in better generalization. Balancing the learning rate and the number of trees is crucial for optimizing the model's performance.

Regularization techniques are employed in GBM to prevent overfitting and improve model generalization. Common regularization methods include limiting the depth of the trees, adding a penalty for the number of leaf nodes, and using subsampling. These techniques help in maintaining a balance between model complexity and predictive accuracy.

GBM provides insights into feature importance, which helps in understanding the contribution of each feature to the model. Feature importance is determined by the reduction in the loss function attributed to each feature. This information can be valuable for feature selection and for gaining insights into the underlying data patterns.

Several hyperparameters can be tuned to optimize the performance of GBM models:
- Number of Trees (n_estimators): The number of boosting stages to be run. More trees can improve accuracy but also increase computational cost.
- Learning Rate: Controls the step size during each iteration. A smaller learning rate requires more trees but can improve generalization.
- Maximum Depth (max_depth): The maximum depth of the individual trees. Limiting depth helps prevent overfitting.
- Minimum Samples Split (min_samples_split): The minimum number of samples required to split an internal node. Higher values prevent overfitting.
- Minimum Samples Leaf (min_samples_leaf): The minimum number of samples required to be at a leaf node. It helps in smoothing the model.
- Subsample: The fraction of samples used for fitting the individual base learners. Using a value less than 1.0 can prevent overfitting.
- Loss Function: The specific loss function to be optimized, such as mean squared error for regression or log-loss for classification.

Gradient Boosting Machine is a robust and versatile algorithm widely used for various predictive tasks. Its ability to build an ensemble of weak learners sequentially, focusing on

correcting errors, leads to high accuracy. By tuning hyperparameters and employing regularization techniques, GBM models can be optimized for specific datasets and tasks, making them a powerful tool in the machine learning toolkit.

D. Stacked Ensemble Machine Learning Algorithm - combines multiple models, or base learners, to produce a stronger predictive model. The idea is to leverage the strengths of different models to improve overall performance and accuracy. Stacked ensembles are particularly useful for complex tasks where single models may not perform optimally.

Stacked Ensemble method uses base learners, respectively individual models that are trained on the same dataset. These models can be of different types, such as decision trees, support vector machines, and neural networks. By combining diverse models, stacked ensembles can capture a wide range of patterns and relationships in the data.

The meta-learner, or second-level model, is trained on the predictions of the base learners. Its role is to learn how to best combine the outputs of the base learners to make the final prediction. The meta-learner can be a simple model, such as linear regression, or a more complex one, like a neural network.

Cross-validation is a critical component of building stacked ensembles. It involves partitioning the data into multiple folds and training the base learners on different subsets. This ensures that the meta-learner is trained on out-of-sample predictions, reducing the risk of overfitting.

Diversity among base learners is key to the success of stacked ensembles. By using models that make different types of errors, the ensemble can reduce overall prediction error. Ensuring diversity can be achieved by varying the algorithms, hyperparameters, and training data used for the base learners.

Stacked ensembles often outperform individual models because they aggregate the strengths of multiple models. They can handle a wider range of data complexities and improve generalization. This technique is particularly effective in competitions and real-world applications where predictive accuracy is paramount.

Implementing stacked ensembles can be more complex compared to single models. It requires careful tuning of base learners, meta-learner, and the cross-validation strategy. Despite the complexity, the potential gains in performance often justify the additional effort.

Several hyperparameters can be tuned to optimize the performance of stacked ensemble models:

- Number and Type of Base Learners: Deciding how many and which types of models to include.
- Meta-Learner Choice: Selecting the appropriate meta-learner to combine the predictions of base learners.
- Cross-Validation Strategy: Choosing the number of folds and how to split the data for cross-validation.
- Hyperparameters of Base Learners: Tuning the individual hyperparameters of each base learner to improve their performance.
- Weighting of Base Learners: Deciding whether to weight the predictions of base learners differently based on their performance.

Stacked ensemble algorithms are a robust and versatile tool in the machine learning toolkit. By combining multiple models, they leverage the strengths of each to deliver superior predictive performance. While the implementation can be complex, the potential benefits in terms of accuracy and generalization make stacked ensembles an attractive choice for challenging predictive tasks.

E. *XGBoost Machine Learning Algorithm* – high scalability, speed, and robust handling of large datasets with high dimensionality. XGBoost implements the gradient boosting framework, where models are built sequentially to correct the errors of previous models. This iterative process enhances the overall model performance by focusing on the hardest-to-predict examples. Each new model minimizes the residual errors of the previous models, leading to improved accuracy.

XGBoost incorporates regularization techniques to prevent overfitting, which is a common issue in machine learning models. It includes L1 (Lasso) and L2 (Ridge) regularization terms in the objective function to penalize complex models. This regularization helps in maintaining a balance between model complexity and predictive power.

One of the standout features of XGBoost is its ability to handle missing data effectively. During training, it automatically learns the best direction to handle missing values in the data. This capability makes XGBoost robust and reliable when dealing with real-world datasets that often contain missing values.

XGBoost is designed for scalability and can handle large datasets efficiently. It supports parallel processing, allowing it to utilize multiple CPU cores for faster training. Additionally, XGBoost can be distributed across clusters, making it suitable for big data applications.

Tree pruning in XGBoost is performed using a technique called 'max depth' to limit the depth of trees. This prevents the model from becoming too complex and overfitting the training data. Pruning ensures that the trees remain manageable and improves the model's generalization ability.

XGBoost provides insights into feature importance, helping to understand which features contribute the most to the predictions. Feature importance can be measured using metrics such as gain, cover, and frequency. This information is valuable for feature selection and for gaining deeper insights into the data.

Several hyperparameters can be tuned to optimize the performance of XGBoost models:

- Learning Rate (eta): Controls the contribution of each tree. A lower learning rate requires more trees but can improve generalization.
- Number of Trees (n_estimators): The number of boosting rounds. More trees can enhance accuracy but increase computational cost.
- Maximum Depth (max_depth): The maximum depth of each tree. Deeper trees can capture more information but are prone to overfitting.
- Subsample: The fraction of samples used for training each tree. Lower values prevent overfitting but might increase bias.
- Colsample_bytree: The fraction of features used for training each tree. This adds randomness and can improve model robustness.
- Gamma: Minimum loss reduction required to make a split. Higher values prevent overfitting by making the algorithm more conservative.

XGBoost is a highly efficient and versatile machine learning algorithm widely used for various predictive tasks. Its ability to handle large datasets, incorporate regularization, and provide insights into feature importance makes it a valuable tool. By tuning its hyperparameters and leveraging its scalability, XGBoost can be optimized for specific datasets and applications, ensuring high performance and accuracy.

ML phyton code

The figure 3 presents the phyton main code segments, optimized and used for accomplishing all the analytical framework flow within the present study.

```
import h2o
from h2o.automl import H2OAutoML
import pandas as pd
import csv
import matplotlib.pyplot as plt
from sklearn.metrics import mean_squared_error
import numpy as np

# Start the H2O cluster
h2o.init()

# Load your dataset
file_path = '//Users//dragoscristea//Downloads//sedimente.csv'
data = pd.read_csv(file_path)

# Drop the specified columns and set the dependent variable
data_cleaned = data.drop(columns=['Fe'])
#data_cleaned_fe = data.drop(columns=['Fe', 'Zn', 'Cd', 'Cr', 'Ni'])
X_metal = data_cleaned
y_metal = data['Fe']

# Combine X and y into a single dataframe
data_metal = pd.concat([X_metal, y_metal], axis=1)

# Convert the dataframe to H2OFrame
data_h2o = h2o.H2OFrame(data_metal)

# Specify the response and predictor variables
response = 'Fe'
predictors = data_cleaned.columns.tolist()

# Split the data into training and testing sets
train, test = data_h2o.split_frame(ratios=[0.8], seed=42)

# Initialize and train the H2O AutoML model
aml = H2OAutoML(max_runtime_secs=600, seed=42, verbosity="info")
aml.train(x=predictors, y=response, training_frame=train)

# View the AutoML leaderboard
lb = aml.leaderboard
#print(lb)

# Get feature importance for each model in the leaderboard
for model_id in lb.as_data_frame().model_id:
    model = h2o.get_model(model_id)
    if 'variable_importances' in model._model_json['output']:
        feature_importance = model.varimp(use_pandas=True)
        print(f"Feature importance for model {model_id}:\n", feature_importance)
    else:
        print(f"Model {model_id} does not support feature importance.")
```

Fig. 3. The AutoML process code for determining the dependent variables based on the dataset curation performed as a result of sampling and analytical procedures

ML models accuracy

For predicting Fe concentration in water, all XGBoost, GBM, Stackedensemble (STE) and DRF recorded high metrics (R-sq > 0.9), while DeepLearning (DL) algorithms indicate an R-sq > 0.8 (fig. 4). Thus, Fe has significant potential to be integrated into a black-box ML-based sensor, considering the actual structure of the dataset.

In terms of Zn concentration in water (fig. 5), the ML models' accuracy metrics are lower, compared to Fe, with STE leading, followed by DL and XGBoost, all slightly over the R-sq value of 0.7. Thus, STE is recommended to be considered as a main ML tool for the future optimization of the prediction framework associated with Zn concentration in water matrix.

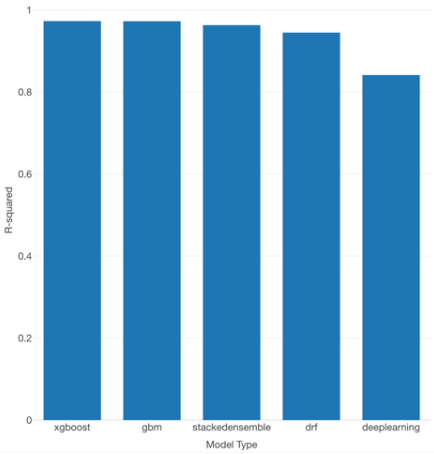

Fig. 4.The accuracy of the main ML algorithms tested for the prediction of Fe concentration in water

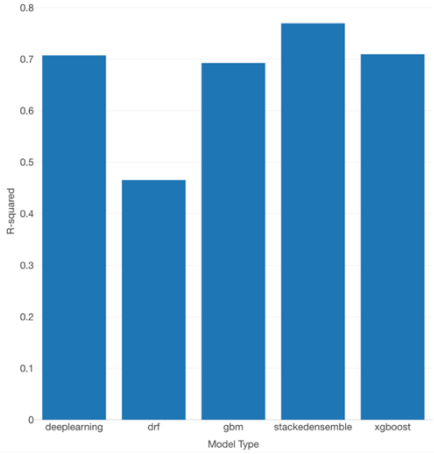

Fig. 5. The accuracy of the main ML algorithms tested for the prediction of Zn concentration in water

The algorithm rating in predicting Cd, Ni and Cr concentration in water is similar to the Zn concluded results, ranking STE in the first place and linking DRF to the lowest accuracy metrics (fig. 6 - 8). Also, in terms of sediments prediction, the situation reveals STE with the highest metrics, followed by DL and XGBoost, with an R-sq just above 0.7. It seems that DRF gives rather modest accuracy, for almost all dependent variables tested in present framework.

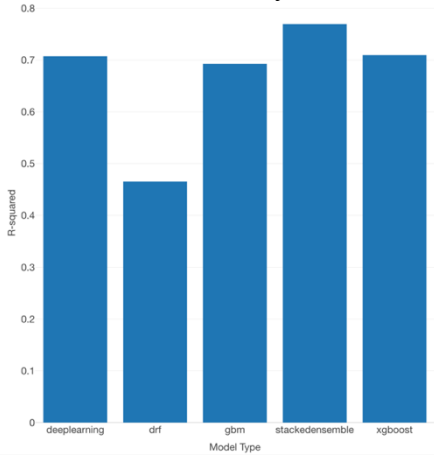

Fig. 6. The accuracy of the main ML algorithms tested for the prediction of Cd concentration in water

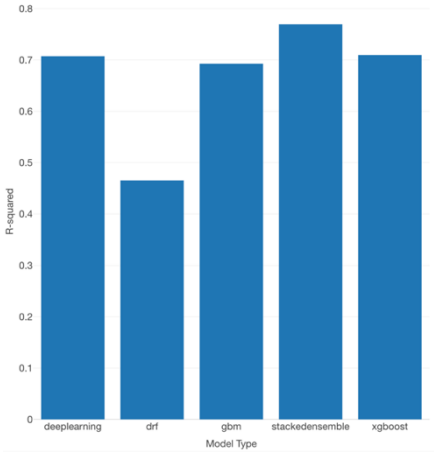

Fig. 7. The accuracy of the main ML algorithms tested for the prediction of Cr concentration in water

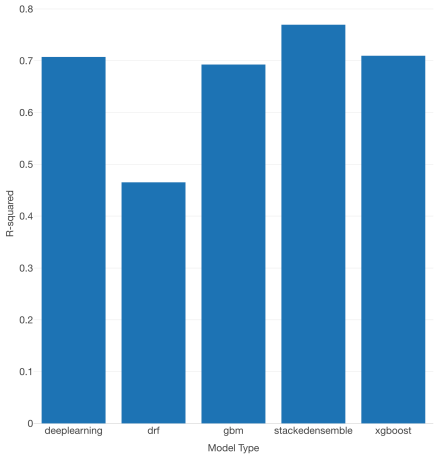

Fig. 8. The accuracy of the main ML algorithms tested for the prediction of Ni concentration in water

ML models feature importance

In terms of the future importance of the predictors used for predicting Fe concentration in the water matrix, it can be observed that BOD5, N-NH_4, NT and pH have the highest impact (fig. 9), a situation that is significantly different compared to the model which quantifies the impact of predictors on Zn concentration in water, influenced mainly by Fe and Cd concentration, as well as by N-NO_2 (fig. 10)

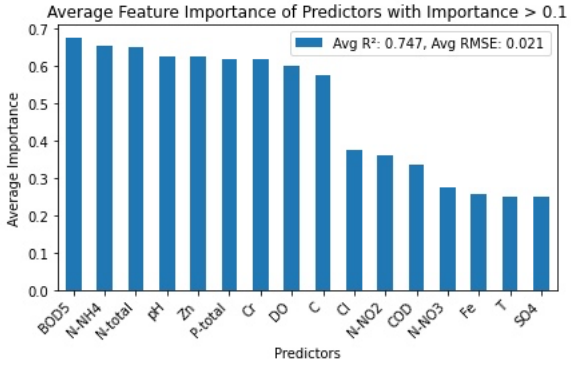

Fig. 9. The average feature importance of each analytical framework predictor, in determining the Fe concentration in water

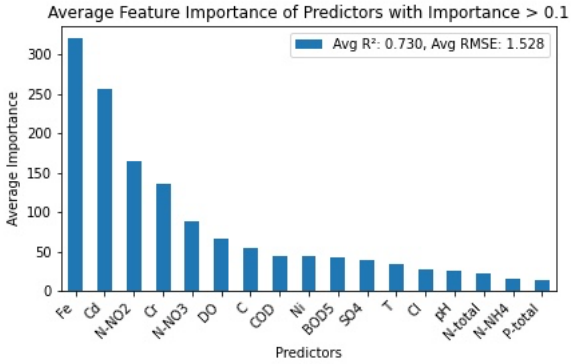

Fig. 10. The average feature importance of each analytical framework predictor, in determining the Zn concentration in water

The Ni concentration is associated with the highest feature importance when it comes to predicting both Cd and Cr concentrations in the water (fig. 11-12). However, in terms of predicting Cd concentration in water, a series of parameters such as NT, Cr, BOD5, N-NH4, pH, DO and Zn must be also considered since they form a complex conditioning matrix (fig. 11). The N-NO_2 and Fe can be rate as secondary predictors, considering their future importance in predicting Cr concentration in water (fig. 12). The prediction model for Ni concentration in water (fig. 13) is mainly conditioned by Fe, Cd and Cr concentration in water column.

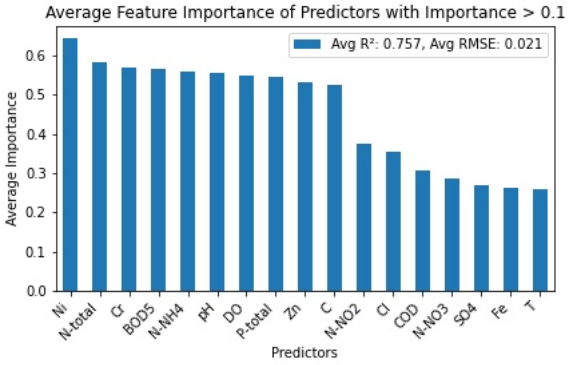

Fig. 11. The average feature importance of each analytical framework predictor, in determining the Cd concentration in water

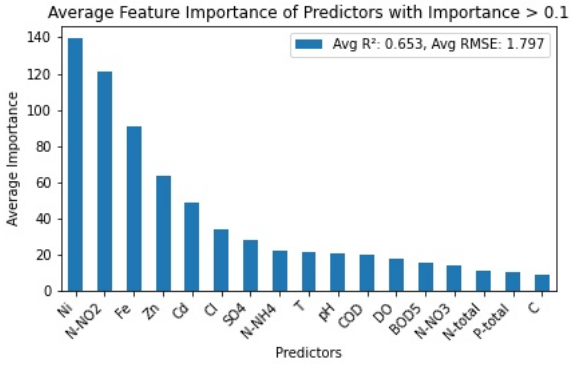

Fig. 12. The average feature importance of each analytical framework predictor, in determining the Cr concentration in water

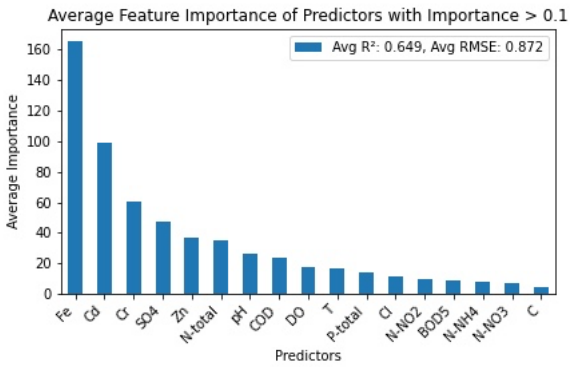

Fig. 13. The average feature importance of each analytical framework predictor, in determining the Ni concentration in water

If considering the resulting sediment prediction models, based on fewer predictors compared to water quality ML models, it can be concluded that Ni concentration in sediments can be predicted mostly based on Zn concentration in sediments (fig. 14), while the prediction of Cr concentration in sediments involves, mostly, the Fe, as a main predictor, and Zn, as a secondary predictor (fig. 15).

The Ni and Cr concentrations in sediments are associated with the highest feature importance in predicting Zn concentration in sediments (fig. 16), while Cr is the main predictor in the prediction model of the Fe concentration in sediments (fig. 17).

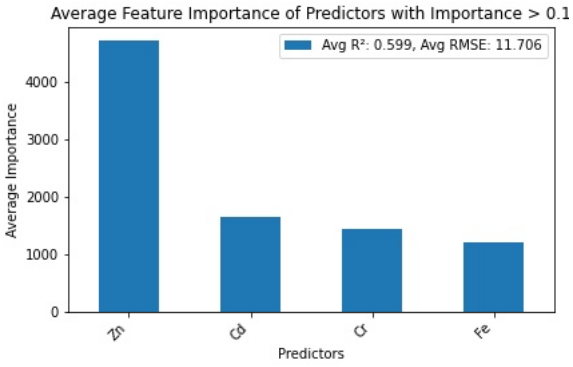

Fig. 14. The average feature importance of each analytical framework predictor, in determining the Ni concentration in sediments

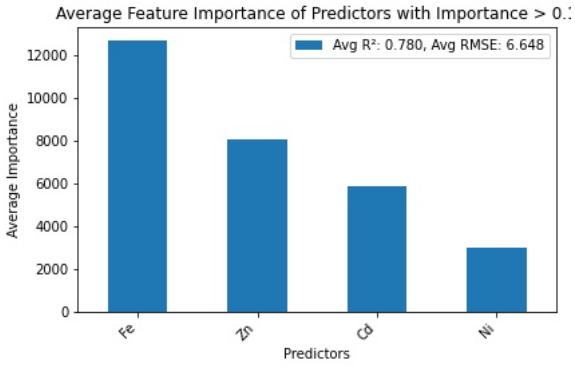

Fig. 15. The average feature importance of each analytical framework predictor, in determining the Cr concentration in sediments

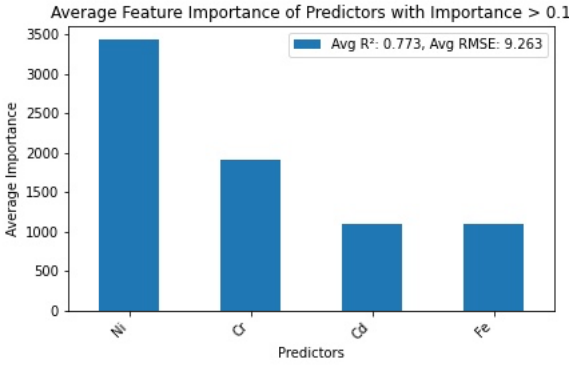

Fig. 16. The average feature importance of each analytical framework predictor, in determining the Zn concentration in sediments

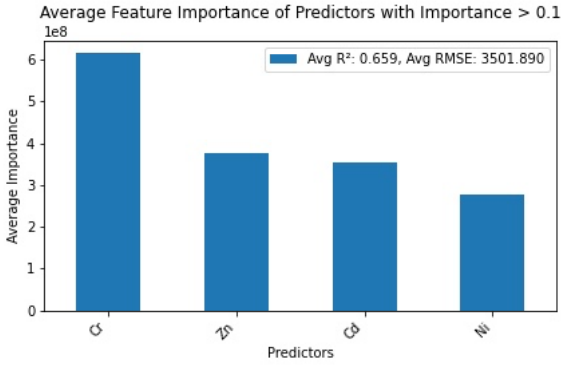

Fig. 17. The average feature importance of each analytical framework predictor, in determining the Fe concentration in sediments

It can be concluded that autoML can offer high efficiency in the design of a machine-learning-based analytical framework that can be used, further on, as a support tool for the development of black-box sensors for the evaluation of water and sediment-quality. Also, in most cases, STE, XGBoost and DL performance was superior, compared to GBM and DRF algorithms. The feature importance value indicates complex and peculiar links between the various predicted dependent variables and the predictors set, emphasizing the importance of the background (e.g. dataset structure, number of inputs, possible environmental disruptive factors etc) in creating a high-accuracy machine-learning-based predictive analytical framework. Future *in-situ* sampling champaigns, followed by analytical procedures, are to be performed in order to develop/extend the existing dataset and to optimize the already obtained models, in order to increase their reliability. Also, various algorithms are to be identified and tested in the near future, in order to verify their potential to generate added value to the already existing analytical framework.