Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Machine Learning Models for Predicting the Leaching of Contaminants from Incineration Bottom Ashes

Master Thesis

Willy M. Schumacher

August 30, 2024

Supervisor: Dr. A. Adelmann,
Collaborator: Dr. R. Boiger

Department of Physics , ETH Zürich

Acknowledgements

I would like to express my sincere gratitude to Dr. Andreas Adelmann for providing me with the opportunity to work on this project. I extend my heartfelt thanks to Dr. Romana Boiger for her unwavering support, valuable input, and for being such a kind and supportive mentor throughout this journey. I am also deeply grateful to Ph.D. Philipp Ingold for his assistance with the data and for sharing his expertise on the background of the project. Additionally, I would like to thank the LES group for their engaging discussions and insightful contributions.

I would also like to acknowledge the developers of Python and the authors of the various Python libraries that were instrumental in this work (see Section 4.1). A special thanks to the creators of the resources in [Aut24b] and [Aut24a], which were invaluable in generating many of the diagrams and figures in this thesis.

Finally, all code used in this project can be found at `https://gitlab.psi.ch/tms_students/willy_column`.

Abstract

Leaching of contaminants in landfills poses a significant environmental concern, as it can lead to soil and groundwater contamination. Therefore, careful monitoring of leaching behavior is crucial for effective landfill management and control. Column leaching tests of bottom ashes are used for evaluating the leaching behaviour at landfills. The tests considered here need about 30 days to estimate long-term leaching behaviour accurately. The lab tests will take longer based on how far into the future leaching behaviour should be estimated.

The goal in this study is to shorten the duration of these lab tests from several weeks to a couple of days by predicting the long term results based on the short term ones. This is done by considering concentrations at low Liquid-to-Solid ratios (LS) based on measurements to estimate the concentrations at higher LS ratios. The models were trained on 23 samples based on column testing methods of different ashes in Switzerland. In these experiments, different chemical properties of the bottom ash were measured, 29 elements (Arsenic, Bromide,...), pH, Redox potential and conductivity.

A subset of these elements were picked to best fit elements of interest decided by the Swiss Federal Council. Fifteen elements were trained and tested, their mean-absolute-percentage error being heavily dependant on the experiment in question, and the mean results over all experiments were as follows: Aluminium = 374%, Antimony = 92%, Arsenic = 26%, Cadmium = 35%, Calcium = 98%, Chloride = 99%, Chromium = 29%, Cobalt = 18%, Copper = 100%, Lead = 96%, Manganese = 12%, Nickel = 24%, Sulfate = 91%, Total Organic Contenct (TOC) = 36% and Zinc = 165%. Sensitivity analysis using SHapley Additive exPlanations (SHAP) shows that the models' predictions are mainly due to the LS value, the concentration we are trying to predict, or functions of it, making these the most important features.

In conclusion two of the tested models, both Random Forest (RF) and Neural Networks (NN), demonstrate significant potential, but additional training data is required to enhance their performance to a level suitable for industrial applications.

Contents

List of Abbreviations

Abbreviation	**Definition**
MSW	Municipal Solid Waste
TOC	Total Organic Carbon
AI	Artificial Intelligence
ML	Machine Learning
RF	Random Forest
GPR	Gaussian Process Regressor
NN	Neural Network
SHAP	SHapley Additive exPlanations
LS	Liquid-to-Solid ratio
KDE	Kernel Density Estimate
IQR	Interquartile Range
CV	Cross-Validation
LOQ	Limit of Quantification

Chapter 1

Introduction

When waste is produced in Switzerland, it is generally incinerated or recycled, each accounting for about 50% of Municipal Solid Waste (MSW) waste, while less than 1% is landfilled [Age16]. This is despite the country's high MSW production, with Switzerland producing 730 kg per capita per year, significantly higher than the European average of 474 kg per capita per year. When incinerated, energy is created and the remaining incineration bottom ash is then landfilled.

The bottom ash is composed of inert, non-combustible materials left over after the combustion process [Eur24]. In state-of-the-art cases, a recovery rate of 80% can be reached [Eur24]. The bottom ash is cooled in two main ways, cooling by air (dry ash), or using water (wet ash) [int24]. The dry bottom ash often has the highest, and lowest non-iron fraction [Abf23] in Switzerland, but this differs by facility. In Hinwil (Canton Zürich) for example, the wet discharged bottom ash has 5 times higher values for lead, 25 times for cadmium and 250 times higher values for copper [Res24]. Also, because cooling takes longer, the hardening of the bottom ash will take longer, allowing for longer intervals for recoveries, and the TOC content in the bottom ash is further reduced due to the ash burning longer [Res24; Gla21]. This then greatly affects the expected concentrations of the measured elements in the ashes.

Once cooled, the bottom ash is landfilled. Landfills can pose risks to the environment despite bottom sealing [Age24], through leaching, mobilization and subsequent spreading from the primary areas [Udd+19]. This can potentially spoil the groundwater and the soil. The landfills are therefore heavily regulated [Fed24], and knowing the concentrations of several different elements during the lifetime of the landfill is important.

Leaching is the extraction of a soluble material from an insoluble solid by dissolution in a solvent (in this case water). Leaching tests provide a means to determine the concentrations of the measured elements by simulating real world leaching [Ers+23]. The solid material (bottom ash) is brought in contact with the solvent (water) producing a leachate which is analyzed for its chemical composition [Ers+23; Age24]. The tests allows for the evaluation of the cumulative mass release based on the liquid-to-solid (L/S) ratio.

There are several methods for leaching tests. Batch tests are usually low cost, and are carried out over hours to days. However, they are associated with arbitrary L/S ratios [Wan16], some of which may be too high for field conditions. Column tests on the other hand, may be closer to the field conditions [Wan16], have better reproducibility, and can assess the leaching behaviour of the pollutants over extended periods [Rou+08; Ers+23]. Generally, column leaching tests are preferred because of the aforementioned reasons.

In this study, we implement both classical and modern models to predict the behaviour of the column leaching tests. We investigate whether the use of ML models and classical models are appropriate for the current data set, and if they can be used to predict the long-term leaching behaviour of the column leaching tests, using short-term results of the leaching tests. This would significantly reduce the duration needed for the leaching tests from several weeks (LS=10) to a couple of days, or even hours (LS=1). In this project we i) gathered experimental leaching test data from the university of bern, ii) developed several models for the prediction of these elements and iii) used SHAP to analyze the importance of the input features to further understand the output of the model (Section 5.8).

Due to their ability to find patterns despite the complexity of the governing processes and material properties, like the soil, density, and composition of the leachate, Machine Learning (ML) and Artificial Intelligence (AI) have been increasingly used in the context of leaching in landfills [Pie+23; BBE17; BDE09]. Other models which have been tried include fuzzy-logic [BEG19] models as well as classic models like Ridge and RF methods [Ers+23].

The leaching test data can be utilized for supervised machine learning, where we can expand the input dimension by projecting the features. The process involves preprocessing the inputs and developing models that demonstrate strong performance. A major challenge in this study is the limited amount of training data, which is not surprising given the time-consuming nature of the experiments. Furthermore, variability in the type of ash can lead to significant differences in the data. Similar efforts to predict long-term leaching behavior from short-term data have been explored in [Ers+23]. Key distinctions in our study include the volume and type of training data; while the referenced study has a larger dataset and focuses on the prediction of 5 different chemical properties, our study considers experiments with 32 different properties. Despite the limited number of experiments, our approach incorporating feature projection shows promising improvements in performance.

Chapter 2

Data

2.1 Leaching Tests

Leaching tests are conducted to estimate to what extent solid materials release elements (typically pollutants) into the soil, ultimately contaminating the surrounding environment. The procedure of a leaching test to see whether the bottom ash meets the landfill criteria is as follows: i) Prepare samples (bottom ash); ii) Prepare solvent (usually water); iii) Mix the sample with the solvent, the solution then passes through the solid and is then collected; v) Analyze the contaminants.

An important term which shows up in this context is the (L)iquid-to-S(o)lid ratio [L/kg]. The LS ratio represents how much liquid we have to the solid. To reach higher LS ratios, one must conduct the experiment for longer. As a rule of thumb, one adds about 6 hours of labwork for each 0.1 LS. So an LS value of 0.5 would require about 30 hours, and an LS value of 10, can take around three weeks [Tiw+15]. The LS value can be related to the duration of leaching in the field experiment using the formula from [FG17]:

$$T_{field} = \rho_s \cdot (1 - n) \cdot d/q \cdot LS \tag{2.1}$$

where ρ_s,n,d and q are the dry solid density (kg/L), porosity (-), the thickness of the release zone (m), and the percolation rate (m/s) [Ers+23]. For typical parameters, even low LS values (LS=2) can correspond with field times over 20 years. Given that landfills generally have a long-term maintenance of maximum 50 years [Umw24], LS values at 5 and 10 are important. If $M_{released,LSX}$ is the released mass at LSX, $V_{w,i}$ the water volume of individual fraction and $c_{frac,i}$ the average concentration for different time spans, then the following holds [FG17]:

$$M_{released,LSX} = \sum_{i=1}^{n_{frac}(LS=X)} C_{frac,i} \cdot V_{w,i} \tag{2.2}$$

2.2 Data

The dataset used for this thesis contains 23 experiments. It was provided by Fachstelle Sekundärrohstoffe, University of Bern. These experiments can further be categorized into nine wet ash experiments, five mixed ash (48.6% dry ash, 49.63% wet ash, 1.7% filtered ash ash), eight dry ash and one magnetic ash. Three of the dry ashes can further be categorized into the size of the particles, 2-16mm, 0.2-2mm and <0.2mm. The smaller sizes often contain heavier metals [Gla21]. The magnetic ash has had its magnetic constituents removed by a magnet, before the experiment began.

Wet ashes are cooled using water, whereas dry ashes are cooled by air. This can affect the properties of the ash, for example, dry ashes will often have a lower TOC concentrations because they have been burning for longer [Gla21].

The concentrations of 32 different elements (see appendix for an overview of the elements: A.0.1) at seven different LS values were measured. Their elements of interest, as well as the elements of interest as described in [Fed24] (page 34) were used as inputs and outputs for our purposes, so a subset of these 31 elements. The concentrations were measured at LS = [0.1,0.2,0.5,1,2,5,10]. 15 concentrations (Aluminium, Antimony, Arsenic, Cadmium, Calcium, Chloride, Chromium, Cobalt, Copper, Lead (Pb+2), Manganese, Nickel, Sulfate, Total Organic C (TOC) and Zinc) as well as pH and Electric conductivity make up the inputs and outputs of our models. So in total, for one given experiment if every property of interest was measured at each LS value, there are 17*7=119 data points. However, a fair amount of data points were not measured. This is summarized in Table 2.1. In total there were 2424 measured data points (Figure 2.1), as 313 points were missing.

The data is plotted in the following violin-density figures (Figure 2.1). It is a way to combine box-plots and kernel density estimator (KDE) plots. The box plots are embedded within the violin shapes, consisting of a black box that represents the standard deviation, with a white point indicating the mean. The surrounding KDE plots are calculated using kernel density estimation. The function can be calculated by summing over the observations, where each observation is assigned its own kernel (in this case, a Gaussian kernel) with a width λ [Tre09]. If X denotes our dataset (our concentrations), N the amount of points, λ the width (typically the standard deviation), then the gaussian density estimate at the point x_0 is given by:

$$\hat{f}_X(x_0) = \frac{1}{N(2\lambda^2\pi))^{\frac{p}{2}}} \sum_{i=1}^{N} e^{-\frac{1}{2}(||x_i - x_0||/\lambda)^2} \tag{2.3}$$

x_i are the measured points, p is the dimension the features, $x \in \Re^p$, N the number of points, $|| \cdot ||$ is the euclidean distance.

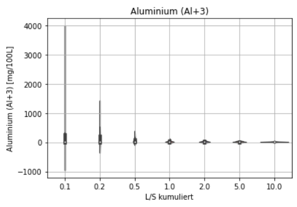
Element	Wet	Mix	Dry	Magnetic
LS value	0	0	0	0
pH	0	0	0	0
Aluminium	0	1	0	0
Antimony	7	4	21	0
Arsenic	7	34	41	0
Cadmium	0	7	7	0
Calcium	0	0	0	0
Chloride	0	28	8	0
Chromium	0	6	6	0
Cobalt	0	6	4	0
Copper	0	5	3	0
Lead (Pb+2)	0	5	4	0
Manganese	0	7	8	0
Nickel	0	2	3	0
Sulfate	0	21	0	0
Total Organic C (TOC)	0	35	28	0
Zinc	0	1	4	0

Table 2.1: Amount of missing data points, by element and type of ash

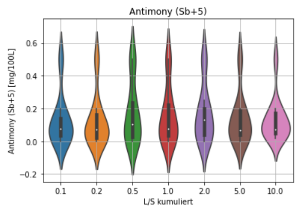
In Figure 2.1, the concentrations of elements plotted against LS values are illustrated. It is observed that manganese shows relatively consistent behavior, with its standard deviation (depicted by the black box) and mean (indicated by the white dot) exhibiting minimal variation compared to other elements like aluminium. Thicker shapes in the plot signify more concentrated data, meaning that data points are closely grouped around certain concentrations.

It's important to note that the apparent constancy of the plots does not imply that the concentration values themselves are stable over time. This is because the Kernel Density Estimate (KDE), as shown in Equation 2.3, is rotationally invariant. Consequently, swapping two concentrations does not alter the shape of the KDE plot.

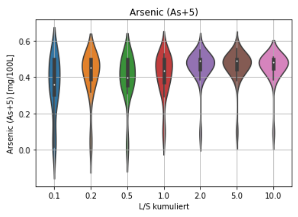
Additionally, elements like cobalt and aluminium exhibit higher variability in their starting concentrations (LS = [0.1, 0.2]) compared to elements such as antimony, as evidenced by the thinner lines at these LS values. Thinner lines represent greater spread in the data, while thinner black boxes indicate lower variance. The white dot within the black box represents the mean concentration of a given element at the corresponding LS value.



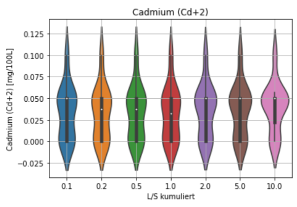
(a) Aluminium



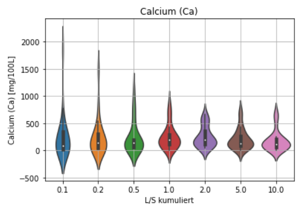
(b) Antimony



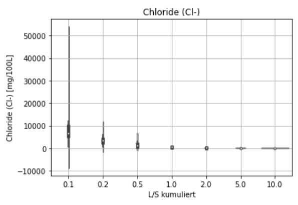
(c) Arsenic



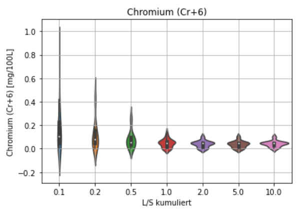
(d) Cadmium



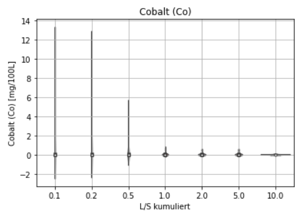
(e) Calcium



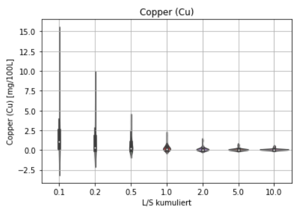
(f) Chloride



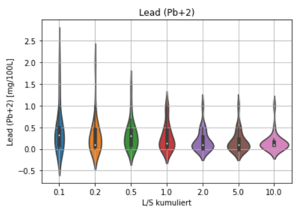
(g) Chromium



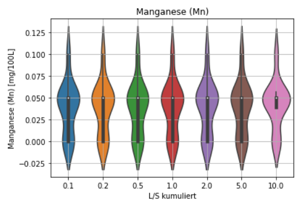
(h) Cobalt



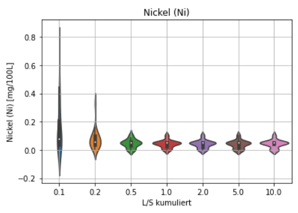
(i) Copper



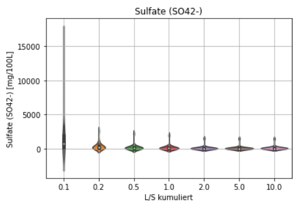
(j) Lead



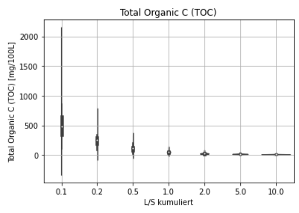
(k) Manganese



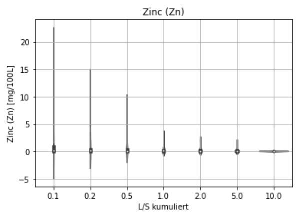
(l) Nickel



(m) Sulfate



(n) TOC



(o) Zinc

Figure 2.1: Kernel probability density of the concentrations. Thicker violins mean more observed measurmenets at these concentrations at a given LS value. The black box indicates the standard deviationo, and the white points the mean.

2.3 Data Preparation & Missing Data

The dataset contains a significant number of missing values, which need to be addressed. We differentiate between values that are below the measurable limit and those that are unmeasured (NA = not acquired). To fill these missing values, one might intuitively consider extrapolation, as has been done in similar projects [Ers+23]. However, in many cases, none of the LS values for a given element in an experiment were measured, making such extrapolation impossible for those experiments

Instead, it was assumed that the concentration measurements of different elements follow a normal distribution and that the missing values were missing completely at random (MCAR). This implies that the absence of a measurement was due to random factors rather than an assumption about the concentration (e.g., it being very low). This approach can be seen as a one-dimensional, single-layer application of the more general (deep) Gaussian process for imputation, as described in [Jaf+23]. To estimate the missing values, the following steps were taken: (i) Identify a missing value at a specific LS value for an element, (ii) use the non-missing concentrations for this element at the same LS value to calculate a mean and standard deviation, and (iii) draw a random sample from the Gaussian distribution defined by this mean and standard deviation, $\mathcal{N}(\mu, \sigma)$.

Consider for example the distribution for Zinc at LS=10. Zinc has few missing values (five in total; see Table 2.1) which have been filled using this imputation method, resulting in e.g. the distribution shown in Figure 2.2.

Additional distributions can be found in the appendix: A.1. when interpreting these distributions, consider Table 2.1 which highlights the extent of imputation using a Gaussian model. In cases where only a few values were measured, the imputed distributions might appear overly Gaussian. For instance, if only two values were measured, the resulting distribution would be a Gaussian centered at their arithmetic mean with a width based on their difference, which may not accurately reflect the underlying data.

The plot seen in figure 2.2 is the plot corresponding with the distribution of the concentration of Zinc at LS 10. In the appendix: A.1, the corresponding plots to all elements and all LS values can be found. They can be an indicator of 'outlier detection', in the sense that one experiment has much higher measured concentrations than the other experiments. They also provide clues about whether or not the gaussian method to fix the missing values (Table 2.1) is appropriate.

Generally, if the bars are all connected, the gaussian approach is more appropriate. If they are not, then this approach may artificially create Gaussians. Cadmium may be such an example (Figure A.4) at LS = 2,5, and 10. We note that Cadmium could only have measured values in the intervals [0.00,

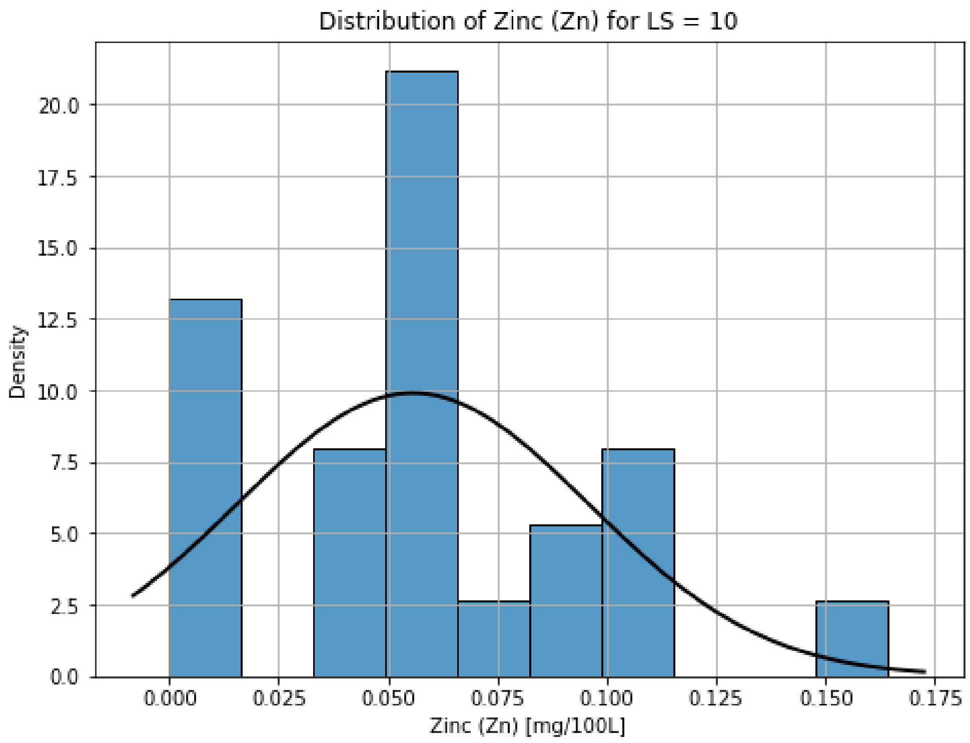


Figure 2.2: Distribution of the concentrations of Zinc at LS=10. Additional distributions can be found in Figure A.1.

0.01] and [0.09, 0.10], for these LS values, and that the values which show in between are a result of the Gaussian imputation method described in section 2, because we have Gaussian-like shapes at the mean between these two values.

When handling unmeasurable data, specifically concentrations below the limit of quantification (LOQ), three approaches were considered: (i) selecting a random number between 0 and the LOQ, (ii) setting the concentration to zero, or (iii) using half of the LOQ value. Since the choice had minimal impact on training or model output, option (iii) was selected for its ease of use, particularly in terms of scaling and feature projection.

2.3.1 Scaling

Many ML models struggle to find trend in the data if the scales of the inputs are vastly different. In Figure 2.1, we observe that some differences between conenctrations at a given LS value can be over three orders of magnitude. Indeed, many of sklearns' algorithms may behave badly if the features do not look like more or less a gaussian with zero mean and unit variance [Lea24]. This highlights the importance of conducting feature scaling.

Five different scaling methods were tested: No scaling, logarithmic scaling, robust scaling, standard scaling and MinMax scaling. The last three can be reviewed in [lea24]. Mathematically, the scalers can be summarized as follows, where $\varphi(x) : x \to \varphi(x)$ is the scaled data:

$$\begin{aligned}
\textit{No scaling:} \quad & \varphi(x) = x && (2.4)\\
\textit{Logarithmic scaling:} \quad & \varphi(x) = \ln(x + \varepsilon) && (2.5)\\
\textit{Robust scaling:} \quad & \varphi(x) = \frac{x - med(x)}{IQR(x)} && (2.6)\\
\textit{Standard scaling:} \quad & \varphi(x) = \frac{x - \mu}{\sigma} && (2.7)\\
\textit{MinMax scaling:} \quad & \varphi(x) = \frac{x - max(x)}{x - min(x)} + \varepsilon && (2.8)
\end{aligned}$$

where μ is the mean of the data (the value of an elements' concentration for a given LS value averaged over the experiments). σ is the standard deviation, ε is a small number, so we do not divide by zero. Note that MinMax and Standard scaler are sensitive to outliers [lea24]. The Interquartile Range (IQR) is calculated as the difference between the medians of the lowest 25% and the highest 75% of the data, and med(x) refers to the median.

Outliers will remain as outliers when using the Robust Scaler, unlike with MinMax or Standard scaling. For instance, MinMax scaling may compress all other values towards zero if a single value is much larger than the others. The Robust Scaler, however, preserves the relative position of outliers, allowing them to remain discernible.

Figure 2.3 provides an illustration of the effects of different scaling methods on the data. Notably, the MinMax scaler compresses many values close to 0 and 1, a consequence of its sensitivity to extreme values. This is evident in the case of the outlier in the "No Scaling" plot—specifically, the data point at [45000, 9]—which is now less conspicuous due to the scaling.

In the "No Scaling" plot, this outlier is prominent due to its values being several orders of magnitude larger than most other points. This highlights the need for scaling to manage such discrepancies.

Conversely, the Robust scaler, which is designed to mitigate the influence of outliers, does not include this extreme value in its normalization. As a result, the outlier remains distinct in the plot. Meanwhile, the Standard scaler reduces the impact of this outlier, making it less apparent compared to the other scaling methods.

MinMax will linearly transform the data to a fixed range (Equation 2.8), because at least one scaled value would be zero, we add an ε-term. ε avoids zeros, and is not a part of sklearns' MinMax scaling. This is important for

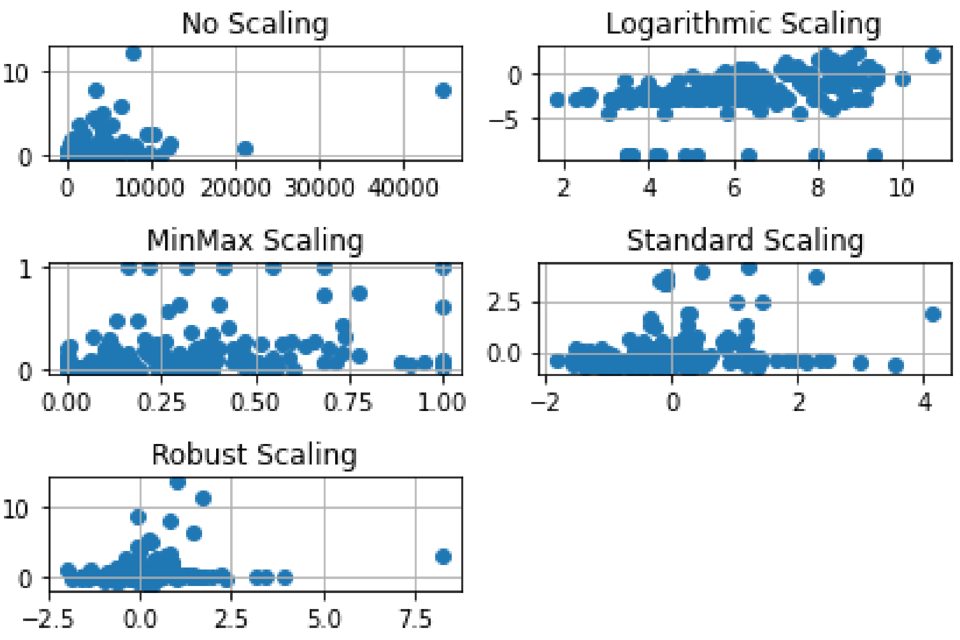


Figure 2.3: Comparison scatter plot of Chloride (x-axis) with Copper (y-axis). All the data has been accumulated and been plotted (so all experiments, and all LS values). Note the outlier for not scaling remains an outlier when using the robust scaler.

feature engineering (Section 2.3.2). In particular, addition ensures that we do not divide by zero during feature projections.

2.3.2 Feature engineering and inputs

In this section, the inputs for the models will be defined.

Recall that the LS values correspond to time in the field as described by Equation 2.1:

Given that LS values are directly related to time in this context, it is logical to include LS as a key input for our models. Incorporating LS into the feature set can be beneficial. For instance, instead of using only the 17-dimensional input vector (which includes concentrations of 15 elements, pH, and conductivity), we could expand the feature vector to include LS. This would provide the model with additional information about the leaching stage at which the concentration is to be predicted.

The approach of finding new inputs based on the previous ones is more generally called feature projection, or feature engineering [C E06]. Here, 4 additional feature projections were used as inputs:

$$\textit{Target LS} \tag{2.9}$$
$$\textit{LS*[Concentration at LS= 0.01]} \tag{2.10}$$
$$\textit{[Concentration at LS = 0.01]/LS} \tag{2.11}$$
$$\textit{exp(-[Target LS])*[Concentration at LS = 0.01]} \tag{2.12}$$

In this case then, n = 17, and D = 21. The third line (Equation 2.11) makes it clear why during scaling we must add some value to remove zeroes from our input as scaling happens before projection. Dividing by zero is ill-defined. Equation 2.12 is rooted in rate equations, which often appear in chemistry [Lib23]. The solutions often have the form of exponentials, because:

$$rate = k[A]^x[B]^y \tag{2.13}$$
$$\Rightarrow \frac{d}{dt}C = -A * C \tag{2.14}$$
$$\Rightarrow C(t) = Bexp(-tA) \tag{2.15}$$

With x and y the respective orders of reactions with the concentrations [A] and [B], and k the rate constant.

In machine learning, especially when dealing with models like Gaussian Process Regression (GPR), it's crucial to manage the input dimensions carefully. High-dimensional inputs can lead to poor performance and computational inefficiencies, particularly when the amount of training data is limited. This is often referred to as the "curse of dimensionality" [Che09], and it's a significant issue when working with models that are sensitive to input dimensions, such as sklearn's GPR [lea23].

To address this, one strategy is to select only the most relevant features based on their importance. A common approach is to use correlation analysis. By examining the correlation coefficients between the output variable (the target element we aim to predict) and each input feature, we can identify which features have the strongest relationships with the output.

For example, if we are predicting the concentration of Calcium at a specific LS value, we can calculate the correlation coefficients between Calcium and each of the other features in our dataset. We then select the features with the highest correlation coefficients, as these are the ones most likely to contribute significantly to predicting the target variable.

Consider Figure 2.4, which illustrates the correlation coefficients between Calcium concentration at LS=0.2 and the other features. By focusing on features with higher correlation coefficients (e.g. Calcium, Lead, Conductivity and Sulfate), we can streamline our feature set, improving model performance and mitigating the effects of the curse of dimensionality.

In the following sections, ”smart inputs” will be used to describe the process of selecting only the most relevant features based on their correlation coefficients.

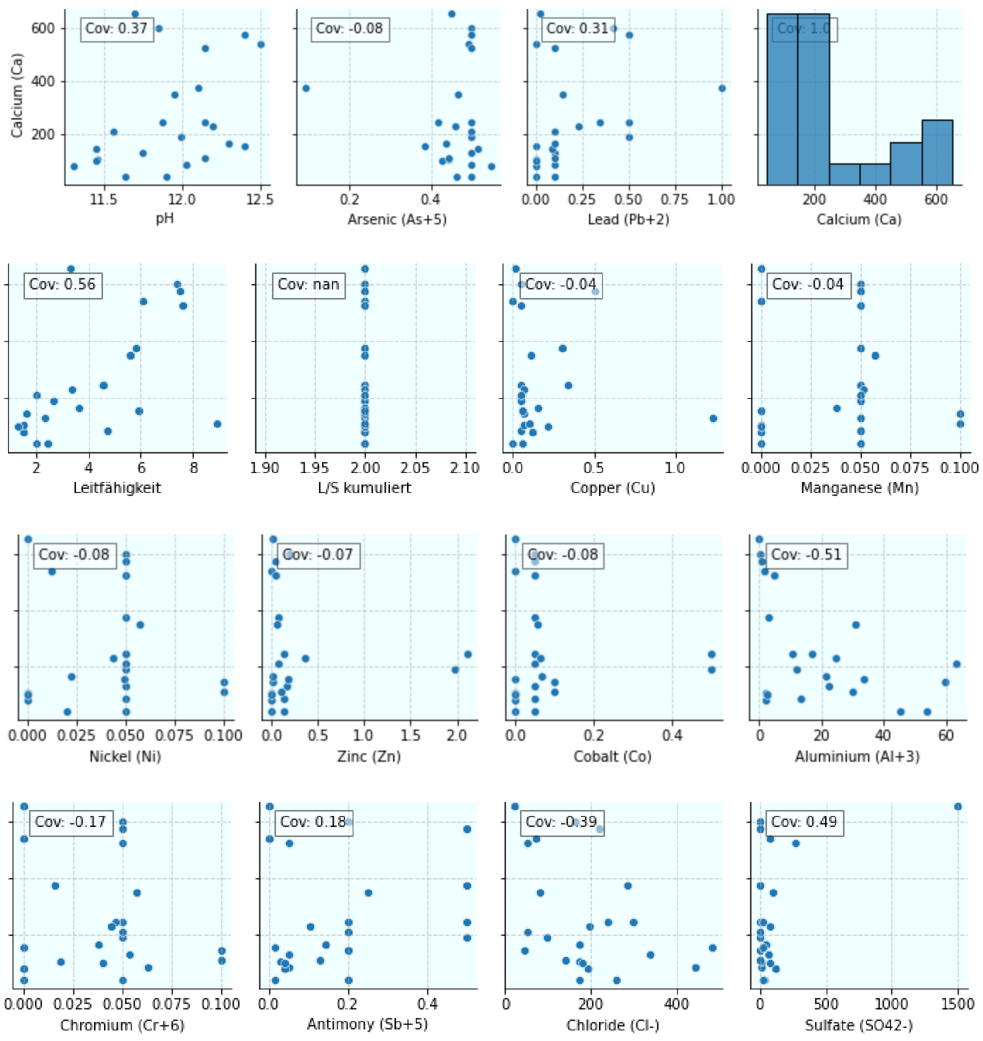


Figure 2.4: Correlations for Calcium at LS = 0.2. Sulfate and conductivity correlate more strongly, making them potential candidates to keep in a reduced input. This does depend on which correlations we see at the other LS values as well.

2.3.3 Jittering

Jittering is a data augmentation technique used to artificially increase the size of a dataset by introducing slight variations or noise to the existing data. [GBC16] This method is commonly employed in fields such as image recognition, where minor alterations (e.g., rotations) of an image still represent the same object. In the context of machine learning, jittering helps to enhance model robustness by providing additional training examples that capture the variability inherent in real-world data.

In this study, Gaussian jittering was applied as follows:

1. **Select Parameters:**
   Choose a width σ for the gaussian noise, and specify the number N of jittered samples to create
2. **Gaussian Noise Creation:**
   (i) For each value z_i in the dataset (including both target values y_i and features x_i), multiply z_i by the width σ.
   (ii) Construct a Gaussian distribution $\mathcal{N}(z_i, \sigma)$ centered around z_i with a standard deviation of 10%. This reflects the uncertainty of measurements of 10%.
   (iii) Sample N random values from this Gaussian distribtuion to create N jittered data points for each orginial point.
3. **Data Augmentation:** This process effectively increases the training set size by a factor of N+1. The new data points are highly correlated with the original points, as they are generated from Gaussian distributions centered around these points. For example, if the original dataset contains high concentration values, the jittered samples will also reflect high concentrations, preserving the distribution characteristics of the original data.

By employing jittering, the model benefits from a more diverse training set, potentially leading to improved generalization and robustness.

Leaching tests yield numerous data points that represent concentrations at various LS values. However, some data points are missing, so Gaussian imputation is employed to fill in the gaps. The resulting dataset exhibits high variability, as illustrated in Figure 2.1, making feature scaling essential. Additionally, feature projection can be applied to enhance the models by extracting and highlighting critical information necessary for accurate prediction.

Chapter 3

Machine Learning Methods

3.1 Models

This section provides an overview of the different models employed in this study, detailing their theoretical foundations, theories behind optimization, training procedures, and methods for calculating outputs. Understanding these models and their parameters is crucial for interpreting their performance and for making informed decisions about model selection and tuning. Readers who are already familiar with these concepts may skip this section and proceed to Section 4.

3.1.1 Ridge & Lasso

The main resource for the following discussion is [Tre09]. Ridge and Lasso are both methods to estimate the coefficients of regression models where the inputs are highly correlated. They only differ in terms of regularization.

Ridge and Lasso Regression are both linear w.r.t the input vector $\mathbf{x} = (x_1, x_2, x_3, ...x_p)$. In our case, as detailed in Section 2.3.2, the inputs include concentrations of elements as well as some feature projections.

The linear regression model has the form:

$$f(\mathbf{x}) = \beta_0 + \sum_{i=1}^{p} \beta_i x_i \tag{3.1}$$

where β is a vector containing the weights.

To measure the prediction error, scikit-learn uses least squares, which minimizes the sum of squared differences between the observed values y_i and the predicted values over all feature-label pairs $\{y_i, f(\mathbf{x}_i)\}_{i=1}^{N}$):

$$\mathcal{L}(y, f(x)) = \sum_{i=1}^{N}(y_i - f(x_i))^2 = \sum_{i=1}^{N}(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij})^2 \tag{3.2}$$

Where in the last line, $\mathbf{X}$ is the $N \times (p+1)$-Matrix containing the features, and $\mathbf{y}$ is the vector containing the labels, β is the vector containing the weights. This can be differentiated, and set equal to zero to find the optimum for the weights in β.

To reduce the number of features used, we can apply regularization (or shrinkage) by penalizing large values of β. Adding a β^2 term results in *Ridge regression*, whereas adding $|\beta|$ results in *Lasso regression*:

$$\hat{\beta}^{ridge} = \underset{\beta}{argmin}[\sum_{i=1}^{N}(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j) + \lambda \sum_{j=1}^{p}\beta_j^2] \tag{3.3}$$

$$\hat{\beta}^{lasso} = \underset{\beta}{argmin}[\frac{1}{2}\sum_{i=1}^{N}(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j) + \lambda \sum_{j=1}^{p}|\beta_j|] \tag{3.4}$$

λ is the regularization parameter that controls model complexity. Larger values of λ will yield few or small values for β, whereas small values will leave β unconstrained. Note that for small values of β, the penalty imposed by Ridge is less severe. This has the consequence of Lasso being better at *feature selection*, as Lasso will often set some feature weights to zero, a behaviour less pronounced in Ridge.

We use LassoCV and RidgeCV, which automatically select the optimal complexity parameter λ by evaluating a range of possible values. The best λ is chosen based on performance metrics derived from cross-validation (CV), as detailed in Section 3.3.

3.1.2 Neural Network

The following section partly follows [Tre09].

Neural networks are inspired by the way biological neuronal networks function. In the brain, neurons are interconnected and communicate through electrical and chemical signals. When one neuron is activated, it can trigger an electrical signal that travels through the network, influencing other neurons and contributing to the formation of complex thoughts and behaviors. This intricate communication system in biological neural networks serves as the foundation for the design of artificial neural networks, where layers of interconnected nodes (analogous to neurons) work together to process and interpret data.

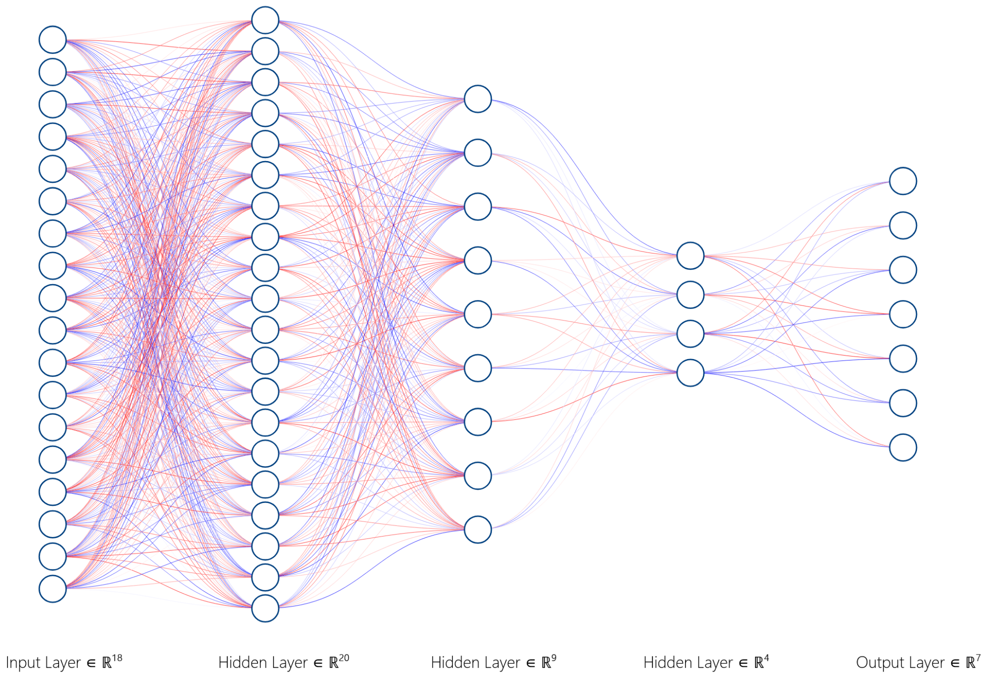
Neural networks have gained a lot of *hype* in the last couple of years. The central idea is that we take our (potentially projected) input features, and linearly transform them as seen in Section 3.1.1 to some new space. We then repeat this M times (once for each *layer*), before transforming this to our output space, hoping that the network (the ensemble of the layers) has learned something clever along the way.

In Section 3.1.1, we saw that the output of the model had the form:

$$f(\mathbf{x}) = \beta_0 + \sum_{i=1}^{p} \beta_i x_i \tag{3.5}$$

What if, instead of considering this the final output, we let this be the *input* to another regression model? We would then generally have the first output be more than one dimensional, creating a vector $\mathbf{y}$.

We can then repeat this M times. In Figure 3.1, an example is shown, where this is repeated M = 4 times.



Input Layer ∈ $\mathbb{R}^{18}$
Hidden Layer ∈ $\mathbb{R}^{20}$
Hidden Layer ∈ $\mathbb{R}^{9}$
Hidden Layer ∈ $\mathbb{R}^{4}$
Output Layer ∈ $\mathbb{R}^{7}$

Figure 3.1: In this figure, we have three hidden layers, the color of each neuron weight is proportional to its strength, i.e. red lines correspond with relative higher weights. Made using [Aut24b]

The output of the first layer is then:

$$y_j = (\sum_{i=1}^{N} w_{ji} x_i + b_j) \tag{3.6}$$

where w_{ji} contains our weights. We can choose to make the network more complex by imposing new functions on our vector $\mathbf{y}$. For example, we can set y_j to zero if it is negative, and leave it if it is positive. Functions which are applied after this linear transformation (Equation 3.6) between two layers are called *activation functions*. If we include such functions, then the output of the first layer is:

$$y_j = f(\sum_{i=1}^{N} w_{ji}x_i + b_j) \tag{3.7}$$

Where $j \in [1, 2, ..., M_1]$, the index of our first layer, and f the activation function, which acts component-wise, some common choices for the activation function include:

$$f(x) = \sigma(x) = \frac{1}{1+e^{-x}} \tag{3.8}$$

$$RELU(x) = max(x, 0) \tag{3.9}$$

$$ELU(x) = \begin{cases} x \quad if \quad x > 0 \\ \alpha \cdot (exp(x) - 1) \quad if \quad x \leq 0 \end{cases} \tag{3.10}$$

$$\tag{3.11}$$

sklearn uses α=1 as default.

We can repeat Equation 3.7 more times, and we can represent w_{ij} as a matrix $\mathbf{W}$. We will have one matrix for each layer in our network, and its shape depends on the shape of the layers before and after the transformation. We give the matrix a superscript corresponding with which layers it transforms between, the matrix from the first to the second layer is an $M_1 \times D$ matrix for an input dimension of D:

$$\mathbf{W}^1 = w_{ij}^l \tag{3.12}$$

More generally, the l-th layer in a neural network has the following form:

$$\mathbf{y}^l = f^l(\mathbf{W}^l\mathbf{y}^{l-1} + b^l) \tag{3.13}$$

With b the *bias*. The final output for the entire network with m hidden layers (the layers between the input and output) is then:

$$output = \mathbf{y}^{m+1} = \tag{3.14}$$

$$f^{m+l}(\mathbf{W}^{m+1}(f^m\mathbf{W}^m \cdots f^1(\mathbf{W}^1\mathbf{x} + \mathbf{b}^1) + \cdots + \mathbf{b}^m)\mathbf{b}^{m+1}) \tag{3.15}$$

A potential problem here is that we are going to have a lot of tuned parameters. For each layer, we get (D+1)*(M+1) trainable parameters for an input layer of dimension D, and output layer of dimension M (the +1 is the bias). When using this many parameters, we can have problems with overfitting. Some methods to reduce overfitting include dropout and early stopping [Tre09]. Dropout will randomly set some percentage of weights equal to zero during training, it has been proven that this can be an effective regularization method [Hin+24]. Early stopping involves stopping learning before we overfit by looking at the convergence on the validation set (see section 3.3 for an overview of validation sets)

The network is trained using gradient descent, as described below in Section 3.1.3.

3.1.3 Gradient Descent

In this section, we will discuss gradient descent in the context of neural networks, the section largely follows [TF01]. The ideas discussed here can be directly transferred to the other methods by simply replacing the predictor in the loss function, see Equation 3.19.

Neural network has unkown parameters, these are called weights. One can consider the output of a neural network to consist of many matrix multiplications, with each entry having a weight. The complete set of weights is denoted as (for a single hidden layer, feed-forward neural network, Figure 3.2):

$$\{\alpha_{0m}, \alpha_m;\ m = 1, 2, ..., M\}\ M(p+1)\ weights \tag{3.16}$$

$$\{\beta_{0k}, \beta_k;\ k = 1, 2, ..., K\}\ K(M+1)\ weights \tag{3.17}$$

The "+1" for the weights is called the *bias*. Consider the sum of squared errors as our error measure, often used in regression:

$$\mathcal{L}(\theta) = \sum_{k=1}^{K}\sum_{i=1}^{N}(y_{ik} - f_k(x_i))^2 = \sum_{i=1}^{N}\mathcal{L}_i \tag{3.18}$$

We can update the weights by minimizing the loss function, which involves calculating the gradient of the loss. The gradient indicates the direction of the steepest increase in the loss, but since our goal is to minimize the loss, we move in the opposite direction. This is done by subtracting the gradient from the current weights, which brings us closer to the minimum of the loss function. The weights are updated as follows:

$$\theta^{r+1} = \theta^r - \gamma_r \partial_\theta \mathcal{L}(\theta) \tag{3.19}$$

$$\frac{\partial \mathcal{L}_i}{\partial \beta_{km}} = -2(y_{ik} - f_k(x_i))g_k'(\beta_k^T z_i)z_{mi}, \tag{3.20}$$

$$\frac{\partial \mathcal{L}_i}{\partial \alpha_{ml}} = -\sum_{k=1}^{K} 2(y_{ik} - f_k(x_i))g_k'(\beta_k^T z_i)\beta_{km}(\alpha^T x_i)x_{il} \tag{3.21}$$

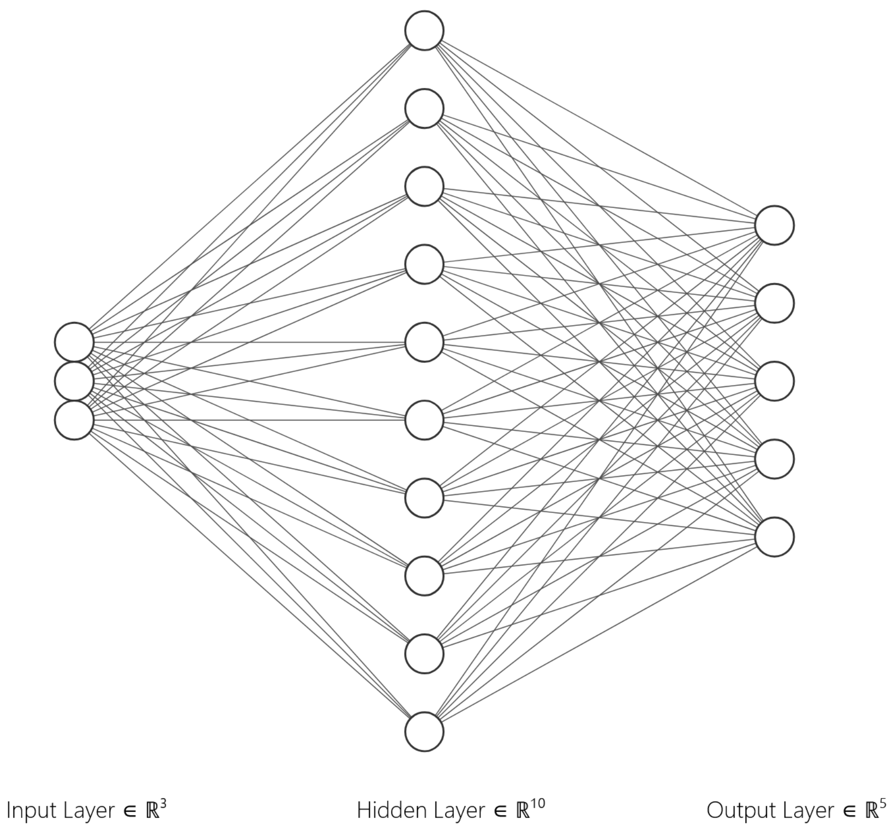


Figure 3.2: Schematic with p=3, M=10 and K = 5. Made using [Aut24b]

Where θ contains the parameters (here, the α's and β's), and γ the learning rate (stepsize). We update the weights using:

$$\beta_{km}^{r+1} = \beta_{km}^{r} - \gamma_r \sum_{i+1}^{N} \frac{\partial \mathcal{L}_i}{\partial \beta_{km}^{r}} \tag{3.22}$$

$$\alpha_{ml}^{r+1} = \alpha_{ml}^{r} - \gamma_r \sum_{i=1}^{N} \frac{\partial \mathcal{L}_i}{\partial \alpha_{ml}^{r}} \tag{3.23}$$

Back-propagation and forward-propagation are fundamental concepts in training neural networks. During the forward pass, the model makes predictions using fixed weights θ^r. This involves propagating the input signal forward through the network to calculate the output and associated errors. Once these errors are known, the weights are updated during back-propagation.

The term "back-propagation" comes from the fact that the error signal is propagated backward through the network to update the weights. This process relies heavily on the chain rule of calculus, where the derivatives are computed in reverse order. For instance, when calculating the derivative for the α's, one can reuse the derivatives calculated for the β's, which appear earlier in the network. This sequential updating of derivatives allows for efficient optimization of the weights, as each step builds on the previous ones, effectively "going backwards" through the network.

3.1.4 Gaussian Regression

The main resource for the following can be found in [C E06]. In this section, we will discuss the theory behind the Gaussian Process Regression. Gaussian Process Regression is a non-parametric method supervised learning method often used in regression. It is non-parametric in the sense that it dynamically changes its function distribution once we add more training data. This is based on bayes' theorem, where the posterior is adapted when the likelihood is changed.

Reminder: A Gaussian distribution is often written using the following notation (the $\boldsymbol{y}$*'s, are often left implicit).*

$$exp(-\frac{1}{2}[(y-f(x))^T\Sigma^{-1}(y-f(x))] = \mathcal{N}(f(x),\Sigma) = \quad Here: \quad p(\mathbf{y}|f,x) \tag{3.24}$$

We have n features $\{\mathbf{x}\}_{i=1}^n$ and n observations (labels) $\{\mathbf{y}_i\}^n$, Σ is the covariance matrix. If y was a vector, then $\mathbf{y}_i - f(\mathbf{x}_i) \in \Re^K$ would have been a vector, and the matrix Σ would be the $K \times K$ covariance matrix. Here, because our concentrations are one dimensional, we will assume the labels to be one dimensional.

We assume that the observed values can be seen as gaussians centered about their true values:

$$y_i = f(x_i) + \varepsilon_i \simeq \mathcal{N}(f(x_i), \sigma_n^2) \quad f(x_i) = \mathbf{x}_i^T\mathbf{w} \tag{3.25}$$

$\mathbf{w}$ is our weight vector. The assumption that it is a Gaussian with uncorrolated noise (since σ_n^2 is diagonal), is referred to as a gassian prior.

The joint probability is then, if we assume the draws to be independent of each other:

$$p(\mathbf{y}|\mathbf{X},\mathbf{w}) = \prod_{i=1}^n p(y_i|\mathbf{x}_i,\mathbf{w}) = \prod_{i=1}^n \frac{1}{\sqrt{2\pi}\sigma_n} exp(-\frac{(y_i - \mathbf{x}_i^T\mathbf{w})^2}{2\sigma_n^2} \tag{3.26}$$

$$= \frac{1}{(2\pi\sigma_n^2)^{n/2}} exp(-\frac{1}{2\sigma_n^2}|\mathbf{y} - \mathbf{X}^T\mathbf{w}|^2) = \mathcal{N}(\mathbf{X}^T\mathbf{w}, \sigma_n^2\mathcal{I}) \tag{3.27}$$

Note that in Equation 3.27, $\mathbf{X}$ is an $l \times n$ Matrix, and $\mathbf{y}$ is a vector of the observations.

For a bayesian model, $p(\mathbf{y}|\mathbf{X}, \mathbf{w})$ is referred to as the likelihood, and $p(\mathbf{w})$ as the prior. The prior expresses our knowledge before any observation, which for us is simply Gaussian distributions with widths given by the matrix Σ. The "real" distribution is the prior weighted by the likelihood, and normalized by the marginal likelihood:

$$posterior = \frac{likelihood \times prior}{marginal\ likelihood}, \quad p(\mathbf{w}|\mathbf{y}, X) = \frac{p(\mathbf{y}|\mathbf{X}, \mathbf{w})p(\mathbf{w})}{p(\mathbf{y}—\mathbf{X})} \tag{3.28}$$

The normalization constant $p(\mathbf{y}|\mathbf{X})$ represents the probability of generating the observed sample for all possible values of the parameters; it can be understood as the probability of the model itself and is therefore sometimes referred to as model evidence. It is independent of the weights $\mathbf{w}$ and given by:

$$p(\mathbf{y}|X) = \int p(\mathbf{y}|X, \mathbf{w})p(\mathbf{w})d\mathbf{w} \tag{3.29}$$

To calculate the posterior more explicitly, we 'complete the square' by calculating the following using the covariance matrix over the features Σ

$$p(\mathbf{w}|X, \mathbf{y}) = \frac{1}{p(\mathbf{y}|X)} exp(-\frac{1}{2\sigma_n^2}(\mathbf{y} - X^T\mathbf{w})^T(\mathbf{y} - X^T\mathbf{w}))exp(-\frac{1}{2}\mathbf{w}^T\Sigma^{-1}\mathbf{w}) \tag{3.30}$$

$$= \frac{1}{p(\mathbf{y}|X)} exp(-\frac{1}{2\sigma_n^2}(\mathbf{y}^T\mathbf{y} - 2\mathbf{y}X^T\mathbf{w} + \mathbf{w}^T \underbrace{(XX^T + \sigma_n^2\Sigma^{-1})}_{=:A} \mathbf{w}) \tag{3.31}$$

$$\tilde{\mathbf{w}} := \mathbf{y}X^T \tag{3.32}$$

$$= \frac{1}{p(\mathbf{y}|X)} exp((\mathbf{w} - A^{-1}\tilde{\mathbf{w}})^T A(\mathbf{w} - A^{-1}\tilde{\mathbf{w}})) \tag{3.33}$$

$$= \frac{1}{p(\mathbf{y}|X)} \mathcal{N}(A^{-1}\tilde{\mathbf{w}}, A^{-1}) \tag{3.34}$$

Notably, in the first line (Equation 3.30), even before diving into the exponentials, we can already identify a "fitting term" and a "penalty" term, similar to Ridge regression:

To find the predictive distribution of a given point $\mathbf{x}_*$, we average the output of all possible models with our posterior using $f_* := f(\mathbf{x}_*)$:

$$p(f_*|\mathbf{x}_*, X, \mathbf{y}) = \int p(f_*|\mathbf{x}*, \mathbf{w})p(\mathbf{w}|X, \mathbf{y})d\mathbf{w} \tag{3.35}$$

$$= \mathcal{N}(\mathbf{x}_*^T A^{-1}X\mathbf{y}, \mathbf{x}_*^T\sigma_n^2 A^{-1}\mathbf{x}_*) \tag{3.36}$$

Note that this is no longer dependent on the weights. The weights have implicitly been used to weigh the posterior probability. This is in contrast to non-bayesian schemes, where one parameter is picked.

Note also that we can use feature engineering [1] on our inputs. For example, for a given vector $\mathbf{x}_i$, we can project each of its components $\mathbf{x}_i^j$ to $\varphi(\mathbf{x})_i^j = [1, sin(x)_i^j, cos(x)_i^j, ...]$ and consider this as our inputs instead. If we stacked these mappings on top of each other, we would have written:

$$f(x) = \varphi(\mathbf{x})^T \mathbf{w} \tag{3.37}$$

at the beginning instead. The analysis would have been the same as before, but we would need to swap out X with $\Phi(X)$ everywhere. Using the shorthand $\Phi := \Phi(X)$, and $\varphi(\mathbf{x})_* =: \varphi_*$ we would get for Equation 3.36:

$$\begin{aligned} p(f_*|\mathbf{x}_*, X\mathbf{y}) &= \mathcal{N}(\varphi_*^T A^{-1}\Phi\mathbf{y}, \varphi_*^T \sigma_n^2 A^{-1}\varphi) && (3.38)\\ & or && (3.39)\\ &= \mathcal{N}(\varphi_*^T \Sigma_p \Phi (K + \sigma_n^2 I)^{-1}\mathbf{y}, && (3.40)\\ & \varphi_*^T \Sigma_p \varphi_* - \varphi_*^T \Sigma_p \Phi (K + \sigma_n^2 I)^{-1} \Phi^T \varphi_*) && (3.41) \end{aligned}$$

Reminder: Φ is an $D \times n$ Matrix. Its columns are the projections from the features. In particular, $\Phi_{ij} = \varphi(\mathbf{x}_i)_j$, where $\varphi(\mathbf{x}) : x \in \Re^l \Rightarrow x \in \Re^D$.

In the last equation, we defined $K = \Phi^T \Sigma_p \Phi$. We then note that the feature space (the $\varphi's$) always show up as $\Phi^T \Sigma_p \Phi$,$\Phi_*^T \Sigma_p \Phi$, or $\Phi_*^T \Sigma_p \Phi_*$. This is usually defined as the *kernel* $k(\mathbf{x}, \mathbf{x}') = \Phi(\mathbf{x})^T \Sigma_p \Phi(\mathbf{x}')$[2] K is then a matrix with inputs given by $k(\mathbf{x}_i, \mathbf{x}_j)$. Its worth mentioning, that because Σ is a covariance matrix, it is positive definite. Because we see that the output distribution always shows up as a multiplication using the kernel, the features become of second importance, and one usually just speaks of the kernel when discussing GPR for this reason.

The kernel often typically has some free parameters (the hyperparameters). The RBF kernel for example, has two free parameters [Duv]:

$$k(x_i, x_j) = \sigma^2 exp(-\frac{(x_i - x_j)^2}{2l^2}) \tag{3.42}$$

The hyperparameter l can be considered the *length scale*. The length scale determines the smoothness of the function. A small length scales means that the function will change rapidly, and a long length scale means that points further away will also interfere, meaning we have a slower change (Figure 3.3). To find the correct hyperparameters, we need to strike a balance

[1]We project the input into some (typically) higher dimensional feature space

[2]The covariance Matrix Σ is now the covariance of the projected features

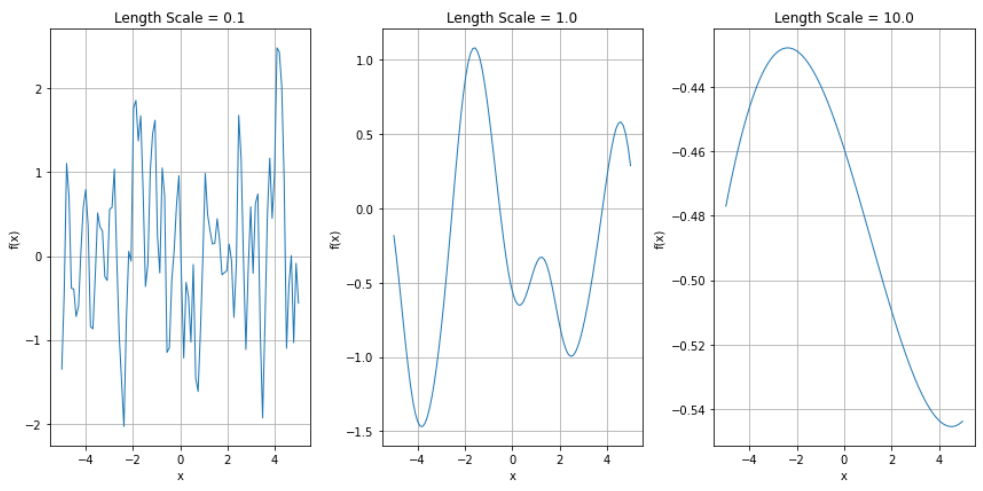


Figure 3.3: Length scale comparison using the RBF kernel using randomly sampled points. The figure is meant as an illustration, and one can imagine that the left figure is capturing noise, whereas the right one does not see the correct underlying pattern displayed in the middle plot.

between avoiding overfitting, which often occurs with too small length scales, and preventing the function from becoming overly smooth due to excessively large length scales. This balance is achieved by optimizing the marginal likelihood.

The *marginal likelihood* is the integral of the likelihood times the prior.

$$p(\mathbf{y}|X) = \int p(\mathbf{y}|\mathbf{f}, X)p(\mathbf{f}|X)d\mathbf{f} \tag{3.43}$$

If we consider the RBF kernel, then the prior is $exp(-\frac{1}{2}\mathbf{w}^T\Sigma^{-1}\mathbf{w})$, and the likelihood is $\mathcal{N}(X^T\mathbf{w}, \sigma_n^2\mathcal{I})$.

It integrates over all possible functions within our function space, allowing us to optimize the marginal likelihood over the hyperparameters. This process ideally ”selects” the correct distributions—those that best fit the labels y —by determining the most suitable hyperparameters. Since:

$$p(\mathbf{y}|X,\theta) > p(\mathbf{y}|X,\theta') \rightleftharpoons lnp(\mathbf{y}|X,\theta) > lnp(\mathbf{y}|X,\theta) \tag{3.44}$$

We can also consider the natural logarithm when finding the optimal hyperparameters $\{\sigma, l, ...\} \in \theta^3$. We call this the *log marginal likelihood*. RBF has a closed form for the log marginal likelihood given by:

$$logp(\mathbf{y}|X) = -\frac{1}{2}\mathbf{y}^T(K+\sigma_n^2 I)^{-1}\mathbf{y} - \frac{1}{2}ln|K+\sigma_n^2 I| - \frac{n}{2}ln2\pi \tag{3.45}$$

[3]Because the logarithm is a monotonically increasing function. A derivative will ’point’ in the correct direction when using gradient descent, which is ultimately what we care about

K is the kernel matrix, with entries $k(x_i, x_j)$. Finding the correct hyperparameters amounts to optimizing this function. The three terms can be interpreted as i) fitting the data to the hyperparameters, ii) complexity penalty term, and iii) is a normalization constant.

We can use gradient descent on the marginal likelihood, and then use the result of the derivative to update our weights via gradient descent ("$w^{l+1} = w^l - \eta * \partial_{\theta_j}\mathcal{L}(\theta)$") [Llo04]. Here, the loss function we use is the log marginal likelihood, and its derivative is (see Appendix:A.0.3A.0.4):

$$\frac{\partial}{\partial\theta_j} logp(\mathbf{y}|X,\theta) = \frac{1}{2}\mathbf{y}^T(K^{-1}\frac{\partial K}{\partial\theta_j}K^{-1})\mathbf{y} - \frac{1}{2}tr(K^{-1}\frac{\partial K}{\partial\theta_j}) \tag{3.46}$$

$$= \frac{1}{2}tr((\alpha\alpha^T - K^{-1})\frac{\partial K}{\partial\theta_j}), \quad \alpha = K^{-1}\mathbf{y} \tag{3.47}$$

Once the hyperparameters have been updated, we can calculate the output of the GPR. Of course, the output is another distribution, see Equation 3.41. For regression then, where the prediction needs to be a point, the *mean* of this new distribution is picked.

The K corresponds with which functions we are fitting with, and its inputs has the form of kernels: $k(\mathbf{x}, \mathbf{x}') = \Phi(\mathbf{x})^T\Sigma_p\Phi(\mathbf{x}')$. Thus, we can fit over different distributions using different kernels. Which kernels we tested can be seen in Section 4.3. It then becomes important which kernel we are considering. If the data, once projected, does not follow a RBF pattern, then fitting using RBF kernels will be difficult.

In summary, the output of Gaussian processing is a *distribution*, and for regression, the mean is picked. The distribution is given by Equation 3.41, and we find the hyperparameters using gradient descent with the loss function given by Equation 3.45. The *kernel* corresponds with which functions we believe will capture the behaviour of the *projected features* $\varphi(\mathbf{x})$.

3.1.5 Random Forest

In the following, we have used [Tre09] as our main resource.

Random Forests for our purposes is simply Forest Regression with a special initialization. So we will begin by discussing Regression Trees.

Regression Trees

To understand how a tree is grown, consider N observations $(\mathbf{x}_i, y_i)$, $\mathbf{x}_i \in \Re, \Re^p$. If we have made a partition of M regions, then the prediction from a regression tree with squared loss would be:

$$f(x) = \sum_{i=1}^{M} c_i I(x \in R_i) \quad c_i = avg(y_j | x_j \in R_i) \tag{3.48}$$

Where I is the identity function. The c_i's are found by minimizing $\sum(y_i - f(x_i))^2$ w.r.t c_i. Identifying the optimal partition across the entire dataset is computationally challenging. Instead, the approach involves incrementally building the partitions. We begin with a single partition and determine the best split by minimizing the squared loss. This process is then repeated on the resulting two partitions, creating a tree of depth two. This approach is continued until the tree reaches its maximum depth or until each leaf is pure, meaning that it contains only one class or a single value.

The first partition is:

$$R_1(j,s) = \{X|X_j \leq s\} \quad and \quad R_2(j,s) = \{X|X_j\, s\} \tag{3.49}$$

This splits one feature inside of $\mathbf{x}$ into two regions, the border of which is given by s.

We now seek the best variable j and point s that solve:

$$\min_{j,s}[\min_{c_1} \sum_{x_i \in R_1(j,s)} (y_i - c_1)^2 + \min_{c_2} \sum_{x_i \in R_2(j,s)} (y_i - c_2)^2] \tag{3.50}$$

No matter which variable $\mathbf{x}_i$ and point s we choose, the minimization inside the square brackets is given by:

$$\hat{c}_1 = avg(y_i|\mathbf{x}_i \in R_1(j,s)) \quad and \quad \hat{c}_2 = avg(y_i|\mathbf{x}_i \in R_2(j,s)) \tag{3.51}$$

For each variable, finding s is an easy task, which can be done efficiently by a computer, and scanning through the variables is also done quickly. Once these two regions have been found, we repeat this process on the resulting regions.

From this, it becomes intuitive that allowing the tree to grow too large may lead to overfitting, as it would essentially memorize each input-output pair, perfectly matching each input x_i to its corresponding output y_i. On the other hand, if the tree is restricted to a very small size, it might fail to capture important patterns in the data. To mitigate these issues, we can enforce a minimum number of nodes, n_{min}, ensuring that the tree maintains enough complexity to capture meaningful behaviors. A node, in this context, is the point at which a decision boundary is made, represented by a pair (j,s).

Random Forest

Random forest considers bootstrapped data: $\mathbf{Z}_1^*, \mathbf{Z}_2^*, ..., \mathbf{Z}_B^*$, where $\mathbf{Z}_j^* = \{(y_i, \mathbf{x}_i)\}_{i=1}^N$ (see Section 3.3 which discusses bootstrapping).

The random forest algorithm proceeds as follows:

1. **Bootstrap Sampling:**
   Draw a boostrap sample T_b (Section 3.3) from the training data.

2. **Tree Construction:**
   Make a random-forest tree by doing the following for each terminal node of the tree until the minimum size n_{min} is reached: a) Pick m random variables from $\mathbf{x}_i$. b) Pick the best split among m, as described above (Section 3.1.5). c)Split the node into two daughter nodes (two regions).

3. **Ensemble of Trees:**
   The final output of the model is the ensemble of trees $\{T_b\}_{b=1}^{B}$, where the prediction is given by the average over the predictions of all trees: $\hat{f}_{RF} = \frac{1}{B}\sum_{b=1}^{B} T_b(x)$.

In summary, the depth of each tree in the forest is crucial, as it determines the number of nodes the ensemble of trees can have. Trees that are too deep may lead to overfitting, while trees that are too shallow may underfit the data, failing to capture important patterns [Tre09].

3.2 Tested and Discarded Models

Two further modeling approaches, that initially seemed promising, but were ultimately discarded due to the underlying dataset, are discussed here.

3.2.1 Curve Fitting

When attempting to identify a model, it is logical to plot the data against time (which corresponds to LS, as described in Section 2). Such plots can reveal patterns like Figure 3.4:

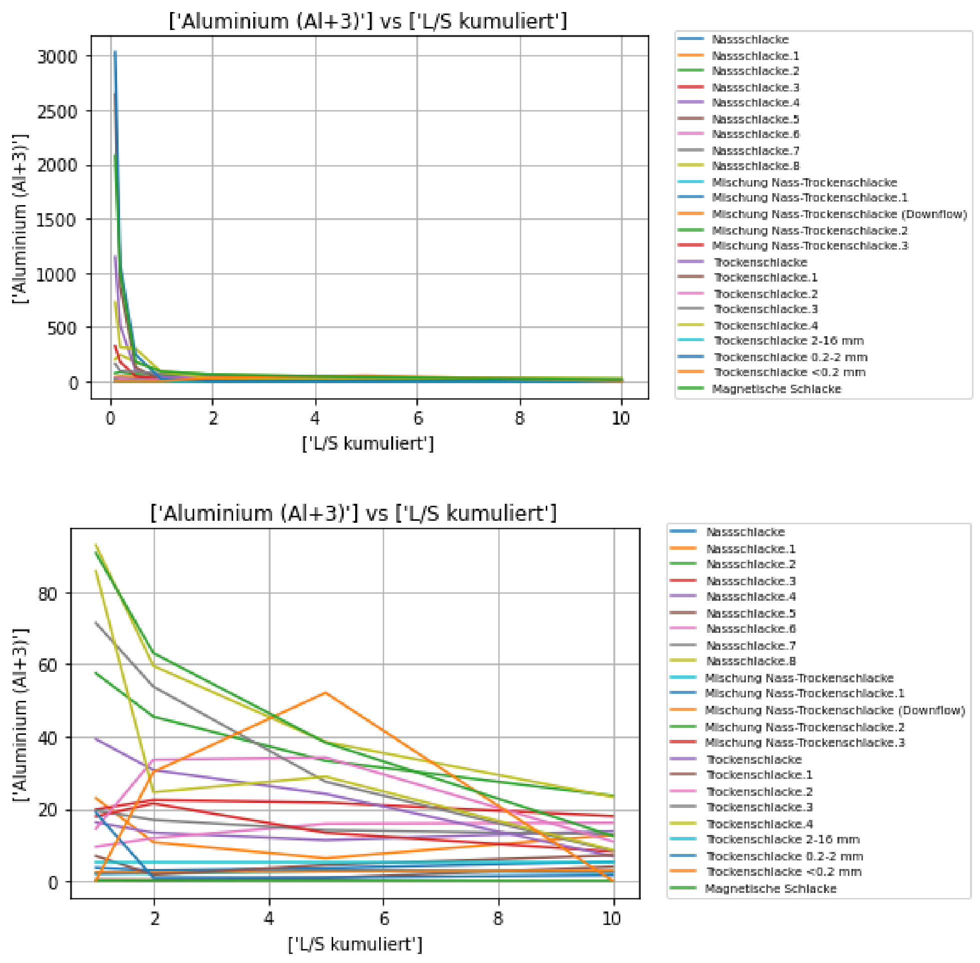


Figure 3.4: Above: Concentrations over all LS values. Below: Except for the first three. The plots for all elements can be found in the appendix A.2

Note that many of the plots do not look as "nice" as these, in the sense that not all curves tend to follow the same shape (see Figure A.2 for an overview of all elements). Even in these plots, we observe some concentrations showing upward spikes.

One approach is to use curve fitting. For example, we might assume that the concentration of Aluminium follows a negative exponential decay, described by the function:

$$Alu[LS] = Aexp(-b \cdot LS) + C \tag{3.52}$$

An attempt was made to use curve fitting for predictions. The approach involved fitting an exponential model to each of the 22 training sets using the SciPy optimization package. The average of these fitted models was then used for predictions on the test set. This method occasionally yielded good results, particularly for elements like chloride and aluminium, and for certain experiments. However, a major issue arose: in some cases, the fitted curves did not conform to an exponential pattern, as illustrated in Figure 3.4. This discrepancy led to the abandonment of this method, as this method does not have the required flexibility needed.

Despite this, using the negative exponential as a feature (Section 2.3.2) could still be beneficial for elements that do follow an exponential decay pattern. In such cases, the negative exponential feature might be significant, and would then show up in SHAP plots (Section 3.4).

Ultimately, the method was deemed too "unstable" [4]. Additionally, its simplicity limits its ability to capture interactions between different elements, making it unsuitable for our needs.

3.2.2 Sindy

Sparse Identification of Nonlinear Dynamics (SINDy) is another method explored for predictions. SINDy is based on the approach outlined in the paper by Brunton et al. [BPK16]. It frames the prediction problem as one of solving an underlying differential equation. The core idea is to use sparse regression techniques, such as Lasso optimization, to identify the differential equations governing the dynamics of the system.

In practice, SINDy involves the following steps:

1. **Formulation:** Construct a library of possible terms (functions of the input features) that might appear in the differential equations.
2. **Sparse Optimization:** Use Lasso regression to identify the most relevant terms from this library that describe the system's dynamics.
3. **Integration:** One the differential equations are identified, integrate them to make predictions.

This approach offers a way to capture the underlying dynamics of the system through differential equations, potentially improving predictions by leveraging the structure of the problem. For further details, refer to the foundational paper by Brunton et al. [BPK16].

Sindy assumes that the dynamical system (here: the concentrations) has the form:

$$\frac{d}{dt}\mathbf{x}(t) = \mathbf{f}(\mathbf{x}(t)) \tag{3.53}$$

[4]For instance, chloride sometimes achieved an R^2 value of 0.98 but other times produced values as low as -50.

We can numerically estimate the derivatives between two points. For example, the derivative for Calcium at LS=0.1, will be the difference between its concentration at LS=0.1 and LS=0.2 divided by $\Delta LS = 0.1$*. If the derivatives are calculated like this, our problem can be readily transformed into the framework of sindy.*

Consider **X** as our feature matrix, so the first row contains the features of the first experiments. The second row the features of the second experiment, and so on:

$$\mathbf{X}_{ij} = \mathbf{x}_i^j \tag{3.54}$$

Here, $j \in [1, 2, 3, ..., 23]$ represents the index of the experiment, and each $\mathbf{x}^j$ is a feature vector in $\Re^n$, where n is the number of features. Therefore, the matrix **X** has dimensions $m \times n$, with m the total number of experiments and n the number of features. This is then projected in the following way:

$$\Theta(\mathbf{X}) = [1, \mathbf{X}, \mathbf{X}^{P_1}, \mathbf{X}^{P_2} ..., sin(\mathbf{X}), ...] \tag{3.55}$$

This typically includes features like a constant, the features themselves, sine of the features, polynomials of the features such as $x_i^j * x_k^l \in \mathbf{X}^{P_2}$ and so on. These functions constitute the potential candidates for the nonlinear dynamics of our system. The goal is to assign weights to these functions to determine which nonlinearities are active in describing the system. This process is analogous to feature selection in Lasso regression. Specifically, we must search for the weights in $\Xi = [\xi_1, ..., \xi_n]$. We set out to solve the following problem:

$$\dot{\mathbf{X}} = \Theta(\mathbf{X})\Xi + \eta\mathbf{Z} \tag{3.56}$$

Where **Z** contains Gaussians with zero mean. Given the importance of rate equations which often take the form $v_0 = k[A]^x[B]^y$, in describing chemical changes in concentrations, it is reasonable to expect that second-order polynomials might carry significant weights. Specifically, in rate equations involving two elements, we would anticipate that terms such as:

$$w_{ij} \cdot x_i \cdot x_j \quad for \quad i \neq j \quad with \quad w_{ij} \neq 0 \tag{3.57}$$

to have non-zero weights w_{ij}. This is because such terms capture the interaction between different elements, which is critical in modeling chemical reactions. If additional elements are involved, third-order polynomials like $[A]^x[B]^y[C]^z$ and higher-order terms would also be relevant.

Sindy faced similar issues to curve fitting in terms of instability. Specifically, it often encountered convergence problems during integration, making it generally unsuitable for our purposes.

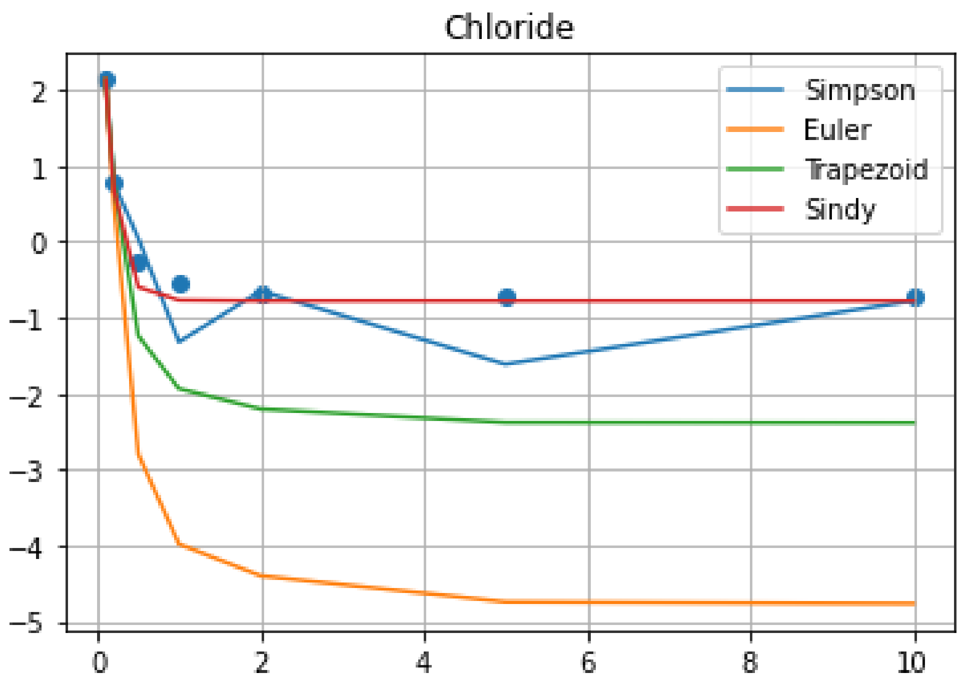


Figure 3.5: Using Sindy: Fitting example for different integrators. This is a fit on the same data on which it was trained. In this example, Sindy's LSODA integrator outperforms the other integrators. Using this to predict new data (not same data) yielded poor results.

In Figure 3.5, an example of using Sindy is illustrated. It is important to note that in this figure, the solution to the differential equation is integrated on the same dataset used to derive the differential equation. For proper testing, the integration should be performed on new, unseen data while utilizing the previously determined weights.

3.3 Bootstrapping and Cross-Validation

In this section we will describe how Bootstrapping and Cross-Validation works. The main source for the following section can be found in [TF01]. Refer to this book for further details, the following is intended as a reminder, and details, e.g. drawbacks of the methods are not discussed here.

3.3.1 K-Fold Cross-Validation

Many of our methods, including Ridge, Lasso and the Neural Network uses Cross-Validation (CV) for estimating the hyperparameters. We split the data into K (=5) folds as shown in Figure 3.6:

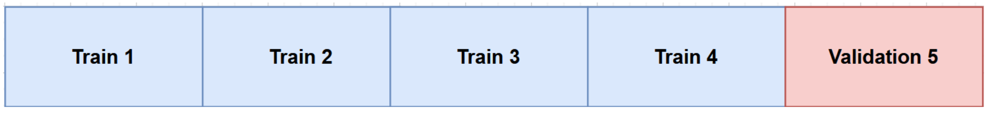


Figure 3.6: The set is split into (for K folds = 5) 5 roughly equal-sized parts; We fit to K-1 folds, and calculate prediction error on the last fold. Made using [Aut24a]

We train on K-1 folds, and predict on the left-out fold. We do this K times, leaving out the kth fold for k = 1,2,...,K. Combining the results yields the average prediction error (Equation 3.58):

$$CV(\hat{f}) = \frac{1}{N}\sum_{i=1}^{N}\mathcal{L}(y_i, \hat{f}^{\neg k(i)}(x_i)) \tag{3.58}$$

Where $\hat{f}$ is the predictor trained on the training data ¬k(i), $\mathcal{L}$ is the loss function (often the mean-squared error: $\mathcal{L}(\mathbf{y}, f(\mathbf{X})) = \frac{1}{N}\sum_{i=1}^{N}(\mathbf{y}_i - f(\mathbf{X}_i))^2$.

So for a function tuned on the training set $\hat{f}$, we can calculate the prediction error from this function on the set that was left out. If we let K=N, then this error is known as the *leave-one-out* cross-validation. Note that k(i) often contains more than one point (for K=5, k(i) contains roughly 80% of the set, and ¬ k(i) contains roughly 20% of the set).

In the end, we will have K models trained like this. The output is then given by the average of these models:

$$f(x) = \frac{1}{K}\sum_{i=1}^{K}\hat{f}^{i}(x) \tag{3.59}$$

3.3.2 Bootstrap

The Random Forest method employs bootstrapping to estimate hyperparameters.

Bootstrapping is akin to cross-validation in that it involves evaluating prediction error using sub-samples of the original dataset. Instead of simply splitting the training data, bootstrapping generates B *bootstrap amples*, which are copies of the original set with replacement. This means that some data points may appear multiple times in a bootstrap sample, while others might not appear at all.

Statistical analysis, fitting, and error estimation are then performed on these B bootstrap samples. To mitigate issues arising from the randomness of this process—such as identical points appearing frequently in the bootstrap samples—it is advisable to use a large number of bootstrap samples (e.g., B=100). This ensures that the Law of Large Numbers (LLN) can stabilize the estimates and reduce variance in the results.

In Figure 3.7, bootstrapping is illustrated. Note that some features may be repeated in each sample, but the overall statistics should resemble the original dataset, so that the total training data is not skewed.

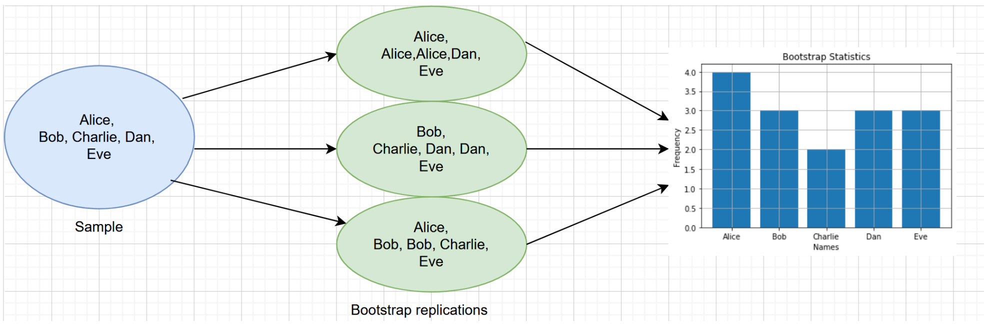


Figure 3.7: Illustration of boostrapping. Note that features can be repeated in the samples. Made using [Aut24a].

The output of the model will be the output of the average over all B models:

$$f(x) = \frac{1}{B}\sum_{i=1}^{B}\hat{f}^{i} \tag{3.60}$$

3.4 Feature importance

To identify important features, we use SHapley Additive exPlanations (SHAP), a method grounded in game theory [Lun23]. SHAP provides a way to explain a model's output by evaluating the impact of each feature on the model's predictions.

The core idea behind SHAP is to assess how the removal of a feature affects the model's output. For each feature $\{i\}$, the model is retrained on all subsets of features $S \subseteq F$, where F represents the set of all features.

We compare the predictions from two models: one with the feature $\{i\}$ included $f_{S\cup+\{i\}}$ and one with the feature $\{i\}$ excluded f_S.

The difference in predictions, $f_{S\cup\{i\}}(x_{S\cup\{i\}}) - f_S(x_S)$, quantifies the impact of the feature $\{i\}$ on the output, where x_S denotes the input corresponding to the feature subset x_S.

Since the effect of withholding one feature affects other features in the model, the preceding differences are computed for all possible subsets $S \subseteq F\backslash\{i\}$. The Shapley values are then computed using a weighted average over all possible differences [LL17]:

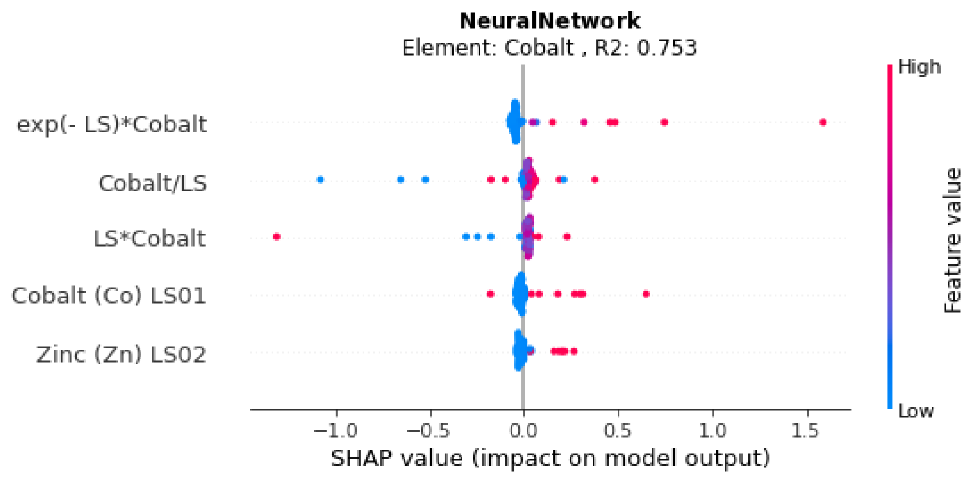


Figure 3.8: SHAP for NN cobalt

$$\varphi_i = \sum_{S \subseteq F \ \{i\}} \frac{|S|!(|F| - |S| - 1)!}{|F|!} [f_{S \cup \{i\}}(x_{S \cup \{i\}}) - f_S(x_S)] \tag{3.61}$$

In the results section, we may encounter a figure like Figure 3.8):

The order of features in a SHAP plot indicates their importance. For example, in Figure 3.8, the feature exp(-LS)*Cobalt ranks as the most important. In SHAP plots, red points represent higher feature values, while blue points represent lower ones. The x-axis shows whether changes in feature values lead to an increase or decrease in the model's output.

In this specific example, higher values of exp(-LS for Cobalt) (which correspond to lower LS values) generally lead to larger output values. This suggests that the concentration of Cobalt decreases as the LS value increases, reflecting a negative relationship between LS and Cobalt concentration over time.

A similar behavior is observed for the feature LS*Cobalt. Here, larger LS values result in a decrease in Cobalt concentration, aligning with the trend observed in exp(-LS for Cobalt).

3.5 Performance Metrics

To evaluate the performance of predictive models, several metrics are commonly used. Each metric provides different insights into the accuracy and quality of predictions. Below, we describe some widely used metrics

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{3.62}$$

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \overline{y}_i)^2} \tag{3.63}$$

$$MAPE = \frac{1}{n}\sum_{i=1}^{n}|\frac{y_i - \hat{y}_i}{y_i}| \tag{3.64}$$

$$MAPE^0 \underset{y_i \neq 0}{=} \frac{1}{n}\sum_{i=1}^{N}|\frac{y_i - \hat{y}_i}{y_i}| \tag{3.65}$$

We will be using MAPE for measuring performance. It was used for tuning hyperparameters, and to find the correct scaling.

1. **Root Mean Squared Error (RMSE)**:
   RMSE measures the square root of the average squared differences between observed and predicted values. It is straightforward to interpret but does not provide a clear benchmark for what constitutes a "good" RMSE value. Lower RMSE values indicate better fit, but what is considered acceptable can vary depending on the context.

2. **R-squared (R^2)**:
   R^2 represents the proportion of variance in the dependent variable that is predictable from the independent variables. An R^2 value of 1 indicates that the model perfectly explains the variance in the data, while a value of 0 means that the model explains none of the variance. It is useful for understanding the proportion of variability captured by the model.

3. **Mean Absolute Percentage Error (MAPE)**:
   MAPE calculates the average percentage error between observed and predicted values. It is easy to interpret as it expresses error in percentage terms. However, MAPE can be problematic with very small actual values, as it can produce extremely large percentage errors.

4. **MAPE^0**:
   The $MAPE^0$ metric is similar to MAPE but is only defined where the observed value y_i is non-zero. It measures the average absolute error relative to the actual values, providing insight into the accuracy of predictions in proportion to the observed values.

For this analysis, we primarily use MAPE to measure performance. MAPE was utilized both for tuning hyperparameters and for determining the appropriate scaling of our models. This choice is motivated by its interpretability and its focus on relative errors, although we acknowledge its limitations, especially with very small values.

Chapter 4

Feature Inputs & Training

In the following, it is assumed that the reader knows what the inputs and methods are. This is described in Chapter 2 and Chapter 3, respectively. This means knowing e.g. what is meant by the kernel of GPR, and the learning rate of a neural network. Here the choice of these terms is discussed. The training of the models, the splitting of the data is also made more explicit.

4.1 Implementation

The programming software used was Python 3.1. Small training tasks were done on a personal laptop, and bigger training tasks were done on ETH's Euler. Euler is ETH's computer cluster, `https://scicomp.ethz.ch/wiki/Euler`, during training Euler VI was used and contains 216 computer nodes each equipped with:

1. Two 64-core AMD EPYC 7742 processors (2.25 GHz nominal, 3.4 GHz peak)
2. 512 GB of DDR4 memory clocked at 3200 MHz
3. Local scratch: 920,618.0 MB

All nodes were connected together via a dedicated 100 Gb/s InfiniBand HDR network. A list of libraries and their uses can be found in Table 4.1.

Library	Use and Source
numpy	Mathematical operations https://numpy.org/
pandas	Reading and saving CSV files https://pandas.pydata.org/
torch	Machine learning models https://pytorch.org/
shap	Sensitivity analysis https://shap.readthedocs.io/en/latest/
matplotlib	Plot creation https://matplotlib.org/
sklearn	Data processing, metrics, and more https://scikit-learn.org/stable/
difflib	Naming and feature extraction https://docs.python.org/3/library/
scipy	Curve fitting https://scipy.org/
seaborn	Correlation plots https://seaborn.pydata.org/
joblib	Model saving https://joblib.readthedocs.io/en/stable/
pysindy	Prediction method (not in results; discontinued) https://pysindy.readthedocs.io/en/latest/
sys, random, re, os, inspect	Miscellaneous utilities https://docs.python.org/3/library/

Table 4.1: Libraries and their main uses. Sources last accessed: 26/08/2024.

4.2 Training

The training process of the models was conducted iteratively for each scaling method, input method, and element (details in Section 4.3). After completing the preprocessing (Section 2), the following steps were executed (see Flowchart: Figure 4.1):

1. **Experiment Selection:**
   One of the 23 experiments was held out for testing, leaving 22 experiments for training.

2. **Cross-Validation:**
   The 22 experiments were split into five folds for cross-validation, used to train hyperparemeters as outlined in Section 3.3. This process yielded five models, each trained on different parts of the training data, with

20% left out for validation.
Additionally, the splitting of the folds was done for five different seeds. This is done to reduce the influence of randomness in the training data.

3. **Iteration Across Experiments:**
   Steps 1 and 2 were repeated for all 23 experiments, allowing for a robust assessment of model generalizability given the small dataset and potential high deviations (see Figures 2.1 and A.2). In total, 575 models were trained for each configuration (25 models per experiment x 23 experiments).

4. **Parameters Variation:**
   Steps 1-3 should be repeated with different preprocessing settings (e.g scaling, noise injection as discussed in Section 2.3.3, inputs (Section 2.3.2), and model configurations (e.g., regularization parameters for Ridge and Lasso, tree depth for Random Forests, kernel selection for Gaussian Process Regression, and architecture the for Neural Networks) for all elements. Due to the time intensity of doing this, some simplifications for the Configuration Optimization had to be made. This is described in Section 4.3.

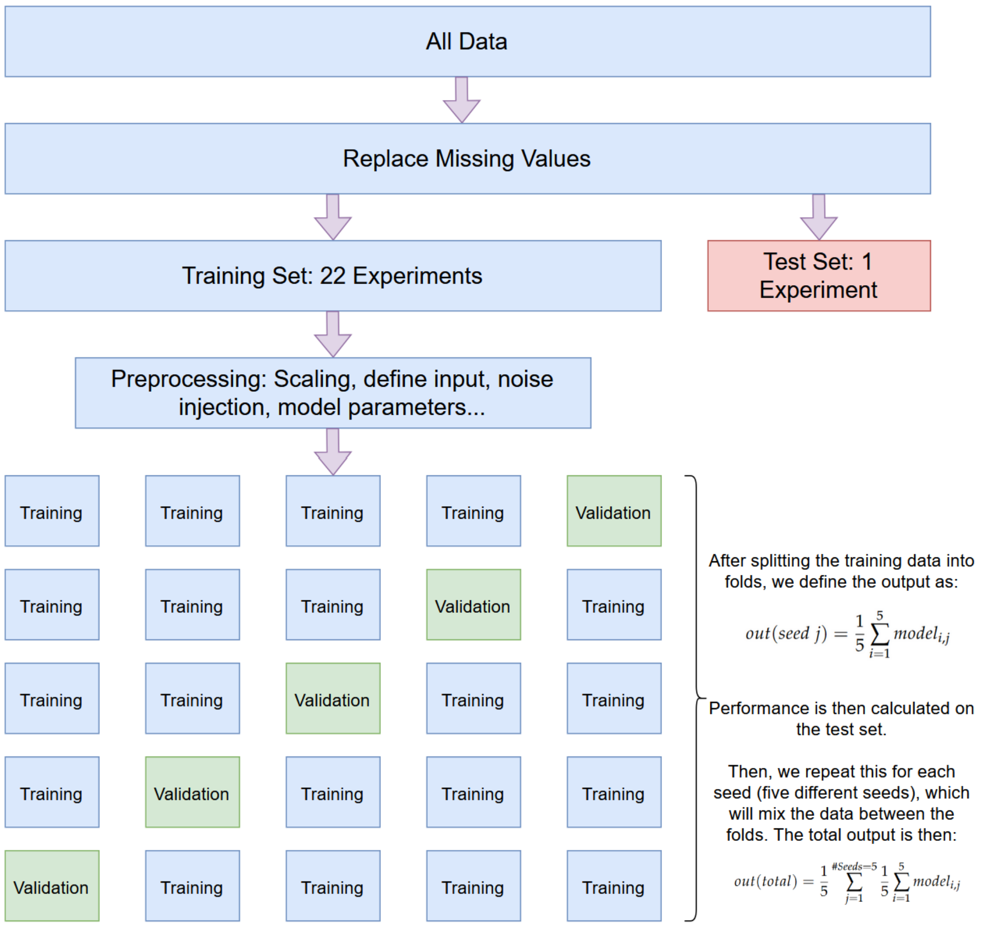

All Data
Replace Missing Values
Training Set: 22 Experiments
Test Set: 1 Experiment
Preprocessing: Scaling, define input, noise injection, model parameters...
Training
Validation
After splitting the training data into folds, we define the output as:
$out(seed\ j) = \frac{1}{5}\sum_{i=1}^{5} model_{i,j}$
Performance is then calculated on the test set.
Then, we repeat this for each seed (five different seeds), which will mix the data between the folds. The total output is then:
$out(total) = \frac{1}{5}\sum_{j=1}^{\#Seeds=5} \frac{1}{5}\sum_{i=1}^{5} model_{i,j}$

Figure 4.1: Flowchart illustrating the training process.

4.3 Optimization of Training Configuration and Input Features

Several key decisions were made regarding input and configuration, as detailed below (see Figure 4.2 for a visual summary). In the following, the ”best” model is chosen w.r.t. the performance metric given by: $MAPE^0_{y_i \neq 0} = \frac{1}{N}\sum_{i=1}^{N} |\frac{y_i - \hat{y}_i}{y_i}|$.

1. **Element Selection:**
   Only Calcium was used to find the best input features and training configurations due to its relatively stable data and absence of missing values (Table 2.1).

2. **Seed selection**:
   To expedite the input parameter search, we used only one seed, result-

ing in five models per test experiment rather than 25.

3. **Method Selection:**
   A single method, Neural Networks, was chosen for optimizing input parameters across all methods to maintain consistency. Neural networks was chosen due to its more promising results during test-runs (with an architecture of [120] (Section 4.5). This might not be ideal for all methods (e.g., GPR's performance could suffer in higher-dimensional spaces, for which NN is well-suited).

4. **Input feature combinations:**
   Five different configurations were tested using Neural Networks:
   a. **Minimal input:** Calcium at LS = 0.1 and the target LS value.
   b. **Extended Input:** Calcium at LS = 0.1, 0.2, and 0.5, plus the target LS value.
   c. **Multi-Element Input:** Calcium and all other elements at LS = 0.1, 0.2, 0.5 plus the target LS value (input dimension: 17*3 + 1 = 52).
   d. **Feature-Enhanced Input:** The above, plus three engineered features (see Section 2.3.2) (input dimension: 55)
   e. **Smart Input:** First three LS values of elements strongly correlated with the target, (Section 2.3.2)
   The standard scaler was used for all these configurations, without any jittering (noise injection). The best input feature combinations was d; Feature-Enhanced Input.
   The results were for $MAPE^0$: The results were (using standard scaling) $MAPE^0$:

   a) Minimal input = 1.29
   b) Extended Input = 0.95
   c) Smart Input = 0.68
   d) Multi-Element input = 0.87
   e) **Feature-Enhanced input = 0.68**

5. **Scaler Optimization:**
   The best-performing input combination was tested with four different scalers: no scaling, logarithmic scaling, robust scaling, standard scaling and MinMax scaling (Section 2.3.2). The results were for $MAPE^0$:

   a) **Robust = 0.59**
   b) Standard = 0.68
   c) Logarithmic = 1.49
   d) MinMax = 1.32

   Robust scaling worked the best,

6. **Noise Injection Testing:**
   The effect of noise injection was then evaluated by training models with no noise, and with Gaussian noise injection ($\sigma = [0.1, 0.2, 0.3, 0.4, 0.5, 1]$, N=5). Models trained without noise injection were found to perform better. The results were ($MAPE^0$):

   a) **No noise = 0.78**

   b) $\sigma = 0.1$ = 1.12

   c) $\sigma = 0.2$ = 1.03

   d) $\sigma = 0.3$ = 1.00

   e) $\sigma = 0.4$ = 0.93

   f) $\sigma = 0.5$ = 0.87

   g) $\sigma = 1$ = 0.9

7. **Final Training:**
   With the optimal input combination, scaler and noise settings established, training was performed for all methods. The final input comprised 55 data points (15 concentrations, pH and Eh at three LS values plus four engineered features) with robust scaling and without noise injection.

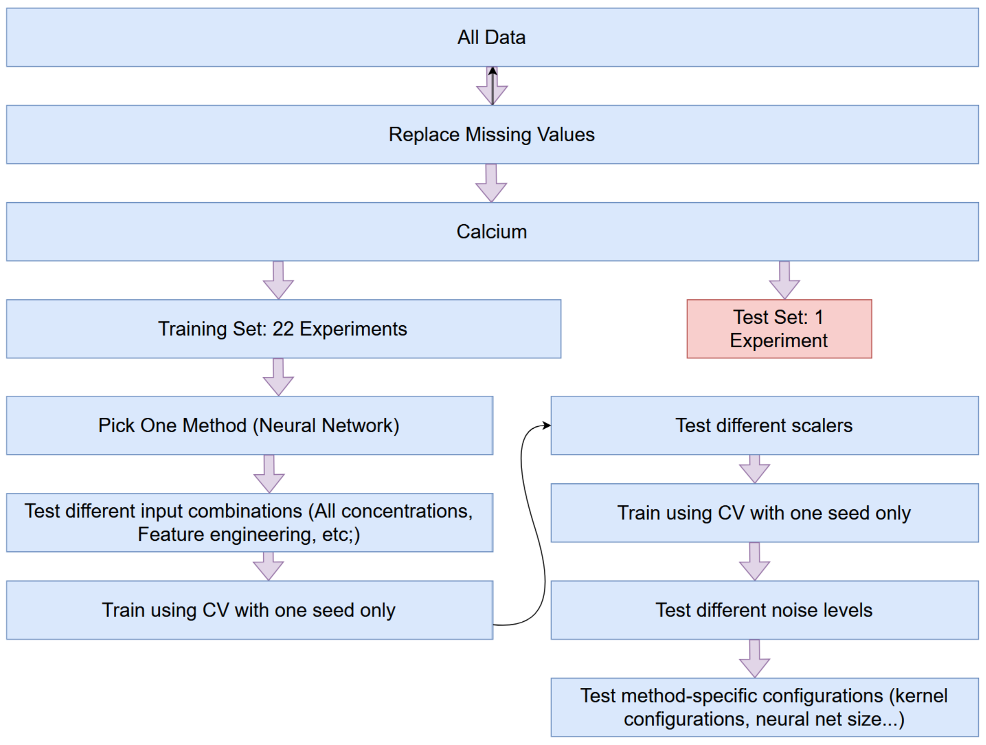

All Data
Replace Missing Values
Calcium
Training Set: 22 Experiments
Test Set: 1 Experiment
Pick One Method (Neural Network)
Test different input combinations (All concentrations, Feature engineering, etc;)
Train using CV with one seed only
Test different scalers
Train using CV with one seed only
Test different noise levels
Test method-specific configurations (kernel configurations, neural net size...)

Figure 4.2: Flowchart describing the process of finding the correct input configurations/pre processing settings. Made using [Aut24a]

4.4 Gaussian Process Regression

Continuing from Section 4.3, the primary method for refining the Gaussian Process Regression (GPR) model involves selecting an appropriate *kernel*. As outlined in Section 3, the effectiveness of GPR depends heavily on how well the chosen kernel can capture the correlations among input elements. For instance, if the relationship between data points resembles an exponential decay, a Gaussian kernel is likely to perform well.

Several kernels were tested during the training process, although many faced convergence challenges, a common issue when working with high-dimensional inputs in sklearn's implementation of GPR [lea23]. The kernels tested included options from the sklearn library, with a comprehensive list available in their documentation [Sci24a] .

The following kernels were considered:

$$Constant\ kernel = k(x_i, x_j) = const \quad \forall x_i, x_j \tag{4.1}$$

$$Dot\ Product = k(x_i, x_j) = \sigma_0^2 + x_i x_j \tag{4.2}$$

$$Matern = k(x_i, x_j) = \frac{1}{\Gamma(\nu)2^{\nu-1}}(\frac{\sqrt{2\nu}}{l}d(x_i, x_j))^{\nu}K_{\nu}(\frac{\sqrt{2\nu}}{l}d(x_i, x_j)) \tag{4.3}$$

$$RBF = k(x_i, x_j) = exp(-\frac{d(x_i, x_j)^2}{2l^2} \tag{4.4}$$

$$Rational\ Quadratic = k(x_i, x_j) = (1 + \frac{d(x_i, x_j)^2}{2\alpha l^2})^{-\alpha} \tag{4.5}$$

$d(x_i, x_j)$ is the euclidean distance, K_ν is a modified bessel function, σ_0^2 is a noise term, ν,α, and l are hyperparameters, and Γ is the gamma function.

These combinations were tested:

1. Matern 0.99
2. RBF = 0.99
3. Matern + Constant kernel = 2.08
4. Constant kernel * Matern = 0.99
5. Constant kernel * Dot Product + Constant kernel + Constant kernel * RBF = 2.04

More complex kernels lead to convergence issues, and were generally performing worse. RBF worked the best, measured using $MAPE^0 \underset{y_i \neq 0}{=} = \frac{1}{N}\sum_{i=1}^{N}|\frac{y_i - \hat{y}_i}{y_i}|$.

The results of the best performing kernel are discussed in the results section: 5.

4.5 Neural Network

Neural nets have perhaps the widest variety of hyperparameters. The *architecture* of the network can have many different forms.

Let us begin with the choice of loss function. Usually, the MSE loss is chosen for neural networks when working with regression tasks $MSE = \frac{1}{N}(\sum_{i=1}^{N}(f(x_i) - y_i)^2$. It is worth noting that this loss will be very *outlier susceptible*, meaning that large y_i's will have a large negative impact on the model parameters (if these y_i's are indeed outliers). If they are not, then we might want to minimize exactly the squared error.

A problem encountered when using MSE loss was exploding gradients [Gir15]. This is sometimes solved when using the L1 loss, or the smooth loss

[Con24]. The smooth loss is given by [Con24] (see equation A.0.5):

$$\mathbf{L1} = \sum_{i=1}^{N} |y_i - \hat{y}_i| \tag{4.6}$$

$$\textbf{Smooth L1} = \sum_{i=1}^{N} \mathcal{L}(x_i, y_i) = \sum_{i=1}^{N} \begin{cases} 0.5(y_i - \hat{y}_i)^2/\beta, & if \quad |y_i - \hat{y}_i| < \beta \\ |y_i - \hat{y}_i| - 0.5 \cdot \beta, & else \end{cases} \tag{4.7}$$

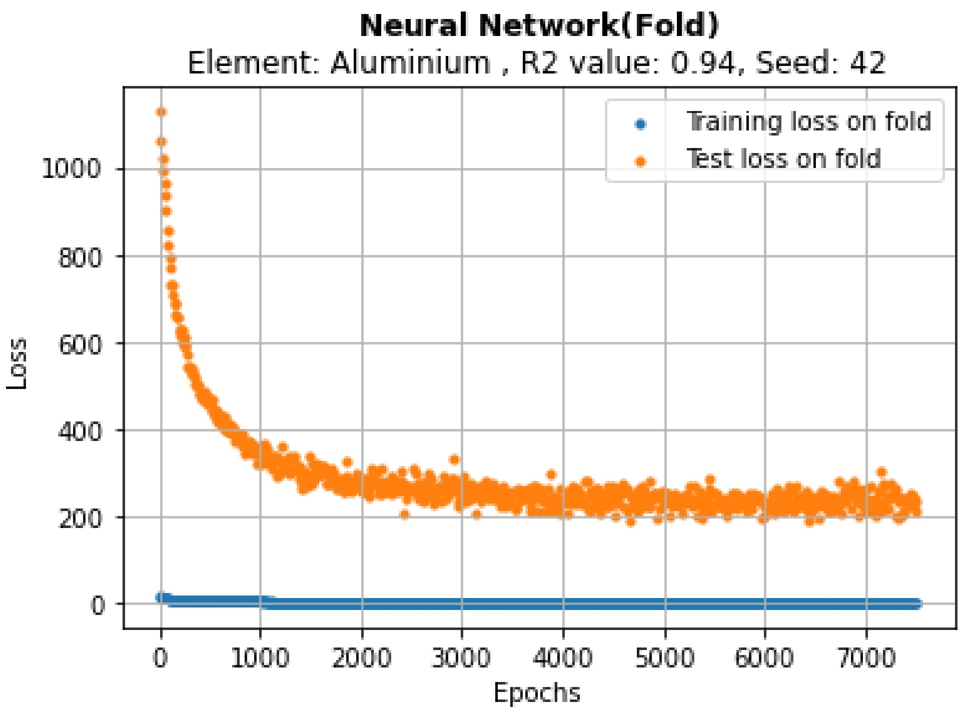
The smooth loss is similar to the L1 loss for large errors, and the MSE loss for small errors.

Because of the exploding gradients leading to infinites during training, the smooth L1 loss was used.

The learning rate was chosen to be $\eta = 0.001$. Smaller learning rates lead to very slow convergence, and higher learning rates lead to convergence problems. Often resulting in infinities and poor performance.

Finding the correct amount of epochs (that is, iterations where we update the model weights (Section 3.1.3) is not always a clear cut task. Too many epochs are time consuming, and may lead to overfitting [GBC16], whereas too few will not let the model train enough.

When searching for the best performing model, the models' performance on the CV set was measured, and the best performing model one on this set was used. If during training, the model is still converging on this CV set during its last 10% of total epochs, the total amount of epochs was increased, if not, training was stopped. This is called early stopping [GBC16]. To estimate the starting number of epochs, 7500 epochs was picked. This is based on training observations (Figure 4.3).



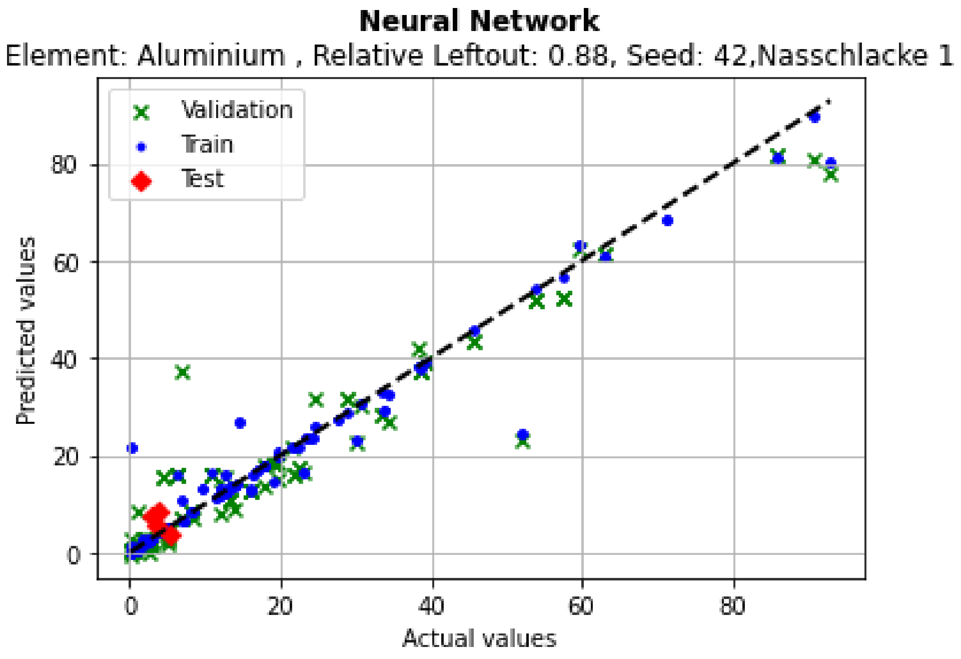


Figure 4.3: Above: Convergence of the model, measured for one particular CV fold using NN architecture [145]. Due to lack of convergence, the total amount of epochs is not increased more than the starting amount of 7500 epochs. R^2 is measured on the validation set. Below: Predicted vs Observed values. $MAPE^0$ is measured on the 4 test points, corresponding with "Relative Leftout".

A key consideration that warrants discussion is the selection of the neural network architecture. The distinction between deep and shallow neural networks is well-known, with deep networks characterized by a greater number of hidden layers, while shallow networks contain fewer layers [GBC16]. Determining the optimal architecture is a complex and often time-consuming process. Here, a systematic approach was employed to address this challenge. With the preprocessing steps, learning rate, number of epochs, loss function, and other hyperparameters already determined, the focus was placed solely on optimizing the network architecture. Specifically, the input parameters were held constant, while various architectural configurations were evaluated.

A few different architectures were tested in the beginning with varying degrees of success, but adding dropouts for regularization [GBC16] helped with overfitting, and adding the previously discussed ELU activation function (Chapter 3), with "(-x)" as input worked well. In the end, each hidden layer had a number of nodes, followed by a leakyRELU function:

$$LeakyRELU(x) = \begin{cases} x, & if x \geq 0 \\ \alpha * x & otherwise \end{cases} \tag{4.8}$$

Using $\alpha = 0.01$, the issue of dying neurons, which impeded training convergence, was effectively resolved. Each layer followed this activation strategy with the Exponential Linear Unit (ELU) function applied to -x, followed by a 30% dropout. The architecture of each layer was consistently structured in this manner. The input layer was configured with a direct mapping from the input shape to the hidden layer shape. In subsequent discussions, a specific architecture will be denoted by its layer widths, such as [120,80]. This notation indicates a network with two hidden layers, comprising 120 and 80 neurons respectively, each utilizing the specified activation functions and dropout rates. In Figure 4.4 this has been illustrated.

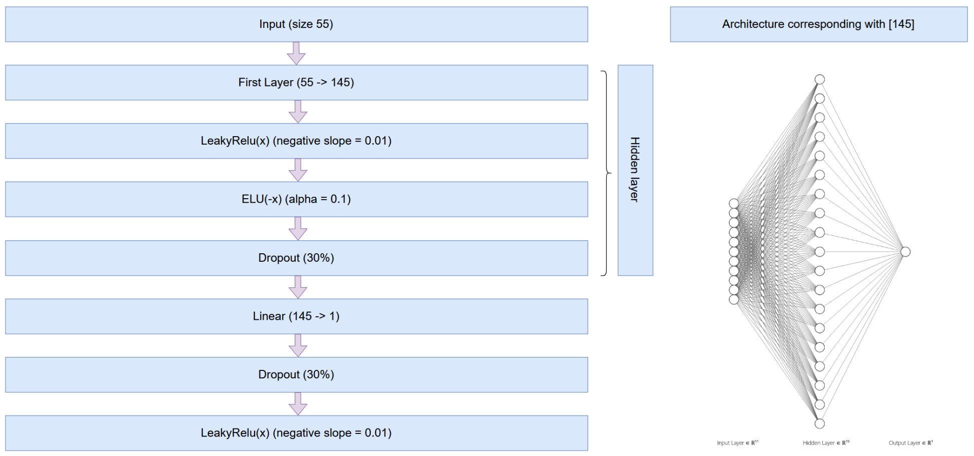


Figure 4.4: One hidden layer. The sizes of the input layer and hidden layer have been divided by five for readability. Adding more hidden layers will result in the "hidden layer block" being stacked below each other with different widhts. But the first and last three boxes are always there, regardless of the chosen architecture. Thus, this corresponds with an architecture denoted by [145], as the entire network can be described by this number. Made using [Aut24a],[Aut24b]

This was then tested with many different architectures. Two grid searches were done, one search considered widths from 20 to 120 with stepsizes of 20, and a depth of 5 e.g. [20],[40],[60],[80],[100],[120],[20,20],[40,20],..., [120,120,120,120,120]. Another grid search was a fine grid search from [105] to [150] with stepsizes of 5, once the results of the rough searches showed that the best performances were for shallow nets (depth less or equal to two). The results of which can be seen in Figure 4.5 and Figure ??.

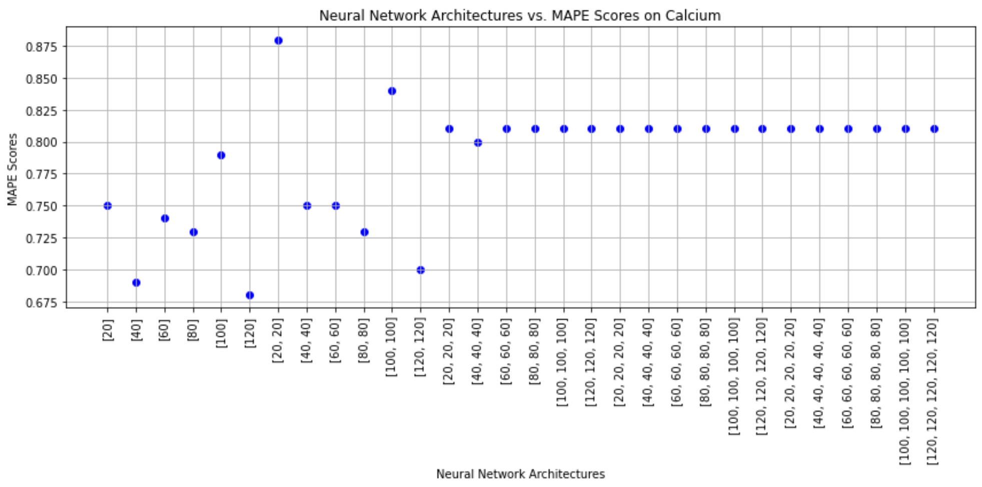


Figure 4.5: Results from a rough grid search. The MAPE values are evaluated on the cross validation set. Finer grid searches around 120 was done, resulting in an architecture of [145]. Note that deeper neural nets (more than 2 hidden layers) leads to convergence issues.

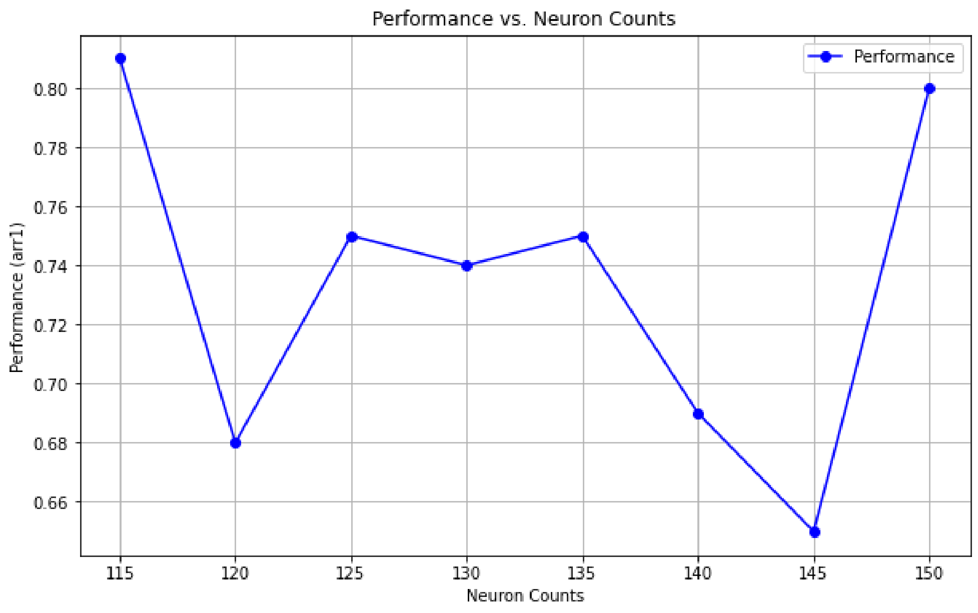


Figure 4.6: Results from finer search

In summary then, the neural network had the architecture described in figure 4.4, with a learning rate of 0.001, a smooth L1 loss, as well as at least 7500 epochs.

4.6 Random Forest

As previously discussed (Section 3), the performance of a Random Forest model is significantly influenced by its depth. Allowing for greater depth increases the risk of overfitting. The number of trees in the forest is also critical, as is the choice of loss function used to determine feature splits. In this study, the number of trees was set to 200, and the depth of each tree was unrestricted, allowing them to grow until each leaf node was pure (i.e., containing only one feature). While a larger number of trees can mitigate overfitting, they do increase computational cost [TF01].

Given the high dimensionality of the input space, a critical concern arises if many input features do not correlate strongly with the output. When the number of features selected per bootstrap sample (m) is small, there is a risk that the selected subsets may lack sufficient correlation with the target variable. This issue is particularly pronounced when m is small. To address this, m was set to the maximum number of features. However, it is important to note that this does not guarantee that each bootstrap sample will include m distinct features, as there is a possibility of the same feature being selected multiple times. The probability that a sample contains every feature is:

$$\frac{55!}{55^n} \approx 2.4 \cdot 10^{-23} \tag{4.9}$$

However, we can consider the probability of one specific sample not appearing at all. This is:

$$(1 - \frac{1}{55})^{55} \approx 0.36 \approx 1/e \tag{4.10}$$

So the probability of a specific sample appearing is then:

$$(1 - 1/e) \tag{4.11}$$

We can multiply this by the total amount of samples (55) which gives us ≈ 35. This means that on average, 35 unique features will show up in each bootstrap. If we assume that at least the three concentrations and the four feature engineering features are important (the concentrations should be important, [Ers+23]), then the probability of none of these showing up at all is small:

$$(1 - \frac{7}{55})^{55} << 1 \tag{4.12}$$

In summary, the depth of each tree was configured to allow the leaves to reach purity. The number of estimators was set to 200, and under the chosen settings, the average number of distinct features selected was 35. It is important to note that while a deeper tree may increase the risk of overfitting, this effect is mitigated by the large number of trees (estimators) in the forest, which collectively smooth out potential overfitting [TF01].

4.7 Ridge & Lasso

The primary concern with Ridge and Lasso regression methods is the choice of the regularization parameter λ. This parameter controls the strength of the regularization applied to the model, impacting the magnitude of the coefficients estimated by these methods.

The objective functions for Ridge and Lasso regression are as follows (recall Chapter 3):

$$\hat{\beta}^{ridge} = \underset{\beta}{argmin}[\sum_{i=1}^{N}(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j) + \lambda \sum_{j=1}^{p} \beta_j^2] \tag{4.13}$$

$$\hat{\beta}^{lasso} = \underset{\beta}{argmin}[\frac{1}{2}\sum_{i=1}^{N}(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j) + \lambda \sum_{j=1}^{p} |\beta_j|] \tag{4.14}$$

In these equations:

1. For Ridge regression, λ is applied to the squared magnitude of the coefficients, which helps to prevent them from becoming too large. Increasing λ will shrink the coefficients more, effectively regularizing the model and reducing overfitting. Conversely, smaller values of λ allow for larger coefficients and potentially more complex models.
2. For Lasso regression, λ is applied to the absolute values of the coefficients. Lasso has the added benefit of performing feature selection by driving some coefficients exactly to zero, thus excluding some features from the model, because it penalizes small coefficients more than Ridge does. Similar to Ridge, increasing λ will shrink the coefficients more and reduce overfitting, but with the additional effect of potentially zeroing out some coefficients.

To determine the optimal value of λ, the models were trained over a range of λ's, each evaluated using Cross-Validation. We used $\lambda \in [0.1, 100]$, and it was generally effective. There are cases where λ is outside this range, for instance when increasing the range, it might go up to 10^3 instead of being limited to 100. Such extreme values frequently resulted in poor model performance or convergence issues, indicating that the range used should be carefully selected based on the data and model requirements.

Chapter 5

Results

In this chapter, we present the results organized into tables, there is one table per model used. Models with the best $MAPE^0$ performance among the other methods on the test data are highlighted in **bold**. The performance of these models is illustrated through their actual vs. predicted plots, as well as their SHAP plots, and also their Relative errors by experiment, which are included at the end of this chapter.

We also provide the R^2 values and Mean Absolute Percentage Error (MAPE) values. These performance metrics are computed from various R^2 and MAPE scores obtained during training. Due to the variability introduced by different random folds during training, high standard deviations could indicate a strong dependence on the training fold distribution, suggesting significant variability between experimental runs. To encapsulate this, σ's are included in the tables below (e.g. Table 5.1).

Reminder: During training on the five folds, we have three sets: the training set ($\approx$ 80% of 22 Experiments), the validation set ($\approx$ 20% of 22 Experiments) and the test set (one experiment). Furthermore, once models for each experiment (which yields 23 results) have been trained, 23*4 sets of label-prediction pairs are generated: $\{\hat{y}_i, y\}$.

1. R^2_{train}, ($MAPE_{train}$): When training one model, there are four training folds. We can compare $\{\hat{y}_i, y\}_{train}$, to calculate the corresponding R^2 ($MAPE_{train}$) score. Averaging this over all five folds, and over five seeds results in R^2_{train} ($MAPE_{train}$)
2. R^2_{CV},($MAPE_{CV}$): When training one model, there is one validation fold. We can compare $\{\hat{y}_i, y\}_{validation}$, to calculate the corresponding R^2 ($MAPE_{CV}$) score. Averaging this over all five folds, and over five seeds results in R^2_{CV} ($MAPE_{CV}$).
3. R^2_{total},($MAPE_{total}$): When training one model, there is one test set. From

this test set we can obtain $\{\hat{y}_i, y\}_{test}$. Gathering these points for all experiments yields 23*4 points. From this we can calculate the corresponding R^2 ($MAPE_{total}$) score. This results in R^2_{total} ($MAPE_{total}$). R^2 is calculated this way instead of an average over each individual R^2 score, because the variance was sometimes zero for a given experiment (that is, the concentration was constant).

4. $R^2_{dry,wet,mix}$, ($MAPE_{dry,wet,mix}$): From R^2_{total} ($MAPE_{dry,wet,mix}$) we have 23*4 points. 9*4 of these points correspond with wet ashes, 8*4 of which are dry, and 5*4 are mixed. The remaining four points corresponds with the magnetic ash, and it does not have its own category. From these points, we can calculate respective R^2 ($MAPE$) scores. These subsets correspond with $R^2_{dry,wet,mix}$ ($MAPE_{dry,wet,mix}$). If performance is significantly better for some of these subsets, then this suggests some ashes are easier to predict than others.
5. R^2 $\sigma_{CV,train}$, R^2 $\sigma_{CV,train}$: During training, there is one validation fold and four training folds with respective label-prediction pairs $\{\hat{y}_i, y\}_{CV,train}$. The R^2 ($MAPE$) is calculated. Over five seeds, there are five such values. The standard deviation of these values correspond with $\sigma_{CV,train}$.
6. $MAPE^0$: Same prediction-label pairs as $MAPE_{total}$ without $y_i = 0$ pairs.

For each model, a scatter plot is generated. In this plot, red points indicate that the model has the best performance according to the Relative = $MAPE^0$ score (with one exception for Aluminium, as will be discussed here, see the neural network table: 5.1), while blue points indicate that the model is outperformed by one of the other four models.

5.1 Comparison

Neural Network	R^2 Values						Mean Absolute Percentage Error						R^2 σ		MAPE σ		
	R^2_{train}	R^2_{CV}	R^2_{total}	R^2_{Dry}	R^2_{Mix}	R^2_{Wet}	$MAPE_{train}$	$MAPE_{CV}$	$MAPE_{total}$	$MAPE_{Dry}$	$MAPE_{Mix}$	$MAPE_{Wet}$	σ_{train}	σ_{CV}	σ_{train}	σ_{CV}	$MAPE^0$
Aluminium	0.69	0.32	0.28	-0.14	-0.02	0.41	0.50	6.50	3.99	3.07	2.51	1.94	0.09	0.09	0.19	1.00	**3.74**
Antimony	0.78	0.65	0.18	-0.21	-2.51	0.34	0.56	0.87	inf	inf	inf	0.94	0.02	0.06	0.03	0.07	1.02
Arsenic	0.78	0.35	-0.47	-1.24	-5.85	-0.32	0.10	0.17	0.30	0.10	0.13	0.60	0.02	0.35	0.00	0.02	0.30
Cadmium	0.91	0.61	0.42	0.39	-0.83	-0.00	0.19	0.33	inf	inf	inf	0.89	0.01	0.06	0.01	0.06	0.53
Calcium	0.71	0.58	0.37	0.26	0.40	0.39	0.27	0.62	1.00	1.49	0.94	0.65	0.03	0.06	0.02	0.05	**0.98**
Chloride	0.59	0.39	0.21	0.27	0.27	0.11	0.25	0.96	1.18	0.89	1.60	1.29	0.05	0.06	0.02	0.08	1.18
Chromium	0.86	0.53	0.32	0.25	0.10	0.13	0.17	0.30	inf	inf	inf	0.43	0.01	0.06	0.01	0.03	0.34
Cobalt	0.93	0.22	0.60	-1.24	0.27	0.59	0.20	0.52	inf	inf	inf	0.42	0.02	0.05	0.02	0.52	0.42
Copper	0.90	0.48	0.12	-1.17	-0.15	0.29	0.52	0.83	inf	inf	inf	1.73	0.02	0.26	0.03	0.05	1.61
Lead (Pb+2)	0.85	0.63	0.06	-0.38	-1.14	0.13	0.39	0.69	inf	inf	inf	0.95	0.03	0.05	0.04	0.06	1.03
Manganese	0.93	0.58	0.42	0.24	0.20	-0.49	0.06	0.14	inf	inf	inf	0.26	0.01	0.06	0.00	0.01	0.21
Nickel	0.91	0.56	0.49	0.22	0.06	-0.25	0.13	0.25	inf	inf	inf	0.21	0.01	0.06	0.01	0.03	0.28
Sulfate	0.32	0.44	0.07	-0.10	-0.17	-3.41	0.30	1.60	4.38	7.21	0.86	2.72	0.05	0.06	0.05	0.27	4.38
Total Organic C (TOC)	0.72	0.46	0.36	0.37	0.53	0.28	0.12	0.40	0.50	0.49	0.46	0.51	0.08	0.06	0.03	0.02	0.50
Zinc	0.89	-0.04	0.41	-0.67	-220.28	0.38	1.13	1.57	inf	inf	inf	2.58	0.06	0.54	0.17	0.24	2.90

The neural network demonstrates superior performance compared to the other models for two specific elements: Aluminium and Calcium. Although GPR has a relative error of 1.00 for Aluminium, the neural network is preferred because the GPR model did not converge and predicted values close to zero. It's important to note that a model consistently predicting zero will have a relative error of 1.00 by default. The neural network had an error of 374% for Aluminium and 98% for Calcium, both of which are quite high compared to the errors the neural network produced for other elements.

The higher MAPE value for Calcium is primarily due to two outlier experiments: Mix 5 and Dry 4. For example, the relative error would have been about 36% lower if Dry 4 had been perfectly predicted (see Figure 5.4).

Aluminium presents several outliers in terms of MAPE scores, with at least one outlier in each ash category, making it a particularly challenging element to model. This difficulty arises because each ash category contains at least one experiment with values close to zero (see Figure 5.2).

In the scatter plot, we see that there is a reasonable correlation between lower MAPE scores, and higher R^2 values, because (with the exception of Sulfate and Arsenic) they cluster in the bottom right corner, see Figure 5.1.

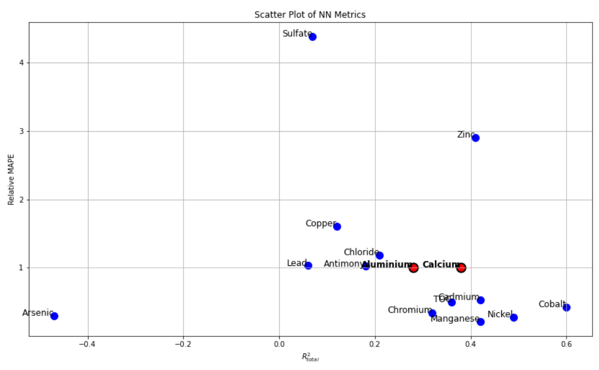


Figure 5.1: Scatter plot of R^2_{total} vs. Relative MAPE for all elements. Aluminium and Calcium is best predicted using the neural network.

Lasso	R^2 Values						Mean Absolute Percentage Error						R^2 σ		MAPE σ		
	R^2_{train}	R^2_{CV}	R^2_{total}	R^2_{Dry}	R^2_{Mix}	R^2_{Wet}	$MAPE_{train}$	$MAPE_{CV}$	$MAPE_{total}$	$MAPE_{Dry}$	$MAPE_{Mix}$	$MAPE_{Wet}$	σ_{train}	σ_{CV}	σ_{train}	σ_{CV}	$MAPE^0$
Aluminium	0.17	-0.26	-0.09	-0.05	-4.14	-0.35	9.72	13.35	11.94	11.58	28.75	4.18	0.17	0.22	2.63	6.98	12.00
Antimony	0.20	-0.10	0.07	-0.08	-4.93	0.06	inf	inf	inf	inf	inf	1.00	0.07	0.06	nan	nan	1.75
Arsenic	0.01	-0.15	-0.07	-0.15	-0.12	-0.10	0.28	0.31	0.29	0.09	0.06	0.62	0.01	0.13	0.04	0.19	0.30
Cadmium	0.00	-0.08	-0.09	-0.10	-4.68	-0.36	inf	inf	inf	inf	inf	inf	0.00	0.04	nan	nan	0.52
Calcium	0.71	0.29	0.32	0.19	0.47	0.30	0.86	1.25	1.12	1.77	0.85	0.71	0.08	0.19	0.15	0.50	1.13
Chloride	0.46	0.16	0.25	0.33	0.34	0.17	1.56	1.76	1.77	1.48	2.26	1.57	0.16	0.43	0.08	0.19	1.72
Chromium	0.00	-0.12	-0.07	-0.08	-0.95	-0.29	inf	inf	inf	inf	inf	0.52	0.00	0.04	nan	nan	0.38
Cobalt	0.82	-2.07	0.53	-0.13	-0.66	0.48	inf	inf	inf	inf	inf	0.28	0.11	4.13	nan	nan	**0.18**
Copper	0.24	-0.61	-0.14	-1.39	0.17	-0.03	inf	inf	inf	inf	inf	1.05	0.30	0.58	nan	nan	1.36
Lead (Pb+2)	0.06	-0.26	-0.05	-0.03	-2.86	-0.21	inf	inf	inf	inf	inf	0.97	0.03	0.17	nan	nan	**0.93**
Manganese	0.00	-0.09	-0.08	-0.08	-3.34	-1.24	inf	inf	inf	inf	inf	0.26	0.00	0.05	nan	nan	0.20
Nickel	0.00	-0.10	-0.08	-0.10	-6.03	-0.97	inf	inf	inf	inf	inf	0.22	0.00	0.06	nan	nan	0.32
Sulfate	0.94	-2.02	-0.09	-1.37	-0.13	-158.69	1.35	4.46	13.71	15.06	0.92	21.06	0.02	3.46	0.31	2.93	16.17
Total Organic C (TOC)	0.60	0.25	0.38	0.21	0.47	0.41	0.45	0.54	0.55	0.64	0.39	0.47	0.06	0.11	0.04	0.09	0.56
Zinc	0.58	-7.10	0.15	-0.22	-135.25	0.06	inf	inf	inf	inf	inf	3.38	0.10	7.55	nan	nan	2.71

Lasso has the best performance for two elements, Lead and Cobalt. It has a Relative error of 18% for Cobalt, and 93% for Lead.

The clustering observed in the scatter plot for the neural network is less apparent for Lasso. Additionally, the R^2 values for both Lead is close to zero. An R^2 value of zero indicates that the model is merely calculating the average of the observed values (or rather, it is no better than a model doing so). The fact that many elements appear in the bottom left of the plot, showing good MAPE scores but poor R^2 values, suggests that the model is primarily effective at predicting small values, and that the large values are suppressed. For Lead, there are approximately three non-zero outliers, as shown in Figure 5.4.

Judging by Figure 5.2, the model is mainly finding averages, as both Lead and Cobalt have some "flat" lines. This suggests that it is finding averages (especially when considering the R^2 so close to zero), but due to the weighting of the MAPE score, we still get good percentages.

We note that Lasso has a smaller correlation between R^2 values and relative errors compared to the neural network when considering the scatter plot, see Figure 5.2.

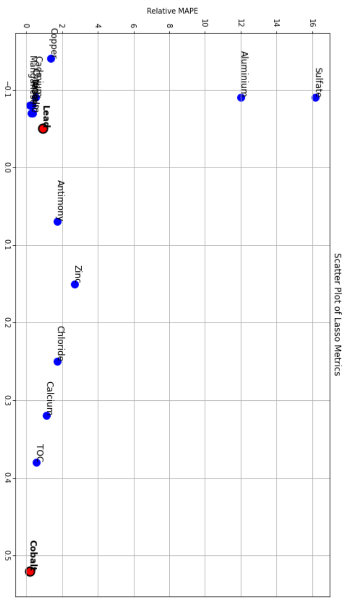


Figure 5.2: Scatter plot of R^2_{total} vs. Relative MAPE for all elements. Cobalt and Lead is best predicted using Lasso.

Ridge Results	R^2 Values						Mean Absolute Percentage Error						R^2 σ		MAPE σ		
	R^2_{train}	R^2_{CV}	R^2_{total}	R^2_{Dry}	R^2_{Mix}	R^2_{Wet}	$MAPE_{train}$	$MAPE_{CV}$	$MAPE_{total}$	$MAPE_{Dry}$	$MAPE_{Mix}$	$MAPE_{Wet}$	σ_{train}	σ_{CV}	σ_{train}	σ_{CV}	$MAPE^0$
Aluminium	0.64	-0.26	-1.18	-0.15	-10.03	-3.42	4.98	9.89	16.76	7.39	57.60	-3.42	0.06	0.31	1.34	6.42	18.82
Antimony	0.80	0.43	0.21	0.03	0.30	0.16	inf	inf	inf	inf	inf	inf	0.03	0.07	nan	nan	**0.92**
Arsenic	0.85	0.31	-1.82	-2.73	-5.71	-1.72	0.06	0.17	0.37	0.18	0.18	0.70	0.07	0.02	0.02	0.10	0.37
Cadmium	0.95	0.88	-1.85	0.54	-0.03	-9.28	inf	inf	inf	inf	inf	inf	0.08	0.04	nan	nan	0.63
Calcium	0.77	0.35	**0.49**	0.31	0.67	0.45	0.71	1.18	1.04	1.79	0.62	0.67	0.08	0.19	0.13	0.42	1.04
Chloride	0.56	-0.04	-2.41	0.40	0.28	-4.43	1.53	2.11	2.77	1.44	2.09	4.26	0.04	0.33	0.09	0.40	2.74
Chromium	0.86	0.63	-4.00	-0.39	0.50	-13.48	inf	inf	inf	inf	inf	inf	0.04	0.03	nan	nan	0.58
Cobalt	-2.39	-1.71	-8.96	-1.22	-1.85	-2.15	inf	inf	inf	inf	inf	inf	inf	inf	nan	nan	1.00
Copper	0.87	0.30	-0.67	-1.60	0.12	-0.67	inf	inf	inf	inf	inf	inf	0.05	0.27	nan	nan	2.35
Lead (Pb+2)	0.89	0.41	-7.13	-0.38	-11.95	-13.60	inf	inf	inf	inf	inf	inf	0.04	0.17	nan	nan	2.00
Manganese	0.96	0.89	-7.41	0.60	0.77	-70.09	inf	inf	inf	inf	inf	1.09	0.01	0.02	nan	nan	0.48
Nickel	0.95	0.85	-0.15	0.50	-0.63	-5.53	inf	inf	inf	inf	0.50	0.38	0.02	0.04	nan	nan	0.41
Sulfate	0.95	-1.01	**0.19**	-1.95	0.19	-112.7	1.21	4.14	12.12	11.48	0.73	19.80	0.01	2.35	0.23	2.67	13.60
Total Organic C (TOC)	0.66	0.62	-4.13	0.47	-0.37	-6.81	0.42	0.64	0.93	0.52	0.62	1.46	0.06	0.16	0.03	0.12	0.93
Zinc	0.97	-38.95	0.58	-0.75	-19.28	0.56	inf	inf	inf	inf	inf	inf	0.01	42.79	nan	nan	**1.65**

Ridge has the best performance of the five models for two elements, Zinc and Antimony with relative errors of 165% and 92% respectively. Both of these show up in the bottom right of the scatter plot, meaning that Ridge only outperforms the other models when it performs its best in both R^2 and MAPE simultaneously.

Antimony has many constant values (see Figure 5.2), which contribute to higher errors, while Zinc has many values close to zero, making it a challenging element to predict. Neither element has clear outliers, as shown in Figure 5.4. Instead, Ridge struggles across several experiments.

Its R^2 behaviour for Zinc is worth noting, as its total R^2 is much better than that over the categories. We can see in 5.2 why this is the case. Some experiments have much larger observed values than the majority, which is all in the very bottom left, driving the variance up (improving the R^2 value), see Figure 5.3.

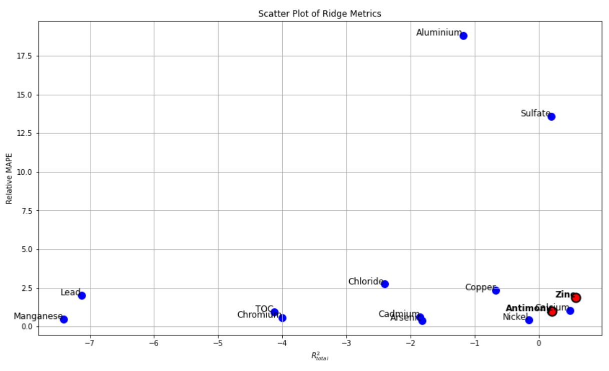


Figure 5.3: Scatter plot of R^2_{total} vs. Relative MAPE for all elements. Zinc and Antimony is best predicted using Ridge.

Random Forest	R^2 Values						Mean Absolute Percentage Error						R^2 σ		MAPE σ		
	R^2_{train}	R^2_{CV}	R^2_{total}	R^2_{Dry}	R^2_{Mix}	R^2_{Wet}	$MAPE_{train}$	$MAPE_{CV}$	$MAPE_{total}$	$MAPE_{Dry}$	$MAPE_{Mix}$	$MAPE_{Wet}$	σ_{train}	σ_{CV}	σ_{train}	σ_{CV}	$MAPE^0$
Aluminium	0.94	0.29	0.27	0.14	-0.98	0.21	9951.01	248.38	8.77	11.71	16.36	2.83	0.01	0.13	131.71	143.35	8.77
Antimony	0.93	0.46	0.47	0.33	-0.57	0.44	154.10	29.72	inf	inf	inf	1.01	0.01	0.12	8.60	8.97	1.09
Arsenic	0.94	0.39	-0.42	-1.30	-3.35	-0.29	21.95	5.52	0.26	0.12	0.11	0.49	0.02	0.15	3.64	3.26	**0.27**
Cadmium	0.98	0.91	0.34	0.37	-1.57	0.13	43.98	13.27	inf	inf	inf	0.75	0.00	0.01	7.80	7.05	**0.35**
Calcium	0.93	0.31	0.14	0.07	0.08	0.15	154.60	38.78	2.02	2.78	2.26	1.37	0.02	0.21	24.39	19.26	2.02
Chloride	0.96	0.58	0.25	0.30	0.26	0.16	232.27	56.91	0.98	0.98	0.99	0.85	0.01	0.15	10.60	11.94	**0.99**
Chromium	0.96	0.75	0.27	0.30	-0.23	0.05	33.94	10.67	inf	inf	inf	0.42	0.01	0.01	2.83	3.72	**0.29**
Cobalt	0.95	-0.26	0.33	-6.05	0.20	0.44	52.07	10.89	inf	inf	inf	1.29	0.03	1.44	10.33	4.75	1.01
Copper	0.94	0.28	0.05	-0.28	0.09	0.05	136.54	30.64	inf	inf	inf	1.69	0.01	0.38	16.76	8.78	1.54
Lead (Pb+2)	0.94	0.46	0.23	0.23	-0.76	0.04	100.23	25.66	inf	inf	inf	1.28	0.01	0.12	11.75	10.59	1.14
Manganese	0.99	0.93	0.62	0.59	0.07	-0.13	11.24	3.88	inf	inf	inf	0.15	0.00	0.01	1.48	1.12	**0.12**
Nickel	0.98	0.85	0.42	0.32	-2.01	-0.20	27.59	6.88	inf	inf	inf	0.18	0.00	0.03	2.25	2.18	**0.24**
Sulfate	0.98	0.71	0.01	-1.68	-0.21	-25.62	633.01	151.94	9.36	14.27	0.58	9.93	0.01	0.11	94.56	59.69	9.37
Total Organic C (TOC)	0.95	0.64	0.16	0.22	0.20	0.09	70.30	14.14	0.35	0.29	0.35	0.32	0.01	0.05	5.06	1.84	**0.35**
Zinc	0.95	-1.69	0.49	-5.70	-49.24	0.62	292.85	76.87	–	–	–	–	0.03	2.00	63.93	39.86	4.66

Random Forest performs well on seven different elements, making it the best model by amount of best fits by far (when using the relative error as the metric). It has the best performance on Arsenic (27%), Cadmium (35%), Chloride (99%), Chromium (29%), Manganese (12%), Nickel (24%) and TOC (35%).

This is consistent with what one might expect, as similar efforts to predict leachate concentrations also got good results when using random forest [Ers+23].

The elements in the figure below cluster in the bottom right corner, except for Arsenic (Figure 5.4). From Figure 5.2, one can assume that this is due to one outlier experiment, as is confirmed in Figure 5.4 .

Random Forest often struggles with larger values (see e.g. Cadmium, Chloride, Nickel and TOC; Figure 5.2), a challenge common to all the elements. However, these larger values typically appear as isolated outliers, suggesting that the model had little training data corresponding to these particular experiments.

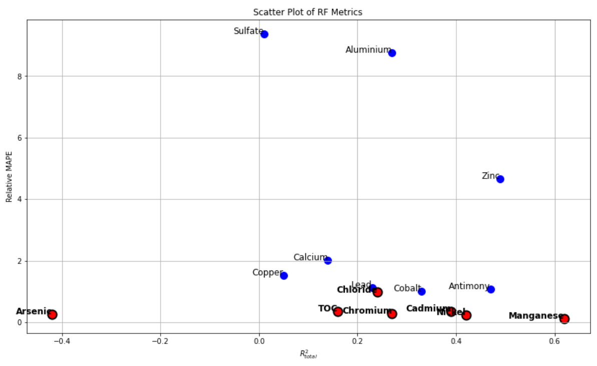


Figure 5.4: Scatter plot of R^2_{total} vs. Relative MAPE for various elements.

Gaussian Process Regressor	R^2 Values						Mean Absolute Percentage Error						R^2 σ		MAPE σ		
	R^2_{train}	R^2_{CV}	R^2_{total}	R^2_{Dry}	R^2_{Mix}	R^2_{Wet}	$MAPE_{train}$	$MAPE_{CV}$	$MAPE_{total}$	$MAPE_{Dry}$	$MAPE_{Mix}$	$MAPE_{Wet}$	σ_{train}	σ_{CV}	σ_{train}	σ_{CV}	$MAPE^0$
Aluminium	0.99	-0.42	-0.74	-0.89	-0.61	-0.40	93.55	1.00	1.00	1.00	1.00	1.00	3.46	0.13	130.66	73.04	1.00
Antimony	0.76	0.11	-0.23	-0.28	-5.08	-0.32	144.31	29.41	inf	inf	inf	1.07	0.02	0.11	7.97	8.87	1.45
Arsenic	0.27	-0.01	-0.50	-0.16	-0.11	-0.62	19.95	5.35	0.30	0.10	0.06	0.65	0.21	0.17	3.65	3.15	0.31
Cadmium	0.08	-0.01	-0.09	-0.09	-4.53	-0.40	35.00	9.93	inf	inf	inf	1.03	0.01	0.04	5.29	5.06	0.51
Calcium	0.99	-0.88	-1.32	-1.85	-0.75	-1.63	157.35	20.91	1.00	1.00	1.00	1.00	0.00	0.07	23.72	6.52	1.00
Chloride	0.99	-0.59	-0.72	-0.85	-0.87	-0.77	232.92	199.04	1.00	1.00	1.00	1.00	7.29	0.06	10.65	2.97	1.00
Chromium	0.06	-0.06	-0.10	-0.11	-1.04	-0.32	24.72	7.81	inf	inf	inf	0.50	0.01	0.04	2.71	2.52	0.38
Cobalt	0.94	-0.69	0.36	-0.04	-0.53	0.27	50.42	6.72	inf	inf	inf	0.31	0.02	1.12	8.00	3.46	0.21
Copper	0.89	0.41	0.14	-2.12	0.28	0.42	141.58	37.33	inf	inf	inf	0.91	0.05	0.22	18.83	14.91	**1.03**
Lead (Pb+2)	0.93	0.32	-0.12	-0.30	-3.91	-0.11	102.74	27.18	inf	inf	inf	0.84	0.01	0.19	11.50	9.97	1.04
Manganese	0.08	-0.02	-0.10	-0.08	-3.44	-1.30	12.54	3.58	inf	inf	inf	0.25	0.01	0.04	0.64	0.99	0.19
Nickel	0.08	-0.03	-0.10	-0.09	-6.04	-1.07	22.42	5.39	inf	inf	inf	0.22	0.01	0.05	1.61	1.55	0.32
Sulfate	0.99	0.42	-0.14	-0.62	-0.52	-0.75	643.54	106.50	0.91	0.97	0.99	0.84	0.00	0.22	77.72	46.71	**0.91**
Total Organic C (TOC)	0.99	-1.71	-1.55	-2.31	-2.97	1.18	72.65	16.79	1.00	1.00	1.00	1.00	0.00	0.18	5.60	1.02	1.00
Zinc	0.98	-4.02	-0.12	-0.18	-13.25	-0.27	294.42	58.30	inf	inf	inf	1.13	0.01	2.75	61.57	28.59	1.31

GPR has the best performance for Copper and Sulfate with 103% and 91% errors respectively. GPR often struggles with convergence on this data, and it often overfits judging by its R^2_{train} data. Considering its use of the RBF kernel, too small length scales will lead to points being "too far" away, leading to a quick exponential decay 3.3, setting the output value to zero. Judging by 5.4, this might be whats going on for Sulfate, and it is likely happening for the elements along the MAPE = 1.00 line in the scatter plot below (see e.g. TOC, Calcium, Lead, Aluminium: Figure 5.5). This suggests that this is a common problem for GPR, making it a poor choice for regression tasks of this nature.

It is evident that GPR was a weak contender due to its inherent problems with large input dimensions (see Chapter 3), so its subpar performance should not be surprising.

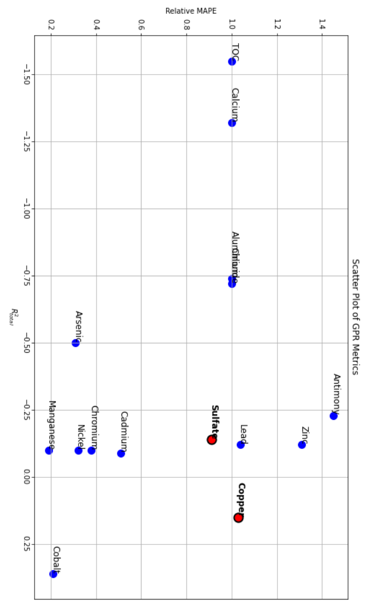
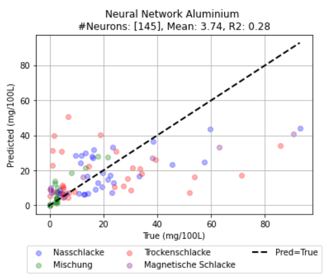


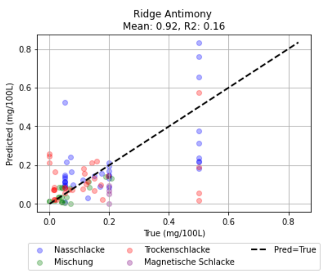
Figure 5.5: Scatter plot of R^2_{total} vs. Relative MAPE for various elements.

Below the predictions by the best performing model for the element in question is plotted against the observed values.

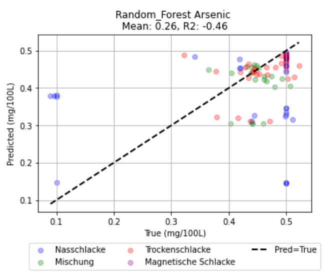
5.2 Best Models Result



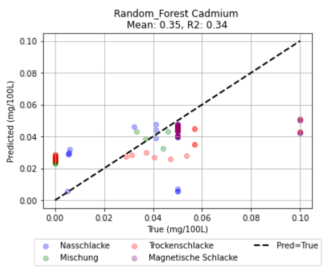
(a) Aluminium



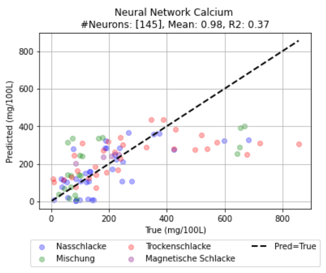
(b) Antimony



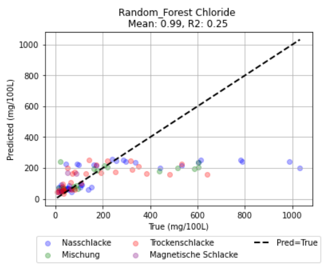
(c) Arsenic



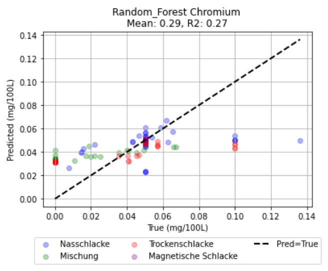
(d) Cadmium



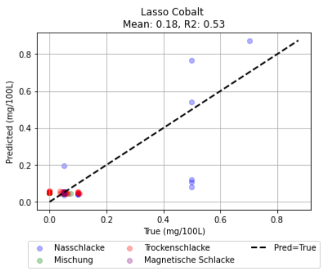
(e) Calcium



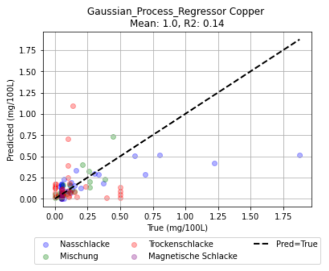
(f) Chloride



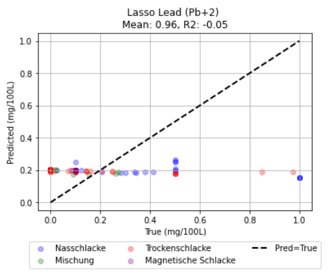
(g) Chromium



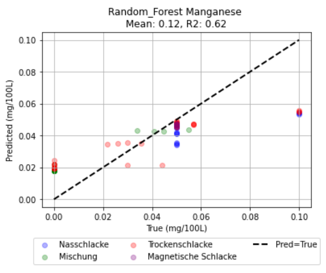
(h) Cobalt



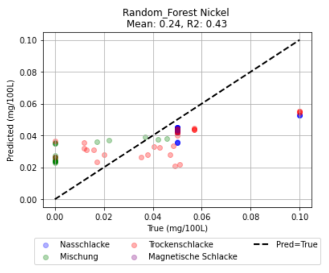
(i) Copper



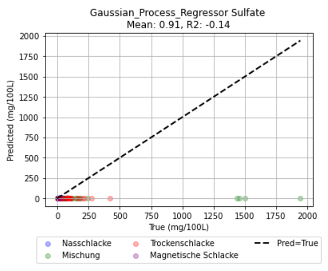
(j) Lead



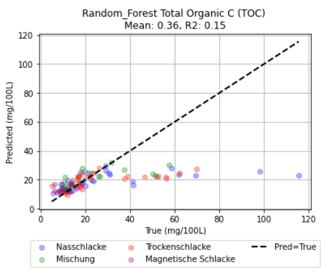
(k) Manganese



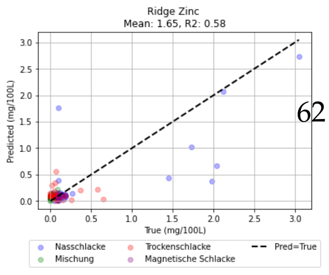
(l) Nickel



(m) Sulfate



(n) TOC



(o) Zinc

Figure 5.6: Predicted vs Observed values of the elements. Mean and R2 refers to $MAPE^0$ and $R^2_{...}$ found in the tables at the beginning of this section e.g. Table 5.1

In Figure 5.6, one should watch for horizontal lines, which indicate models that simply calculate the average. If the mean is close to 1.00 and the R^2 value is close to 0, this observation is further supported. This suggests that Lead, Cobalt, and Sulfate are ”unreliable.” Given that these elements are either modeled by a linear model (Lasso) or a model prone to predicting zero for small length scales (GPR), this is likely the underlying issue.

Sulfate, Cobalt, and Lead warrant further investigation to identify potentially better models. Alternatively, one might consider using the second-best model for these elements. For instance, the neural network could be a suitable alternative for Cobalt. Although its relative error is 42%, it has an R^2 value of 0.6, which would have been reflected in the plots seen in Figure 5.7, had this model been chosen.

They may seem more convincing, in the sense that the model has learned some of the underlying features. Models which are only finding the average typically has small standard deviations compared to the deviations of the observed values, as shown in Figure 5.9.

Antimony, Arsenic, Cadmium, Chloride, Chromium, Cobalt, Copper, Lead, Manganese, Nickel, Sulfate, TOC, and Zinc all exhibit outliers (Figure 5.2)—observed as relatively isolated points. This is also evident in Figure A.1, where disconnected and relatively low bars indicate these isolated points. This suggests that these elements, in particular, would benefit from additional data.

In contrast, Aluminium and Calcium appear to be the only elements without clear outliers. Notably, these are also the elements where the neural network performs the best. This observation could imply that with more training data, the neural network potentially outperforms other models for all elements, if enough training data is provided.

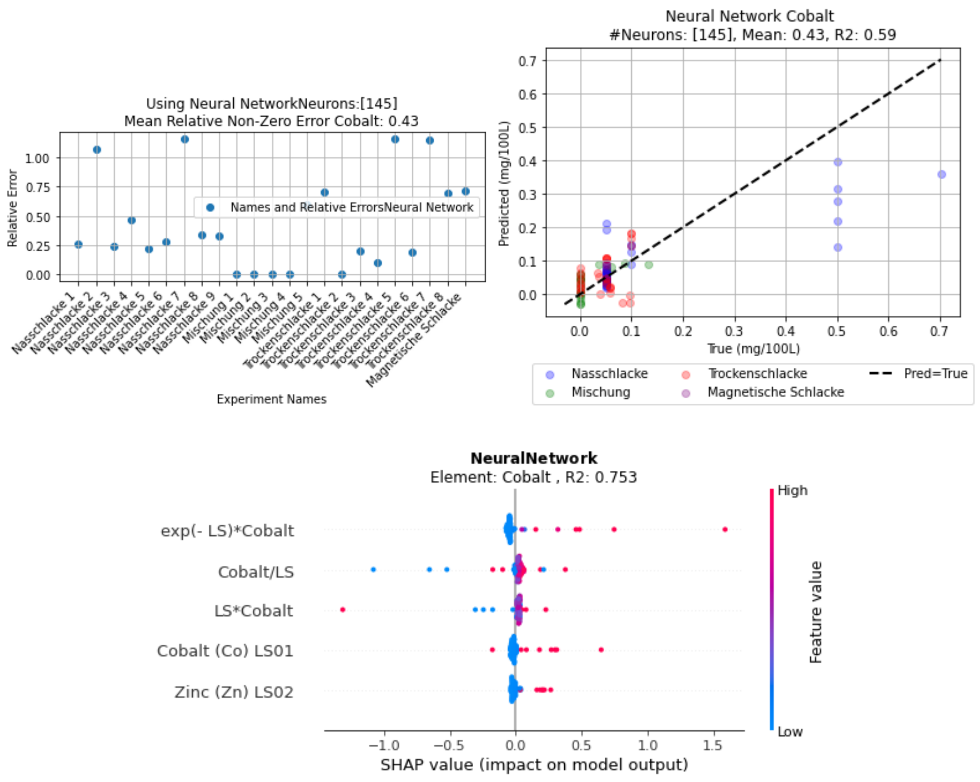


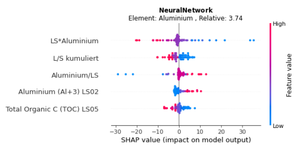
Figure 5.7: Cobalt Example

5.3 SHAP Results

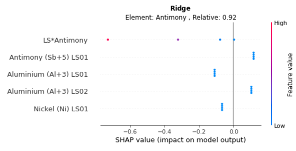
Reminder: When interpreting a SHAP plot for regression:

- **Vertical axis (top to bottom):** Represents the importance of each feature.
- **Horizontal axis:** Colors indicate feature values—red for higher values, blue for lower ones. Points to the left of the y-axis suggest that the feature decreases the output value, while points to the right indicate an increase in the output.

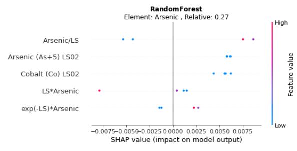
For our discussion, particularly given the small size of the training sets, it is advisable to focus on only the first three features. This is due to the inherent unreliabilities associated with small datasets.



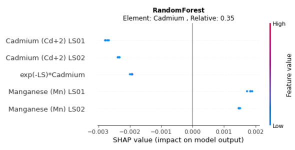
(a) Aluminium



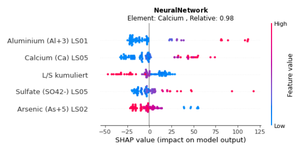
(b) Antimony



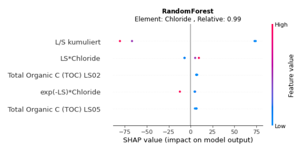
(c) Arsenic



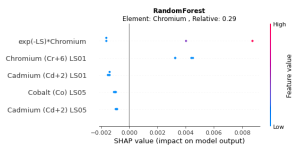
(d) Cadmium



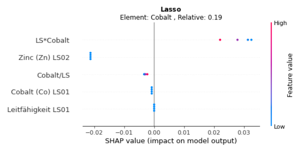
(e) Calcium



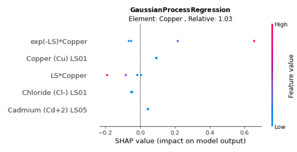
(f) Chloride



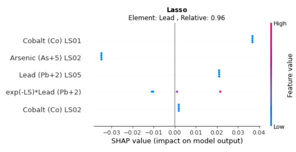
(g) Chromium



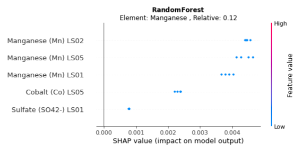
(h) Cobalt



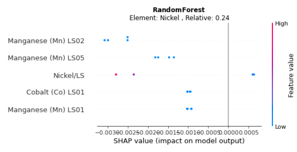
(i) Copper



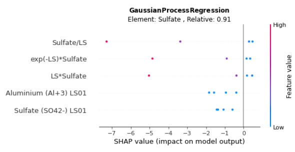
(j) Lead



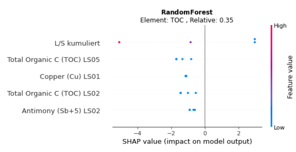
(k) Manganese



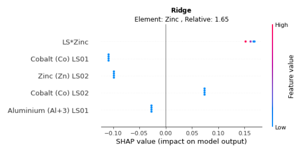
(l) Nickel



(m) Sulfate



(n) TOC



(o) Zinc

Figure 5.8: Overview of SHAP. The Relative score (seen above in each plot) corresponds with $MAPE^0$.

In the following figures, one should note that SHAP works slightly differently for neural networks, as opposed to kernel explainers. This yields a different amount of points in the plots.

Discussions here primarily use the violin plots (Figure 2.1) and compare them with the SHAP models (Figure 5.8). Chemical perspectives or viewpoints based on differential equations are largely excluded from consideration. This is because the processes involved are likely influenced by many complex, difficult-to-model factors, such as the boundary conditions of the leachate, non-homogeneous diffusion paths, and chemical interactions between elements. In other words, the governing processes are very complicated. It is hoped that most of these complexities can be captured statistically, similar to how macroscopic gas behaviors are modeled using thermodynamics rather than solving a Lagrangian in $\Re^{6N}$ dimensions. For further details regarding the statistics, concentrations plotted against the LS value by experiment are shown in Figure A.2.

In Figure 5.8, we can see that most models are behaving as expected, where the element itself, functions of it or LS, rank high on the list. Lead and Nickel are two exceptions to this. This can often suggest the model is simply calculating averages, yielding plots like Figure 5.9. Specifically for Lead, the horizontal behavior and an R^2 value close to 0 indicate a model that is not effectively capturing the variability. For Nickel, Manganese ranks high, likely due to the leaching of Nickel being relatively constant after LS = 1.0, as illustrated in Figure 2.1.

When looking at these figures it can sometimes be helpful to keep the violin plots in the beginning in mind, see Figure 2.1. If one looks at e.g. the mean, it can tell us something about the behaviour of the element. In the following, the plots seen in 5.8 are briefly discussed. Usually, one can find a reasonable connection between the LS value and the decay/increase of the concentration in question when compared with the violin plots.

In the following, each Element and its SHAP figures' relation to the violin figures at the beginning (Figure 2.1) are briefly discussed:

1. **Aluminium** seem to decrease as LS increases based off of Figure 2.1. Therefore, one would expect an increase of e.g. LS*Aluminium to correspond with smaller values. This is exactly the behaviour reflected in the SHAP plot (Figure 5.8). The inverse relationship of Aluminium/LS is also consistent with the violin plots. Aluminium has about four experiments which lead to large relative errors, as can be seen in figure 5.10.

2. **Antimony** does not exhibit a clear decrease in concentration. However, it is reassuring to note that Ridge still prioritizes functions of antimony concentrations. Additionally, Ridge does not show any clear outliers with respect to the MAPE score, as illustrated in Figure 5.4.

3. **Arsenic** appears to be relatively constant, as shown in Figure 2.1, although it seems to increase slightly. This is confirmed by the SHAP figure (Figure 5.8), which shows that larger LS values correspond to higher output values. There is one notable outlier with respect to the MAPE errors: the wet ash 1 experiment exhibits a large MAPE error, as illustrated in Figure 5.4).

4. **Cadmium** exhibits a similar issue to Arsenic with the Wet 1 experiment. According to Figure 2.1, Cadmium shows a trend where the mean concentration increases slightly but is largely constant. This observation is supported by its SHAP figure, which indicates minimal change. The LS value has a relatively minor impact compared to constant values (only as exp(-LS), in third place), suggesting that Cadmium leaching is almost time-independent.

5. **Calcium** is expected to decrease as LS increases, based on the violin figures 2.1. This behavior is observed; interestingly, the concentration of Aluminium ranks higher in the prediction of Calcium. Since Aluminium and Calcium have similar patterns, this suggests that high values of Aluminium and Calcium often occur in the same type of ash. However, it is also possible that this correlation is influenced by statistical outliers and might not hold with a larger training dataset.

6. **Chloride** should decrease with increasing LS based off of figure 2.1, and this trend is confirmed by its SHAP figure 5.8. The concentration of Chloride itself is not a significant factor for the model, which is reasonable considering that the large deviations for Chloride diminish once LS reaches 1.00.

7. **Chromium** should decrease with increasing LS, implying that e.g. exp(-LS)*Chromium should have red points on the right. We note that large starting concentrations lead to higher resulting LS values, which aligns with physical expectations.

8. **Cobalt** behaves similarly to Aluminium, Zinc, and Copper (Figure2.1), showing a rapid decrease with increasing LS and then flattening from around LS = 1.0. In Figure 5.8, the pattern of its first input (LS*Cobalt) mirrors that of Zinc, except with Zinc and Cobalt swapped. Larger LS values decrease the output relative to smaller LS values.

9. **Copper** also shows a clear decrease with increasing LS (Figure 2.1), indicating that higher LS values correspond to smaller concentrations. When exp(-LS) is big, then LS is small, which means the output should be bigger, which is exactly what can be seen.

10. **Lead** presents a more complex behavior. Although the variance decreases and flattens, the concentration of Lead remains mostly constant after LS = 1.0 (Figure 2.1). The variance reduces and centers more closely around the mean, indicating that the large outlier values at lower LS have been largely resolved. By LS = 1.0, the variance is very small compared to smaller LS values, and the mean stays relatively constant. The SHAP diagram might be surprising, as neither LS nor Lead appears among the top two features. This is not entirely unexpected given that the model did not perform exceptionally well.

However, despite the relatively constant mean and decreasing variance, the full story of Lead is not entirely clear. While one might expect the high variability in the initial stages of leaching to diminish, the changes are not straightforward. Some experiments show constant concentrations, others increase, and some decrease. This variability appears to be dependent on the specific experiment conducted (Figure A.2).

11. **Manganese** exhibits a constant behavior. Both the means and densities in Figure 2.1 (as well as Figure A.2) show minimal changes, indicating that the individual experiments themselves are also relatively constant. This is reflected in the SHAP diagram, where constant inputs for Manganese at LS = [0.1, 0.2, 0.5] are decisive. Larger starting inputs result in larger outputs, consistent with the trends suggested by the violin plots.

12. **Nickel** has a constant mean behaviour after LS=0.2 according to Figure 2.1.

    Consequently, predicting its top SHAP features is challenging. One might expect e.g. Nickel at LS = 0.5 to be a top feature, suggesting that the constant value after LS = 0.5 simply corresponds to the value at LS = 1. This behaviour is not seen in the SHAP plot.

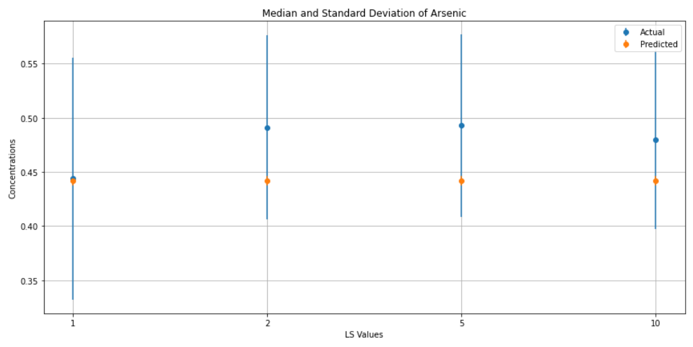
    Instead, the SHAP analysis indicates that the constant value for Nickel is heavily dependent on the value of Manganese at LS = [0.2, 0.5]. This observation is supported by Figure A.2, where their behaviors appear quite similar (even by experiment).

13. **Sulfate** exhibits a constant behavior even after LS = 0.1. This suggests that a significant portion of leachable sulfate is readily extractable and does not require large amounts of solvents (e.g., rain) for leaching (see equation 2.1 and equation 2.2). Despite this, some leaching still occurs, as indicated by Figure A.2, where the right-side plot for Sulfate shows ongoing leaching and is not overshadowed by the large value at LS = 0.1. Based on this figure, it is challenging to determine whether there is a general increase or decrease in the substance after LS = 1.0. However, a few experiments do have a slight increase at LS = 5 and 10. Indeed, Figure 5.8 suggests that we have a general increase, as high Sulfate/LS (meaning a small LS value) decreases the value, and that we have a small increase of leaching at large LS values.

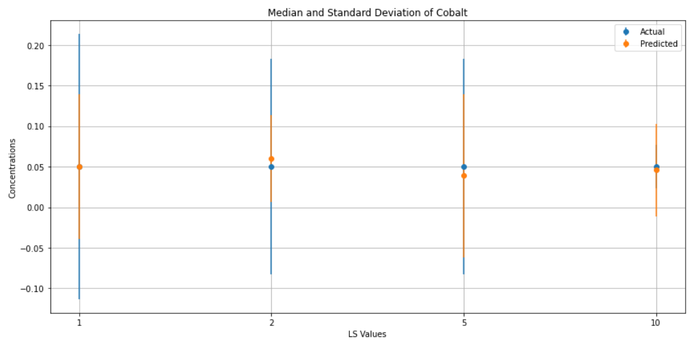
14. **TOC** shows a pattern of exponential or possibly linear decay, as illustrated in Figure 2.1. It clearly decreases with increasing LS values, which is consistent with the SHAP plot in Figure 5.8. In the SHAP plot, larger LS values are associated with smaller concentrations. Addition-

ally, small concentrations at LS = 0.5 lead to smaller concentrations at higher LS values.

15. **Zinc** also shows a negative trend with increasing LS values, similar to Copper and Cobalt. Notably, in the SHAP analysis, ”LS*Zinc” ranks as the most important feature, while the second most important feature is Cobalt, which mirrors the concentration of Zinc. This relationship is evident in Figures 2.1 and A.2.



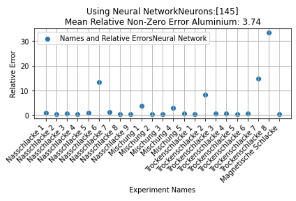
(a) Arsenic



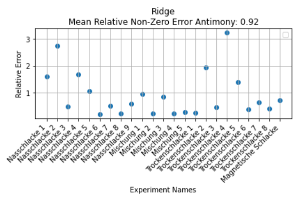
(b) Cobalt

Figure 5.9: Standard Deviation and Mean of the predictions and observations for Arsenic using the Lasso model and for Cobalt using the neural network. Observe that the standard deviation for Lasso is small compared to that of the observed values. This suggests that Lasso is only calculating the average, and is producing relatively constant values.

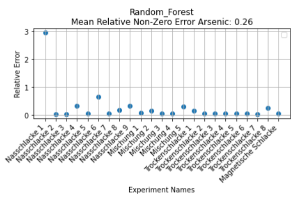
5.4 Results by Experiment



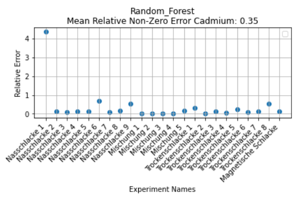
(a) Aluminium



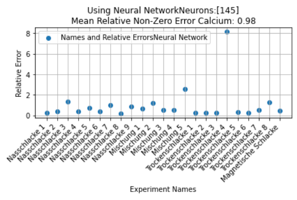
(b) Antimony



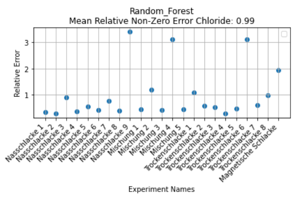
(c) Arsenic



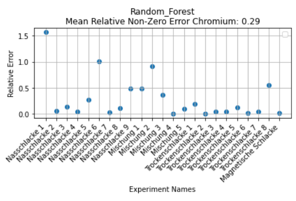
(d) Cadmium



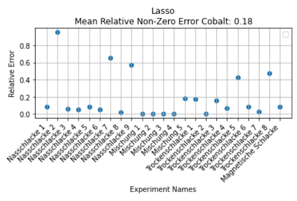
(e) Calcium



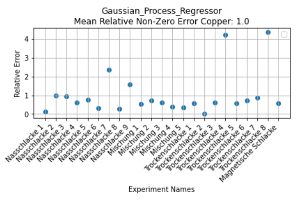
(f) Chloride



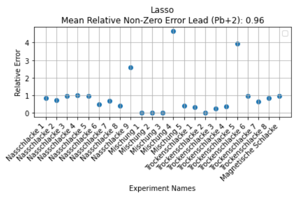
(g) Chromium



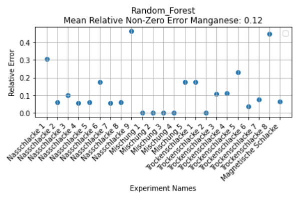
(h) Cobalt



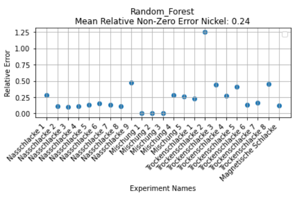
(i) Copper



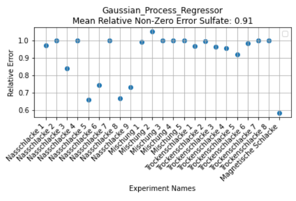
(j) Lead



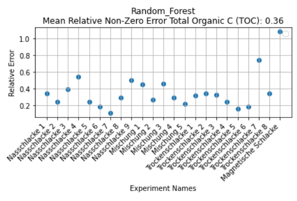
(k) Manganese



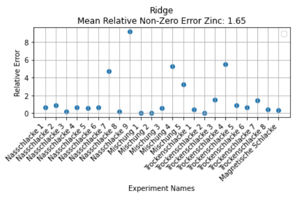
(l) Nickel



(m) Sulfate



(n) TOC



(o) Zinc

Figure 5.10: Overview of Relative Errors by Experiment. When a value would be divided by zero, its value is set to zero and excluded from the calculation of the total mean (e.g. Mix 1,2 and 3 for Nickel).

In the figures presented in Figure 5.10, the Relative Error for each left-out experiment is shown. A very large error for a particular experiment suggests either a) that the experiment is an outlier or b) that the experiment has values close to zero. A typical example of an outlier is Arsenic, where Wet Ash 1 exhibits a significantly higher relative error compared to other experiments. This observation is further supported by Figure 5.2.

Chapter 6

Conclusion

Soil and groundwater contamination from leaching of contaminants in landfills poses a significant environmental concern. This work attempted to predict long-term leaching behaviour based on short-term column leaching tests via a data-driven approach. Different machine learning models (Ridge, Lasso, GPR, NN, RF) were investigated.

The training configuration (scaling, input dimension, feature projections) was optimized on one element, calcium, using a single fold splitting seed and only one method, neural networks. As performance metric the $MAPE^0$ was chosen. The best training configuration was found to be the robust scaler, with feature enhanced input of dimension 55 with no jittering. Subsequently, training was conducted for all methods using this configuration, followed by an analysis with SHAP.

The main takeaway from all developed and investigate methods is that more training data is needed, especially due to the inhomogeneities of the current data set. If we assume an experiments' MAPE error of more than 200% of the mean as an outlier, then Cobalt and Zinc have five outliers; Aluminium and Chloride have four outliers; Antimony, Arsenic, Cadmium, Chromium, Copper, Lead, Manganese have three outliers; Calcium, Nickel, TOC two; Sulfate zero (though GPR predicted values close to zero). With 23 experiments in total, having three or more outliers per element significantly impacts the overall performance, which is detrimental to the models' reliability. In the paper by A. Ershadi et al (See [Ers+23]), figure 6 illustrates performances by training size. This figure largely reinforces the suspicion that more data is needed for better performances.

Concerning the trained models, some simplifications concerning the configuration optimization were employed. Optimizing the configuration for all elements, not only Calcium, can improve the performance significantly, and even more so if done for each method independently.

Considering that GPR and Ridge exhibited the poorest performance, frequently predicting constant values and showing generally worse R^2 and MAPE values. This suggests that the input dimensions for these models require additional care. It is particularly notable that Ridge performs significantly worse than Lasso, despite their similar structure. This discrepancy is likely due to Lasso's superior feature selection capabilities, suggesting that reducing the input space could improve performance, as not all features contribute effectively. The fact that the Neural Network (NN) and Random Forest (RF) outperform Ridge and GPR to such an extent further supports the notion that the input dimension may be excessively large for GPR and Ridge. So the simplifications done in Section 4.2 may have been too crude. This is then a good approach for future work.

The performance of the prediction for some of the elements is still fairly reasonable. Arsenic, Cadmium, Chromium, Cobalt, Manganese, Nickel and TOC all have mean errors below 50%. To understand to what extent these models should be used, one must carefully consider what regulatory limits one may use, if a 50% miss still leaves the concentration below the regulatory limit for a landfill, then quick tests may be done. Out of the seven elements with errors below 50%, six were predicted using Random Forest. Having six elements with errors below 50% was only also achieved for the neural network, which even outperformed all the other models if we were to use R^2 as our performance indicator instead of $MAPE^0$.

Despite some MAPE values being relatively poor, the SHAP analysis was fairly convincing. Features involving the element itself, functions of it, or LS consistently ranked among the top two in importance, with the exception of Lead. The frequent appearance of non-trivial functions in the SHAP analysis suggests that feature projections are highly valuable. Based on this observation, exploring rate equations or other feature projections further could enhance the models' predictive accuracy. In addition, using SHAP for feature selection instead of correlation coefficients may prove useful. Also, the raw data contains 32 different elements. Meaning that only about half (17 elements) was used in this project. Studying whether or not it can be useful to include some or all of these remaining elements could be valuable.

The models are not yet suitable for real landfill applications, but the methodology developed here, when applied to a larger dataset, shows great potential.

Bibliography

[Abf23] Zürcher Abfallbewertung". *Kurzbericht Schlackenprojekt*. Accessed: 2024-07-30. 2023. URL: https://www.zh.ch/content/dam/zhweb/bilder-dokumente/themen/umwelt-tiere/abfall-rohstoffe/abfallwirtschaft/publikationen/kehrichtverwertung-kva/schlackenprojekt_zh_kva_kurzbericht_2023.pdf.

[Age16] European Environoment Agency. *Minicipa Waste Management Fact Sheet Switzerland*. Accessed: 2024-07-30. 2016. URL: https://www.eionet.europa.eu/etcs/etc-ce/products/country-profiles-on-the-management-of-municipal-waste-1/switzerland_msw_2016.pdf.

[Age24] European Environment Agency. *Diversion of waste from landfill in Europe*. Accessed: 2024-07-30. 2024. URL: https://www.eea.europa.eu/en/analysis/indicators/diversion-of-waste-from-landfill.

[Aut24a] Various Authors. *Draw.io: Create your own flowcharts*. Accessed: 2024-08-21. 2024. URL: https://app.diagrams.net/.

[Aut24b] Various Authors. *NN-SVG: Create Neural Net Diagrams*. Accessed: 2024-08-21. 2024. URL: http://alexlenail.me/NN-SVG/AlexNet.html.

[Axl97] Sheldon Axler. *Linear Algebra Done Right*. Springer, 1997.

[BBE17] Majid Bagheri, Alireza Bazvand, and M. Ehteshami. "Application of artificial intelligence for the management of landfill leachate penetration into groundwater, and assessment of its environmental impacts". In: *Journal of Cleaner Production* 149 (Feb. 2017). DOI: 10.1016/j.jclepro.2017.02.157.

[BDE09] Senem Bayar, Ibrahim Demir, and Guleda Onkal Engin. “Modeling leaching behavior of solidified wastes using back-propagation neural networks”. In: *Ecotoxicology and Environmental Safety* 72.3 (2009), pp. 843–850. ISSN: 0147-6513. DOI: https://doi.org/10.1016/j.ecoenv.2007.10.019.

[BEG19] Ahmad Bazoobandi, Samad Emamgholizadeh, and Hadi Ghorbani. “Estimating the amount of cadmium and lead in the polluted soil using artificial intelligence models”. In: *Environmental Monitoring and Assessment* 191.11 (2019). Received 08 Jul 2019, Accepted 24 Oct 2019, Published online: 27 Nov 2019, pp. 933–951. DOI: 10.1080/19648189.2019.1686429.

[BPK16] Steven L. Brunton, Joshua L. Proctor, and J. Nathan Kutz. “Discovering governing equations from data by sparse identification of nonlinear dynamical systems”. In: *Proceedings of the National Academy of Sciences* 113.15 (2016). Department of Mechanical Engineering, University of Washington, Seattle, WA 98195; Institute for Disease Modeling, pp. 3932–3937. DOI: 10.1073/pnas.1517384113.

[C E06] C. K. I. Williams C. E. Rasmussen. *Gaussian Processes for Machine Learning, the MIT Press, 2006, ISBN 026218253X. c 2006 Massachusetts Institute of Technology*. Accessed: 2024-07-30. 2006. URL: www.GaussianProcess.org/gpml.

[Che09] Liangyue Chen. “Curse of Dimensionality”. In: *Encyclopedia of Database Systems*. Ed. by Ling Liu and M. Tamer Ozsu. Boston, MA: Springer, 2009. DOI: 10.1007/978-0-387-39940-9_133.

[Con24] PyTorch Contributors. *torch.nn.SmoothL1Loss — PyTorch 2.0 documentation*. Accessed: 2024-08-16. 2024. URL: https://pytorch.org/docs/stable/generated/torch.nn.SmoothL1Loss.html.

[Duv] David Duvenaud. *Kernel Cookbook*. Accessed: 2024-07-30. URL: https://www.cs.toronto.edu/~duvenaud/cookbook/.

[Ers+23] Amirhossein Ershadi, Michael Finkel, Bernd Susset, and Peter Grathwohl. “Applicability of machine learning models for the assessment of long-term pollutant leaching from solid waste materials”. In: *Waste Management* 171 (2023), pp. 337–349. DOI: 10.1016/j.wasman.2023.09.001.

[Eur24] Confederation of European Waste-to-Energy Plants. *Bottom Ash Fact Sheet*. Accessed: 2024-07-30. 2024. URL: https://www.cewep.eu/wp-content/uploads/2017/09/FINAL-Bottom-Ash-factsheet.pdf.

[Fed24] Swiss Fedlex. *Gewaesserschutzverordnung*. Accessed: 2024-07-30. 2024. URL: https://www.fedlex.admin.ch/eli/cc/1998/2863_2863_2863/de.

[FG17] Michael Finkel and Peter Grathwohl. "Impact of pre-equilibration and diffusion limited release kinetics on effluent concentration in column leaching tests: Insights from numerical simulations". In: *Waste Management* 63 (2017). Special Thematic Issue: Sanitary Landfilling, pp. 58–73. ISSN: 0956-053X. DOI: https://doi.org/10.1016/j.wasman.2016.11.031. URL: https://www.sciencedirect.com/science/article/pii/S0956053X16306924.

[GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[Gir15] Ross Girshick. "Fast R-CNN". In: *arXiv preprint arXiv:1504.08083* (2015). arXiv: 1504.08083 [cs.CV]. URL: https://doi.org/10.48550/arXiv.1504.08083.

[Gla21] Andreas Glauser. "Factors influencing the quality of bottom ash from municipal solid waste incineration in Switzerland". Thesis. Bern: Universität Bern, 2021. URL: https://boristheses.unibe.ch/id/eprint/2697.

[Hin+24] Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R. Salakhutdinov. *Improving neural networks by preventing co-adaptation of feature detectors*. Accessed: 2024-08-14. 2024. URL: https://arxiv.org/abs/1207.0580.

[int24] Power engineering international. *Ash handling: Why dry bottoms are better than wet bottoms*. Accessed: 2024-07-30. 2024. URL: https://www.powerengineeringint.com/world-regions/asia/ash-handling-why-dry-bottoms-are-better-than-wet-bottoms/.

[Jaf+23] Bahram Jafrasteh, Daniel Hernandez-Lobato, Simon Pedro Lubian-Lopez, and Isabel Benavente-Fernandez. "Gaussian processes for missing value imputation". In: *Knowledge-Based Systems* 273 (2023), p. 110603. ISSN: 0950-7051. DOI: https://doi.org/10.1016/j.knosys.2023.110603.

[lea23] Scikit - learn. *Gaussian Process Sklearn*. Accessed: 2024-07-30. 2023. URL: https://scikit-learn.org/stable/modules/gaussian_process.html.

[Lea24] Scikit Learn. *Preprocessing data*. Accessed: 2024-07-30. 2024. URL: https://scikit-learn.org/stable/modules/preprocessing.html.

[lea24] scikit learn. *Compare scalers*. Accessed: 2024-07-30. 2024. URL: https://scikit-learn.org/stable/auto_examples/preprocessing/plot_all_scaling.html#sphx-glr-auto-examples-preprocessing-plot-all-scaling-py.

[Lib23] LibreTexts. *Rate equations*. Accessed: 2024-07-30. 2023. URL: https://chem.libretexts.org/Bookshelves/General_Chemistry/Map%3A_General_Chemistry_(Petrucci_et_al.)/14%3A_Chemical_Kinetics/14.03%3A_Effect_of_Concentration_on_Reaction_Rates%3A_The_Rate_Law#:~:text=the%20exponent%20to,reactant.

[LL17] Scott M. Lundberg and Su-In Lee. "A Unified Approach to Interpreting Model Predictions". In: *Advances in Neural Information Processing Systems* 30 (2017). Paul G. Allen School of Computer Science and Department of Genome Sciences, University of Washington, Seattle, WA 98105, pp. 4765–4774. URL: https://arxiv.org/abs/1705.07874.

[Llo04] Stephen Lloyd. *Convex Optimization*. Stanford University, 2004.

[Lun23] Scott M. Lundberg. *SHAP Documentation*. Accessed: 2024-08-17. 2023. URL: https://shap.readthedocs.io/en/latest/.

[Pie+23] Ester Piegari, Giorgio De Donno, Davide Melegari, and Valeria Paoletti. "A machine learning-based approach for mapping leachate contamination using geoelectrical methods". In: *Waste Management* 157 (2023), pp. 121–129. ISSN: 0956-053X. DOI: https://doi.org/10.1016/j.wasman.2022.12.015.

[Res24] Stifung Zentrum Für Nachhaltige Abfall- und Resourcennutzung. *ZAR*. Accessed: 2024-07-30. 2024. URL: https://zar-ch.ch/en/home/competencesprojects/dry-discharge.

[Rou+08] Nicolas Roussat, Jacques Méhu, Mohamed Abdelghafour, and Pascal Brula. "Leaching behaviour of hazardous demolition waste". In: *Waste Management* 28.11 (2008), pp. 2032–2040. ISSN: 0956-053X. DOI: https://doi.org/10.1016/j.wasman.2007.10.019.

[Sci24a] Scikit-learn. *Gaussian Kernels Documentation scikit-learn*. Accessed: 2024-08-14. 2024. URL: https://scikit-learn.org/stable/api/sklearn.gaussian_process.html.

[Sci24b] Scikit-learn. *Mean Absolute Percentage Error Documentation scikit-learn*. Accessed: 2024-08-14. 2024. URL: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.mean_absolute_percentage_error.html.

[SO10] Alan Stuart and Keith Ord. *Kendalls Advanced THeory of Statistics, Distribution Theory*. Wiley, 2010.

[TF01] Robert Tibshirani Trevor Hastie and Jerome Friedman. *The Elements of Statistical Learning*. -, 2001.

[Tiw+15] Manoj Kumar Tiwari, Samir Bajpai, U.K. Dewangan, and Raunak Kumar Tamrakar. "Suitability of leaching test methods for fly ash and slag: A review". In: *Journal of Radiation Research and Applied Sciences* 8.4 (2015), pp. 523–537. ISSN: 1687-8507. DOI: https://doi.org/10.1016/j.jrras.2015.06.003.

[Tre09] Jerome Friedman Trevor Hastie Robert Tibshirani. *The elements of statistical learning: Data mining, inference, and prediction*. 2nd ed. 2009. Corr. 3rd printing 5th Printing. Springer Series in Statistics. Springer, 2009. ISBN: 0387848576; 9780387848570; 9780387848587; 0387848584.

[Udd+19] T. Uddh Soderberg, D. Berggren Kleja, M. Astrom, J. Jarsjo, M. Froberg, A. Svensson, and A. Augustsson. "Metal solubility and transport at a contaminated landfill site – From the source zone into the groundwater". In: *Science of The Total Environment* 668 (2019), pp. 1064–1076. ISSN: 0048-9697. DOI: https://doi.org/10.1016/j.scitotenv.2019.03.013. URL: https://www.sciencedirect.com/science/article/pii/S0048969719309921.

[Umw24] Bundesamt fuer Umwelt BAFU. *Landfills in Switzerland*. Accessed: 2024-07-30. 2024. URL: https://www.bafu.admin.ch/bafu/de/home/themen/abfall/fachinformationen/abfallentsorgung/deponien.html.

[Wan16] George C. Wang. *The Utilization of Slag in Civil Infrastructure Construction*. WP, 2016.

Appendix A

Appendix

A.0.1 Full List of Measured Elements

A list of all measured elements is provided below. They were measured at seven different LS values (LS cumulative = [0.1,0.2,0.5,1,2,5,10]). Values used for inputs are highlighted in **bold**:

1. Redox Potential: Measures the ability of a solution to oxidize or reduce substances.
2. **pH**: Indicates the acidity or alkalinity of the solution, measured on a scale from 0 to 14. A pH less than 7 indicates acidity, while a pH greater than 7 indicates alkalinity.
3. **Conductivity**: Measures the ability of the water to conduct electrical current.
4. Sodium (Na)
5. Ammonium (NH4+)
6. Potassium (K)
7. **Calcium (Ca)**
8. Magnesium (Mg)
9. Strontium (Sr)
10. Barium (Ba)
11. Iron (Fe)
12. **Manganese (Mn)**
13. **Aluminium (Al+3)**
14. **Arsenic (As+5)**

15. **Antimony (Sb+5)**
16. **Cadmium (Cd+2)**
17. **Chromium (Cr+6)**
18. **Copper (C)**
19. **Cobalt (Co)**
20. **Lead (Pb+2)**
21. **Nickel (Ni)**
22. **Zinc (Zn)**
23. Tin (Sn)
24. Titanium (Ti)
25. Silica (Si)
26. Fluoride (F-)
27. **Chloride (Cl-)**
28. Bromide (Br-)
29. Nitrate (NO3-)
30. **Sulfate (SO42-)**
31. **Total Organic Carbon (TOC)**: Measures the amount of organic carbon in water.
32. Total Inorganic Carbon (TIC): Measures inorganic carbon sources.

The elements are listed in the order they appear in the raw dataset.

A.0.2 Bayes Theorem

This appendix includes additional theorems, figures, and supporting material that are not presented in the main report to maintain readability.

The formulas are referenced from the following source: [Axl97].

Spectral Theorem:

A simplified form of the Spectral Theorem is as follows:

Let A be a positive definite matrix of size nxn in $\Re$. Then:

$$A = UDU^T$$

with $UU^T = I$.

Independent probabilities satisfy:

$$P(A \cap B) = P(A) * P(B)$$

Bayes Theorem [SO10]:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \tag{A.1}$$

A.0.3 Derivatives on Matrices:

For an invertible Matrix A (positive definite matrices are invertible, due to the *spectral theorem* [Axl97]):

$$I = AA^{-1} \rightarrow 0 = \partial(AA^{-1}) = \partial A * A^{-1} + \partial A^{-1} * A \tag{A.2}$$

$$\partial(A^{-1}) = -A(\partial A)A^{-1} \tag{A.3}$$

A.0.4 Derivative of the logarithm of the determinant on a positive definite symmetric matrix

Consider a positive definite symmetric matrix A(x) $\in n \times n$ which depends on the vector **x**=:x$\in \Re^n$. The following is known as Jacobis Formula:

$$\frac{\partial}{\partial x} ln|A| = tr(A^{-1}\frac{\partial A}{\partial x}) \tag{A.4}$$

$$\tag{A.5}$$

Proof (consider the spectral theorem), $A = UDU^T$ for real symmetric positive definite matrices A. It follows immediately:

$$A = \sum_{i=1}^{N} \lambda_i u_i u_i^T$$

$$u_i^T u_j = \delta_{ij}$$

$$A^{-1} = \sum_{i=1}^{n} \frac{1}{\lambda_i} u_i u_i^T$$

$$|A| = \prod_{i=1}^{n} \lambda_i$$

$$U = \mathbf{u}_i$$

From this, Jacobis Formula can be derived as follows:

$$\begin{aligned}
U^T U &= I \\
\partial U^T U + U^T \partial U &= 0 \\
|A| &= \prod \lambda_i \\
\partial ln(|A|) &= \sum \frac{1}{\lambda} \partial \lambda \\
&\textit{Since:} \\
\sum_i \frac{1}{\lambda_i} \partial \lambda_i &= tr(D^{-1} \partial D) = \\
tr(U^T U D^{-1} U^T U \partial D U^T) &= tr(U D^{-1} U^T U \partial D U^T) = tr(A^{-1} \partial A)
\end{aligned}$$

In the last line, we used cyclicity of traces (tr(ABC) = tr(BCA)). The shorthands $\partial B = \frac{\partial B}{\partial \mathbf{x}_i}$ (for some index i),$\prod f_i = \prod_{i=1}^n$, and $\sum_i f_i = \sum_{i=1}^n f_i = \sum f$ have been used.

A.0.5 Losses and performance:

Consider in this section the features and labels $(\mathbf{x}_i, y_i)$ as well as the models prediction $\hat{y}_i$.

The $R2$-score is often used when measuring the performance of a predictive regression model, see e.g. [Ers+23]:

$$\mathbf{R}^2 = 1 - \frac{\sum_{i=1}^N (y_i - \hat{y}_i)^2}{\sum_{i=1}^N (y_i - \bar{y})^2} \tag{A.6}$$

With $\bar{y} = \frac{1}{N} \sum_{i=1}^N y_i$

The mean-squared-error is the sum of the squared difference of the predictions and labels, divided by the amount of points [Sci24b]:

$$\mathbf{MSE} = \sum_{i=1}^N (y_i - \hat{y}_i)^2 \tag{A.7}$$

The mean-absolute-percentage error (MAPE) score and Relative error is given by:

$$\mathbf{MAPE} = \frac{1}{N} \sum_{i=1}^N |\frac{y_i - \hat{y}_i}{y_i}| \tag{A.8}$$

$$MAPE^0 \underset{y_i \neq 0}{=} \frac{1}{N} \sum_{i=1}^N |\frac{y_i - \hat{y}_i}{y_i} \tag{A.9}$$

If no labels are zero, they are equivalent. Note that MAPE (despite its name) is not in percent. I.e a MAPE score of 0.2 should be interpreted as an average error of 20%, NOT 0.2%. If some labels are zero, MAPE will be infinite (sklearn will return a large value), and $MAPE^0$ will simply skip these.

Another loss function, which is not as prone to outliers is the L1 loss, or the smooth L1 loss:

$$\mathbf{L1} = \sum_{i=1}^{N} |y_i - \hat{y}_i| \tag{A.10}$$

$$\textbf{Smooth L1} = \sum_{i=1}^{N} \mathcal{L}(x_i, y_i) = \sum_{i=1}^{N} \begin{cases} 0.5(y_i - \hat{y}_i)^2/\beta, & if \quad |y_i - \hat{y}_i| < \beta \\ |y_i - \hat{y}_i| - 0.5 \cdot \beta, & else \end{cases} \tag{A.11}$$

$\beta = 1$ is by default in the pytorch package.

A.1 Plots of the distributions

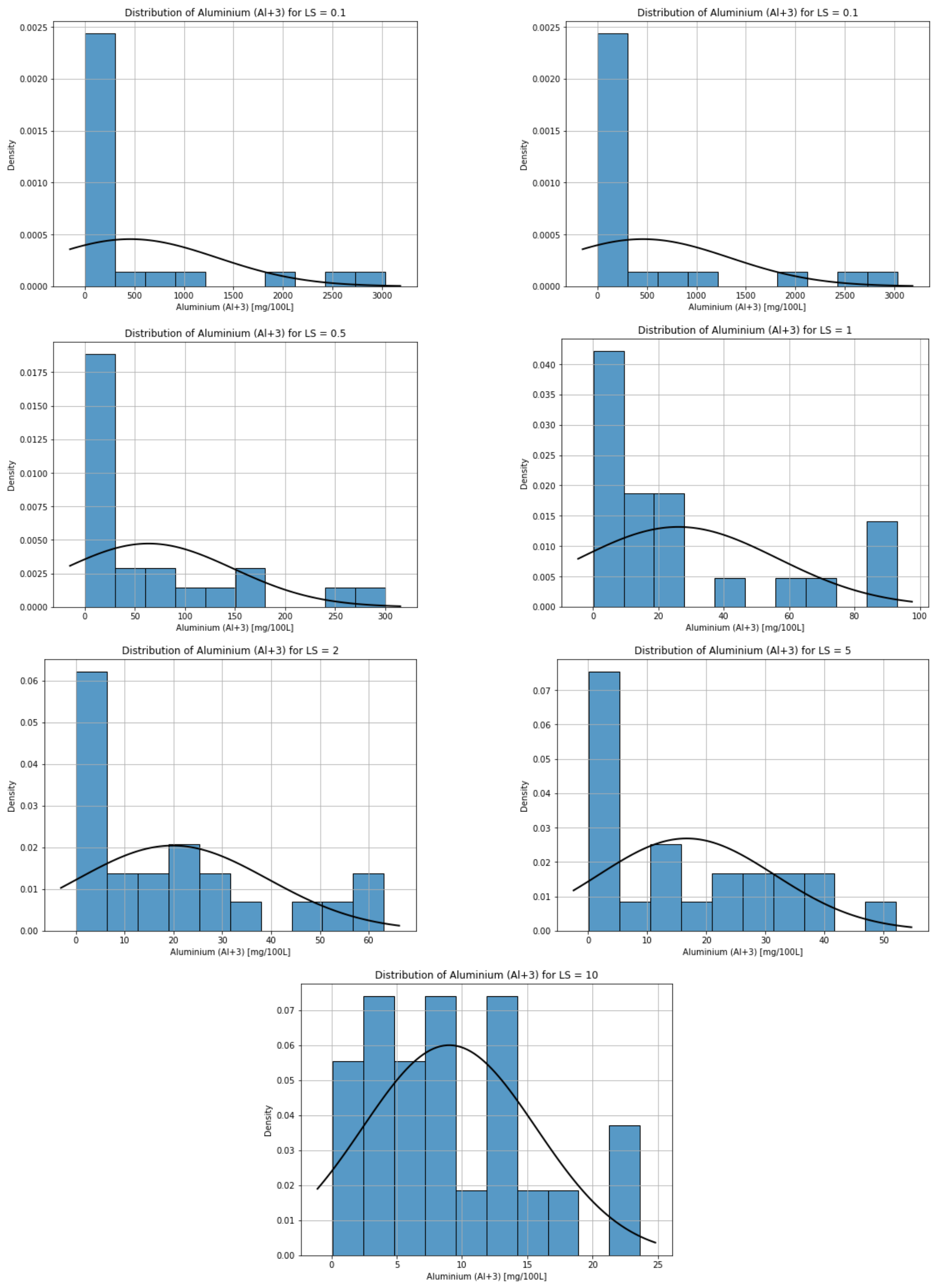


Figure A.1: Distribution of Aluminium by LS

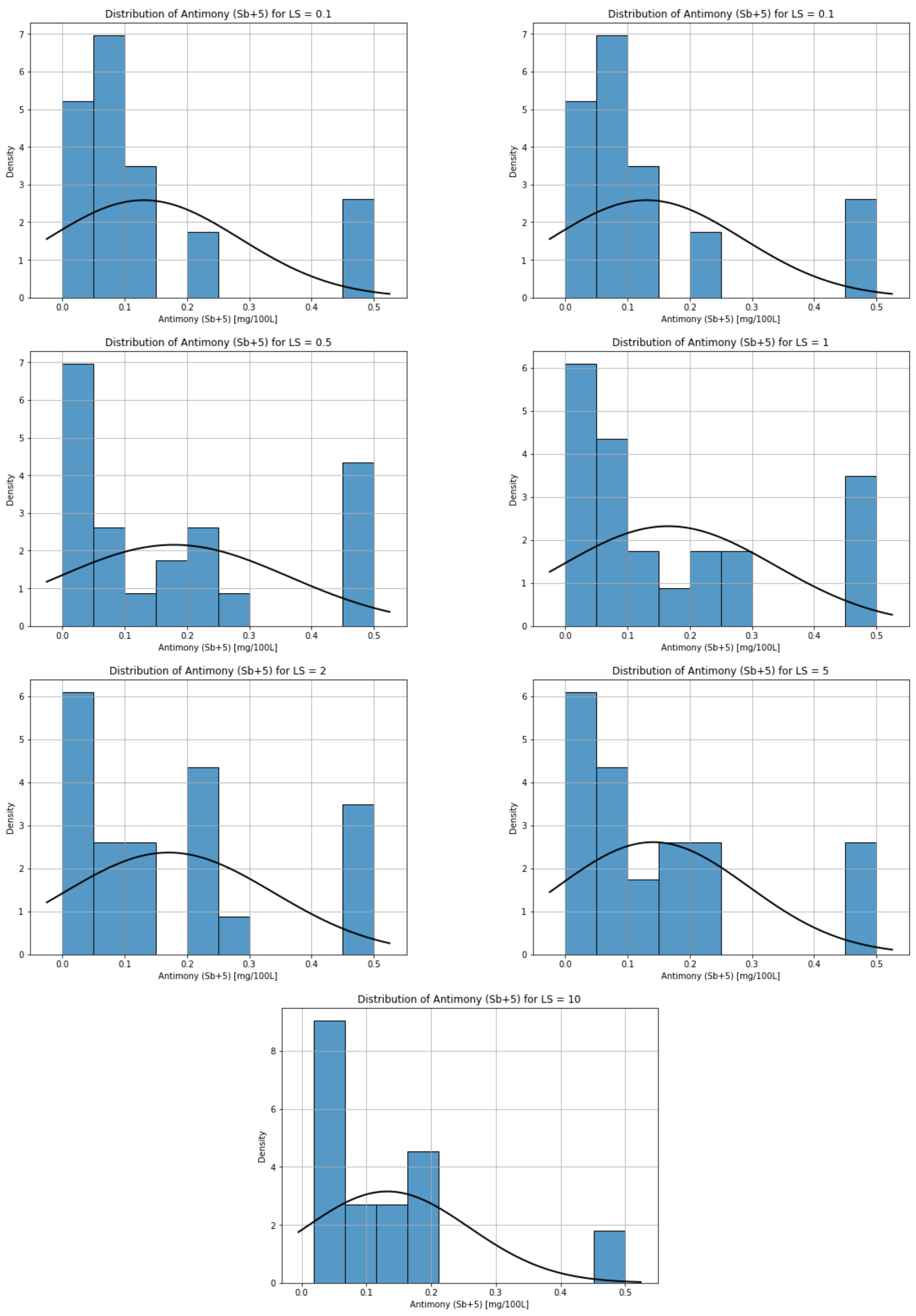


Figure A.2: Distribution of Antimony by LS

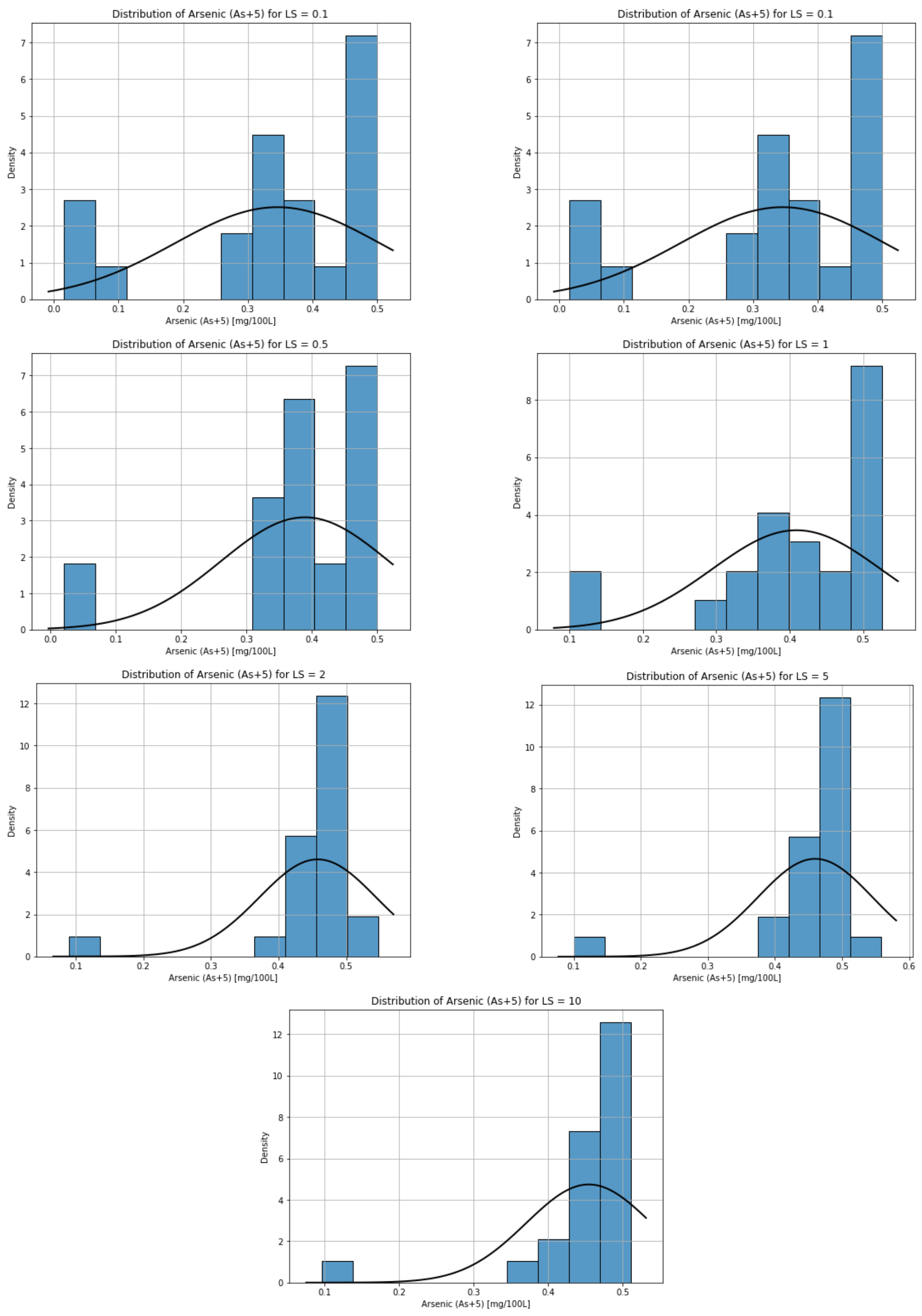


Figure A.3: Distribution of Arsenic by LS

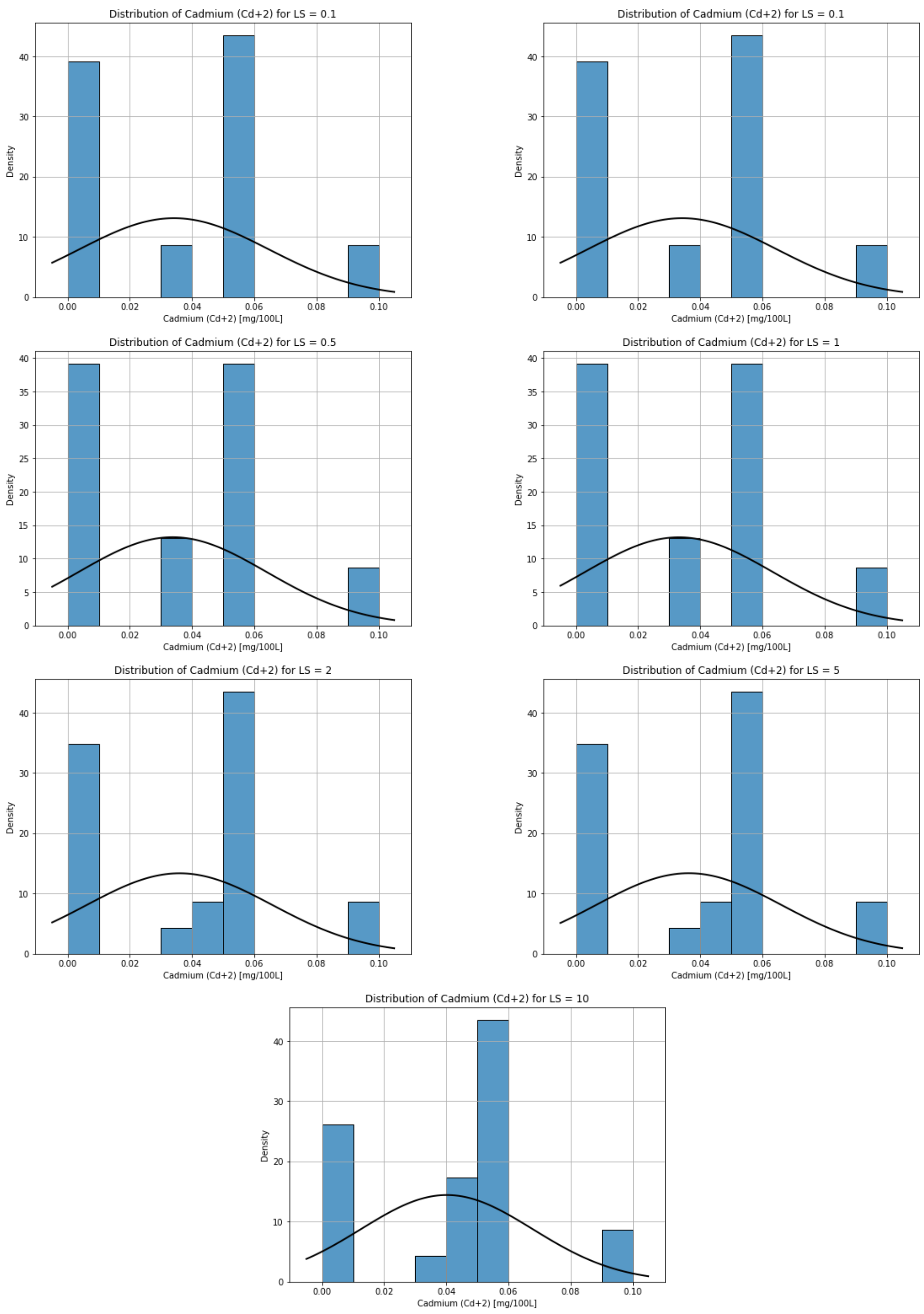


Figure A.4: Distribution of Cadmium by LS

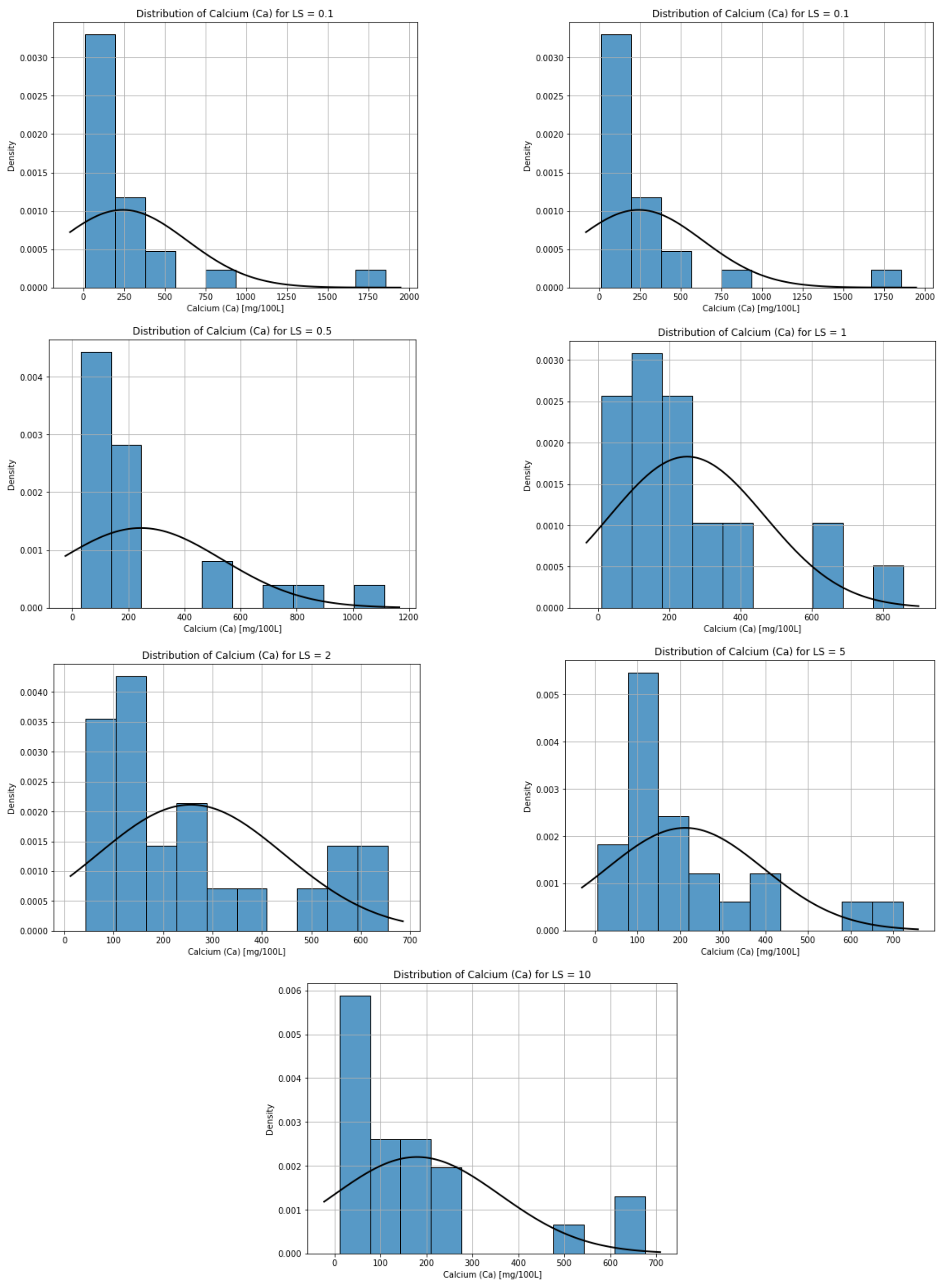


Figure A.5: Distribution of Calcium by LS

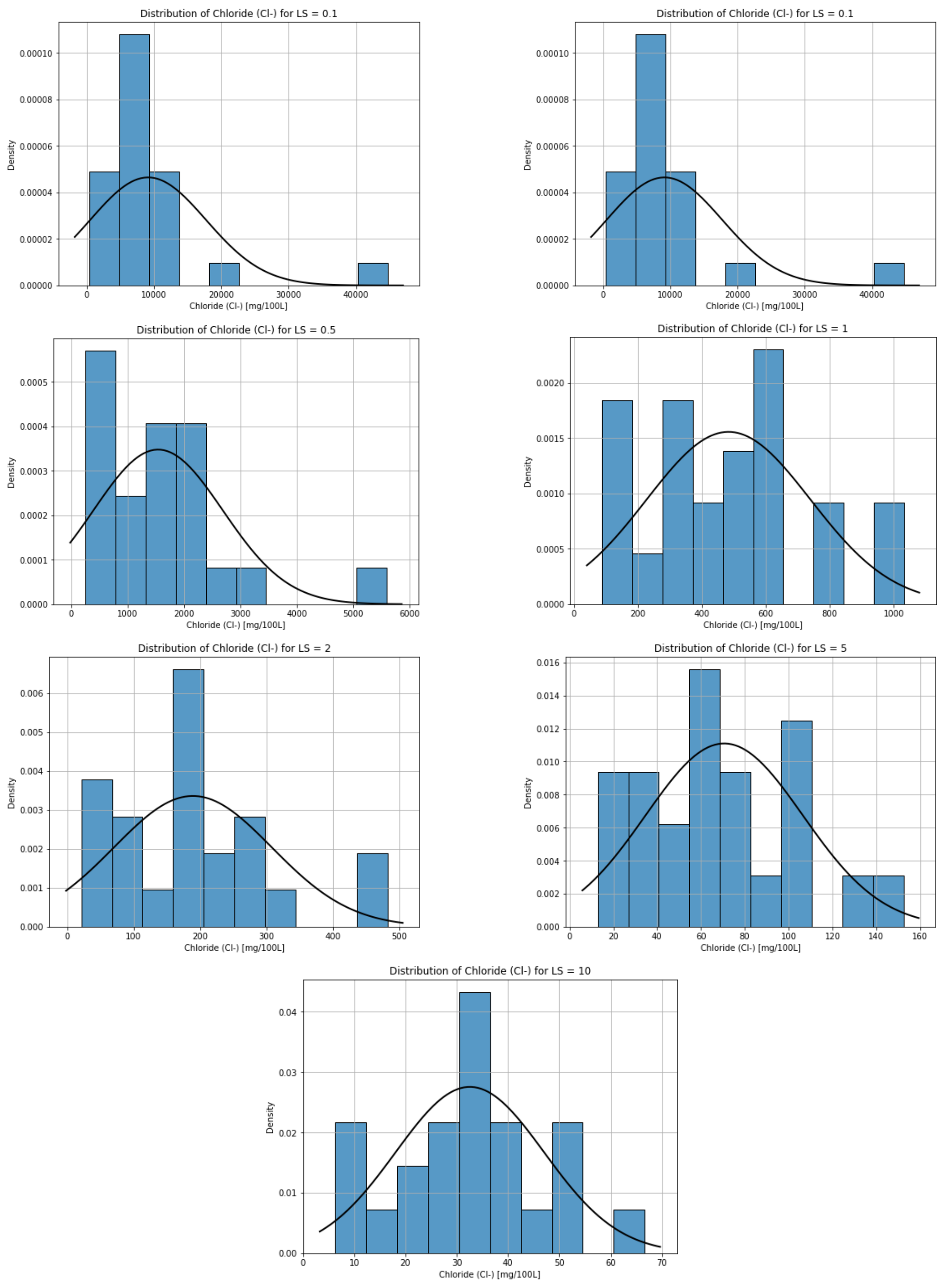


Figure A.6: Distribution of Chloride by LS

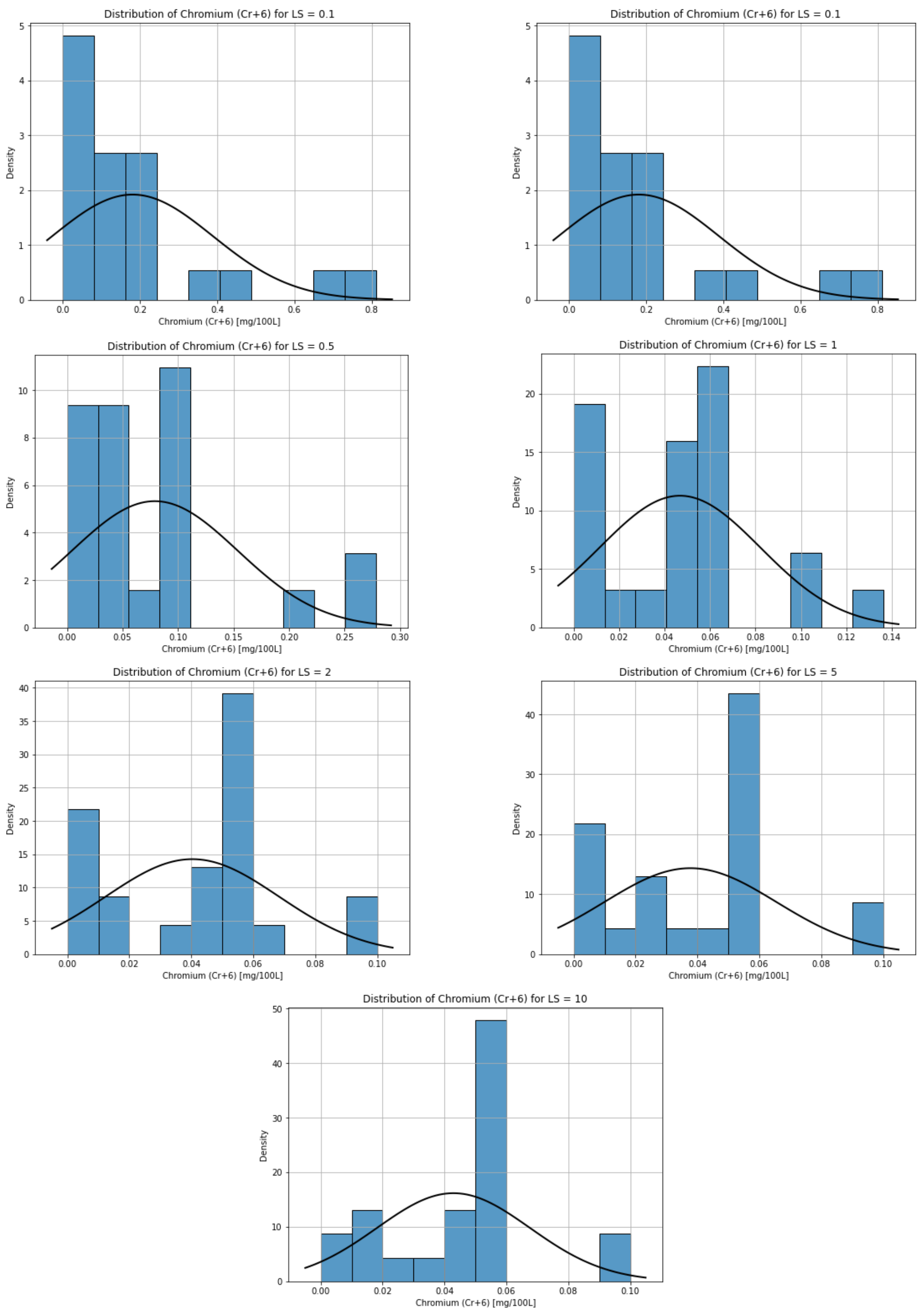


Figure A.7: Distribution of Chromium by LS

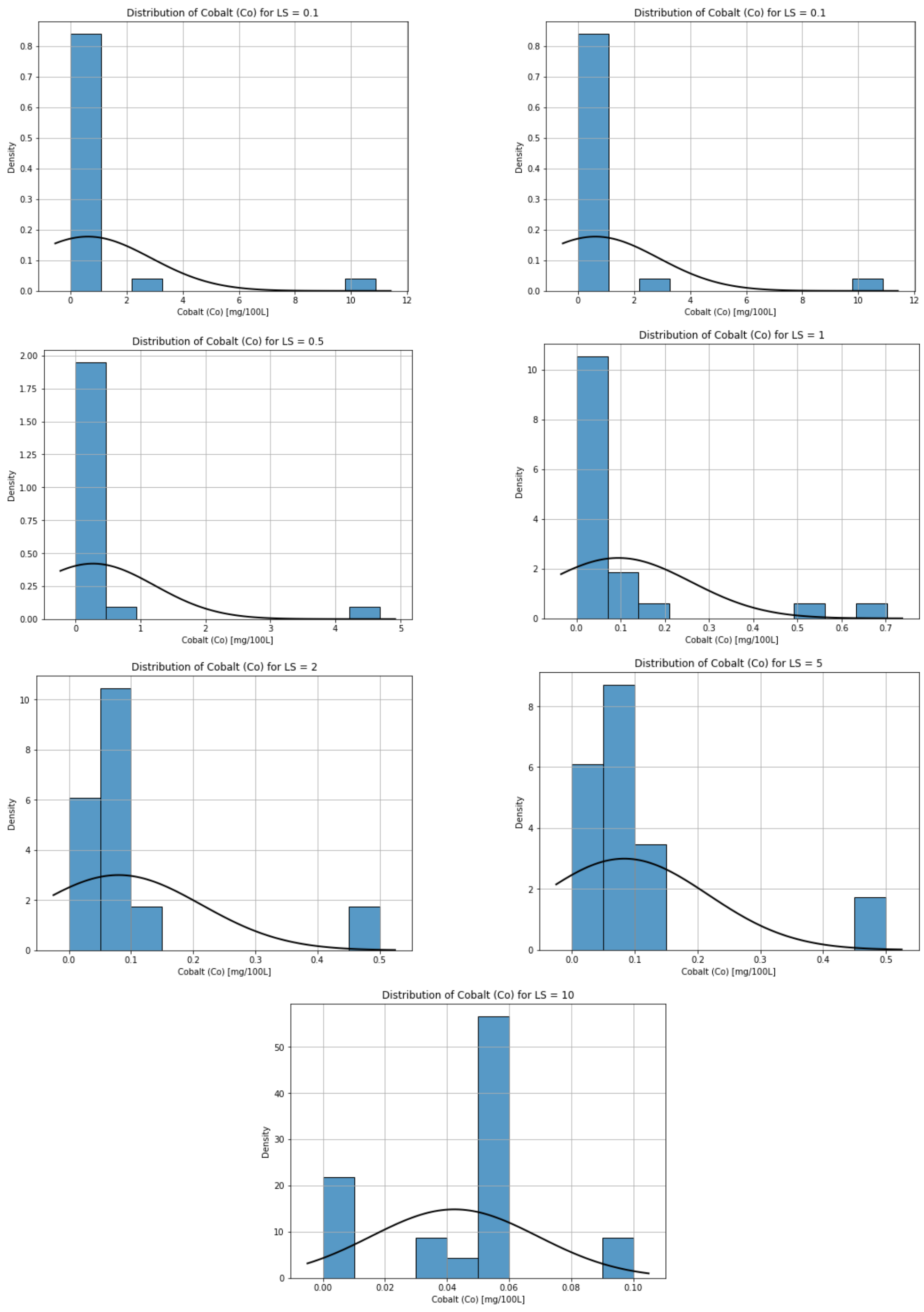


Figure A.8: Distribution of Cobalt by LS

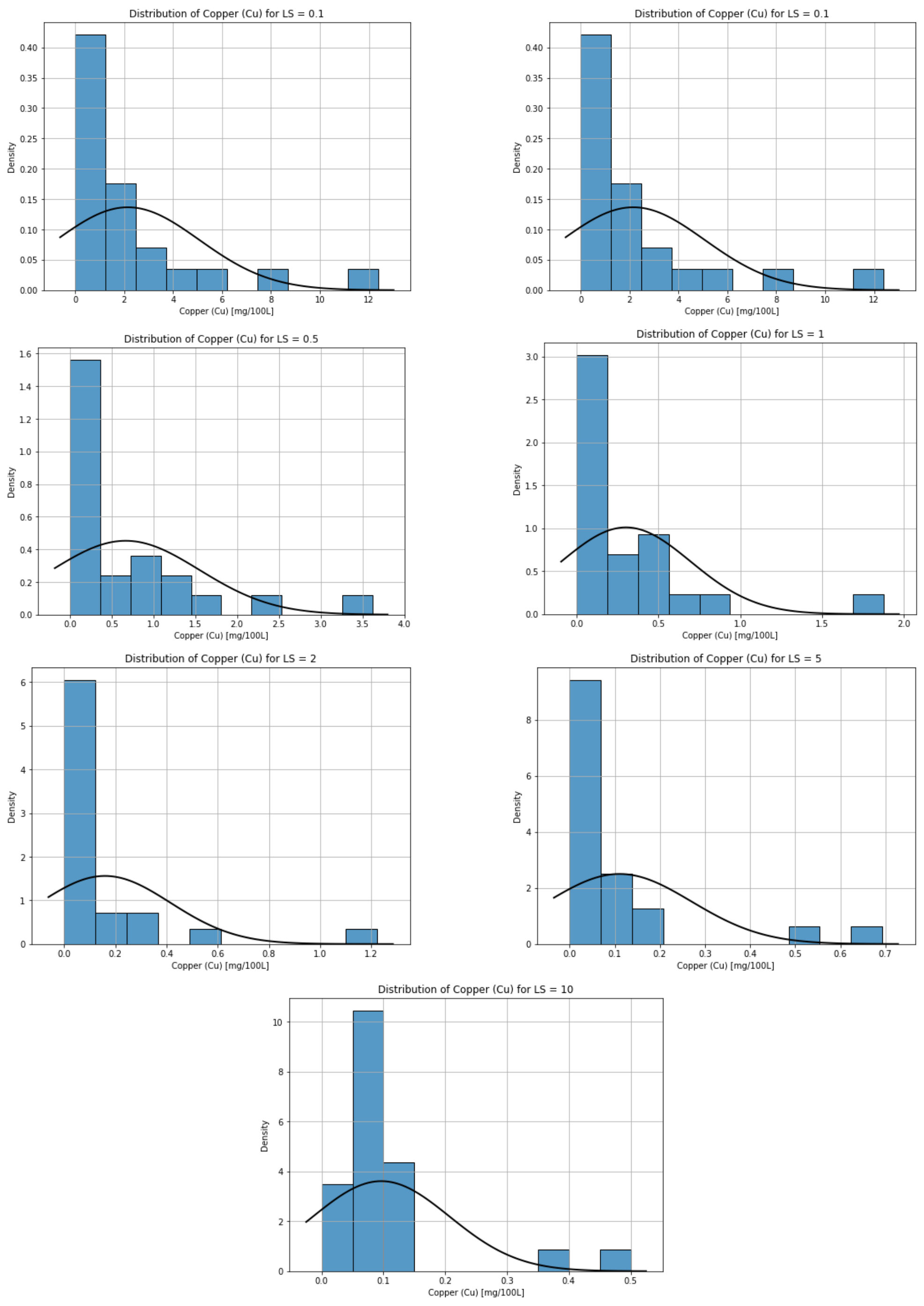


Figure A.9: Distribution of Copper by LS

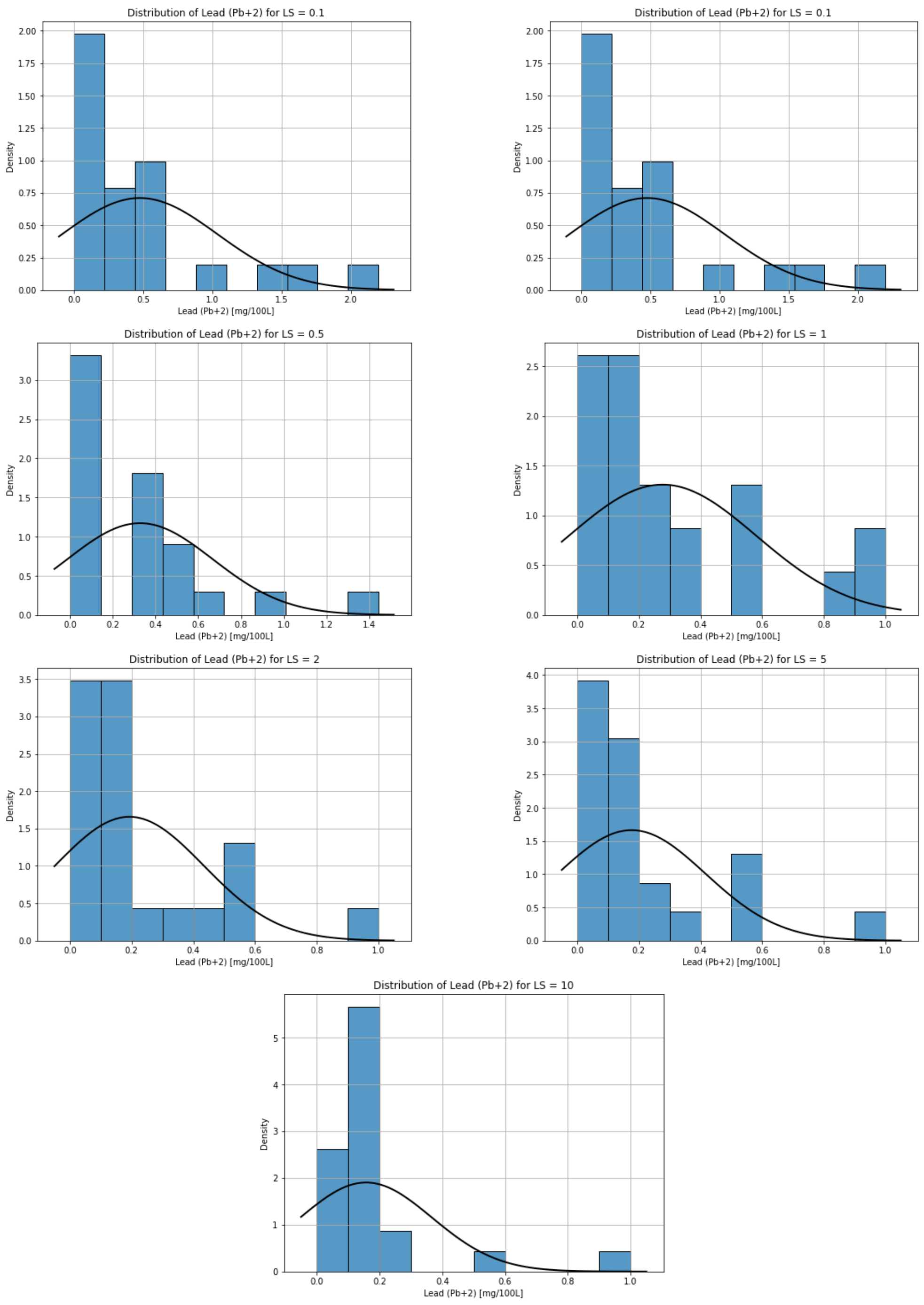


Figure A.10: Distribution of Lead by LS

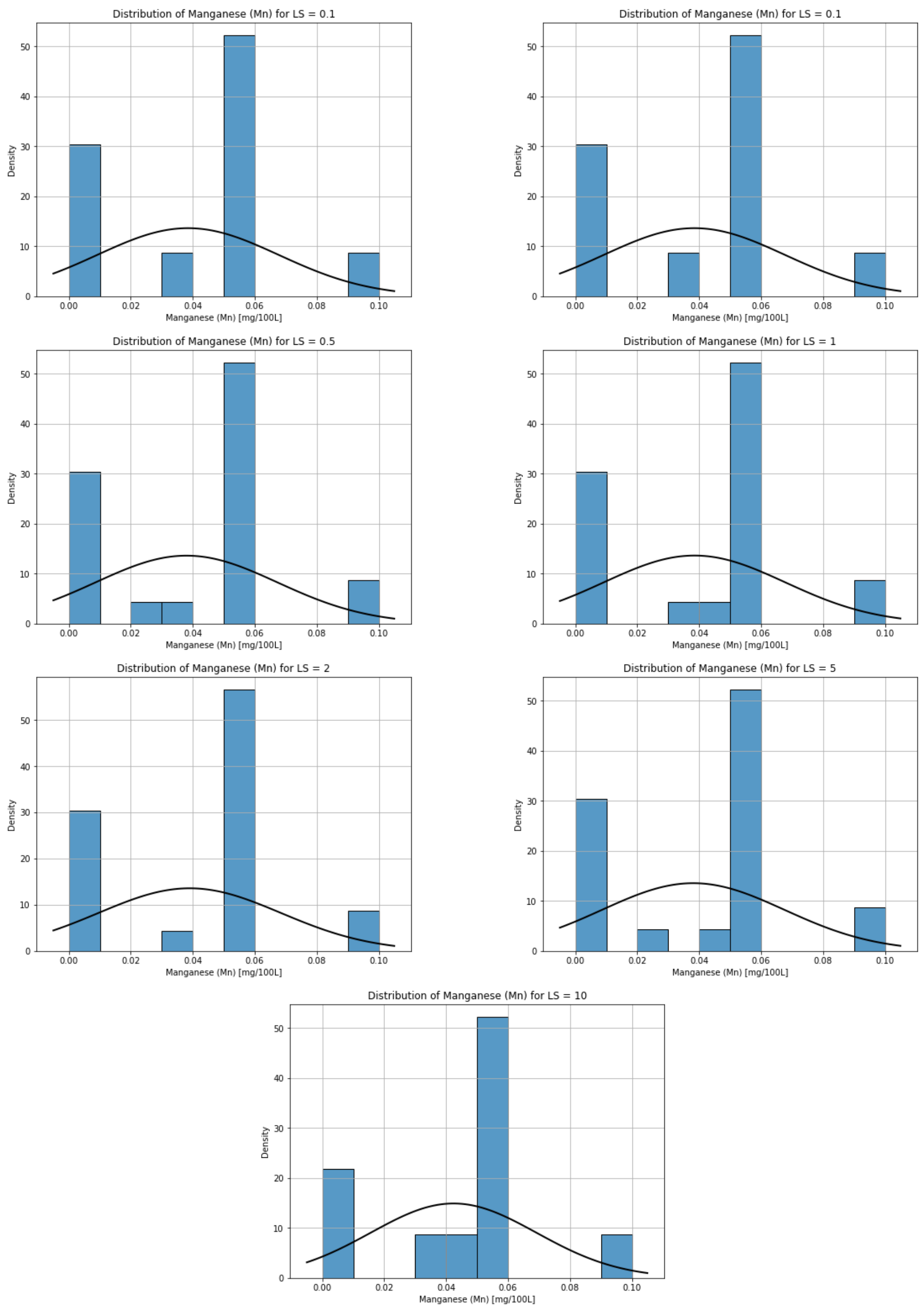


Figure A.11: Distribution of Manganese by LS

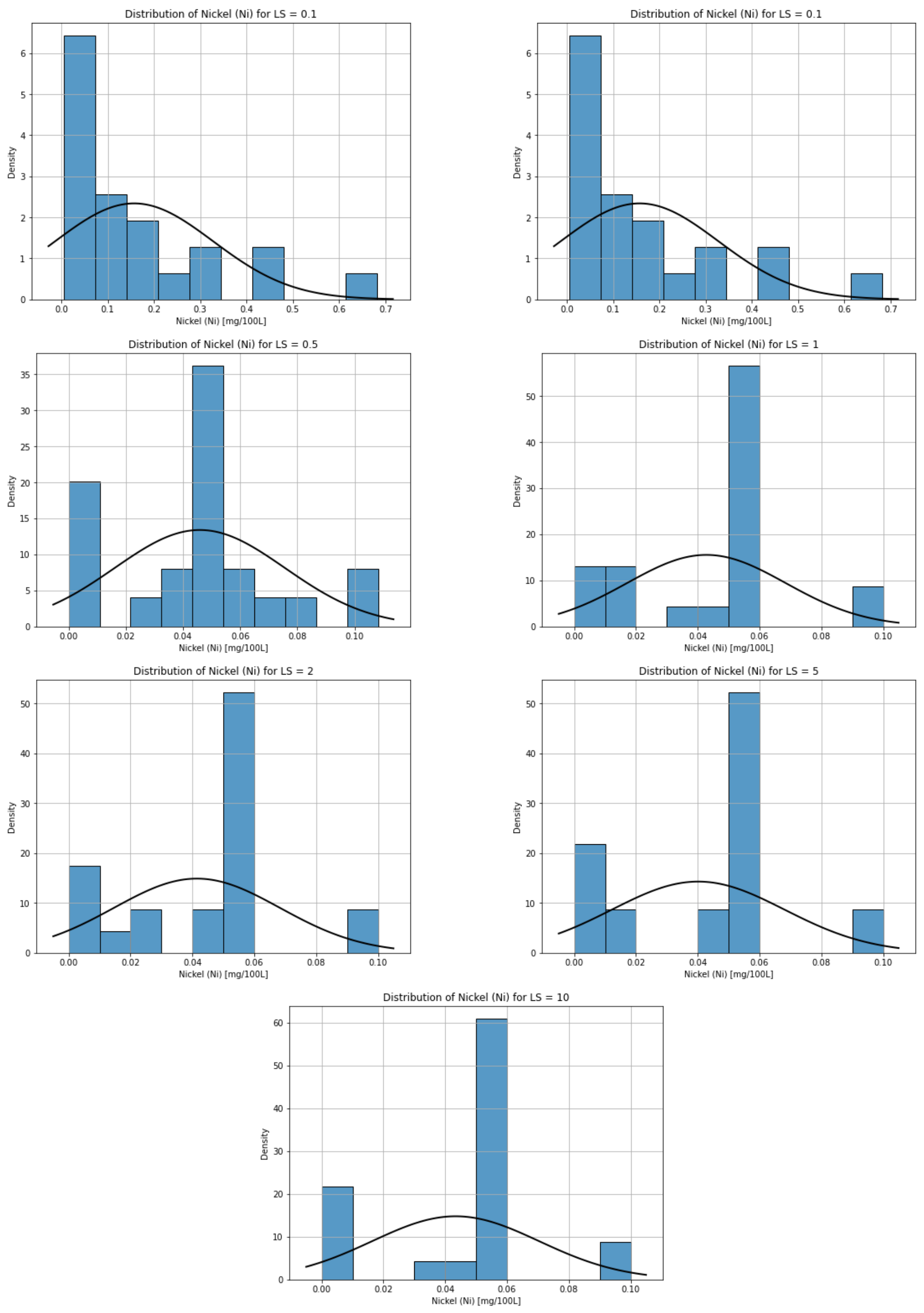


Figure A.12: Distribution of Nickel by LS

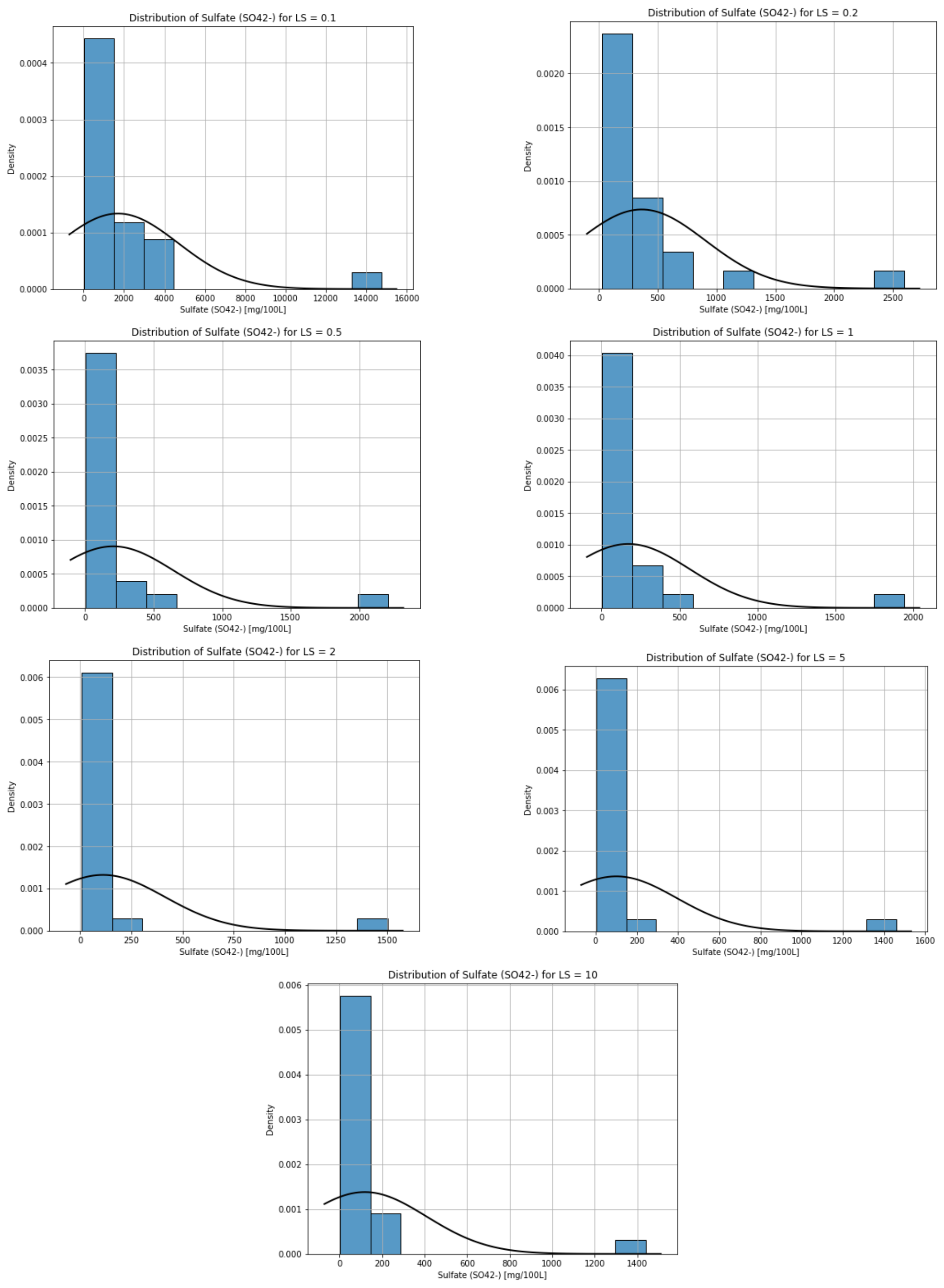


Figure A.13: Distribution of Sulfate by LS

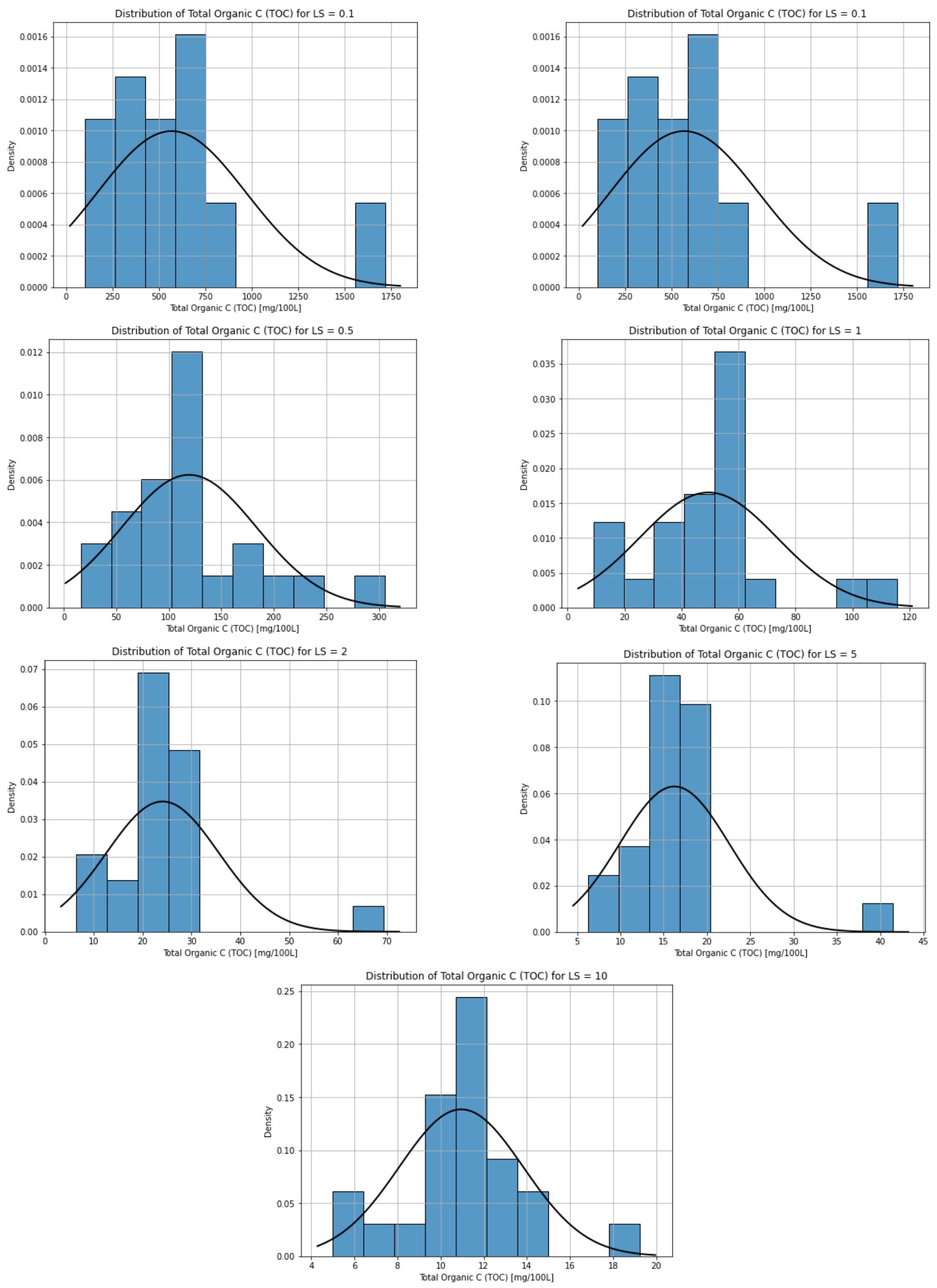


Figure A.14: Distribution of TOC by LS

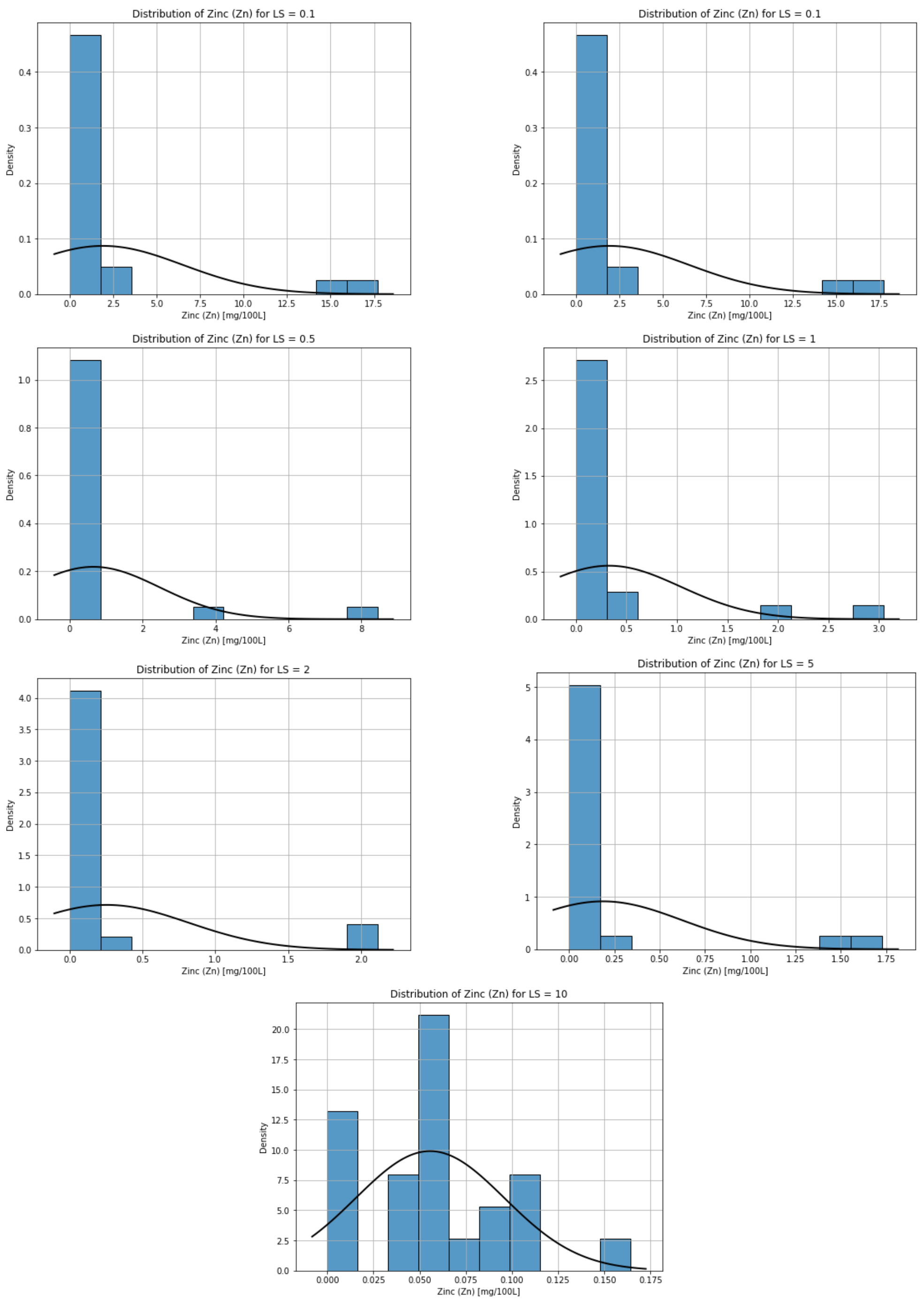
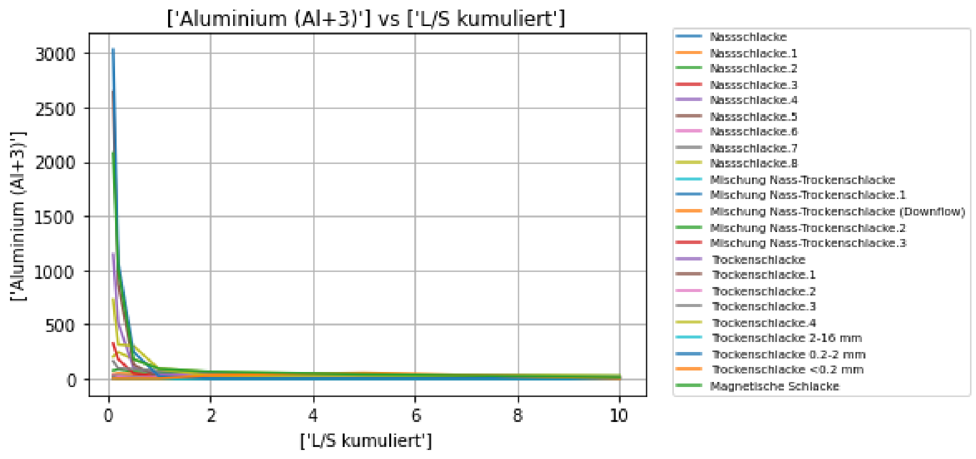


Figure A.15: Distribution of Zinc by LS

A.2 Plots of Concentrations vs. LS

The following plots display the concentrations versus LS values. For a more readable representation, see the violin plots in Section 2.1. These violin plots provide a clearer view due to the large number of experiments, but the raw data can be examined in the plots presented here.

Each element is represented by two plots: one displaying all LS values and another focusing exclusively on the target LS values, which are the values we aim to predict. This separation improves readability, particularly because starting concentrations can be significantly higher at smaller LS values compared to other data points. This disparity can obscure deviations at higher LS values. For instance, see the case of Aluminium below.



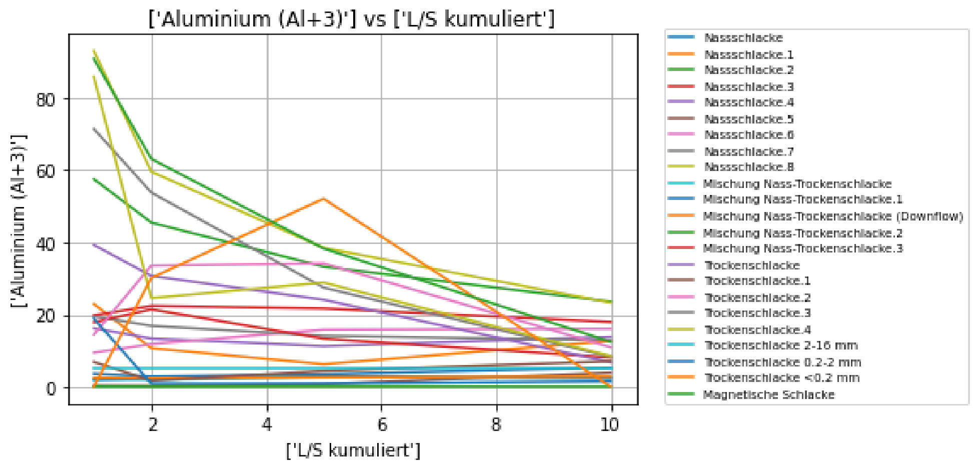
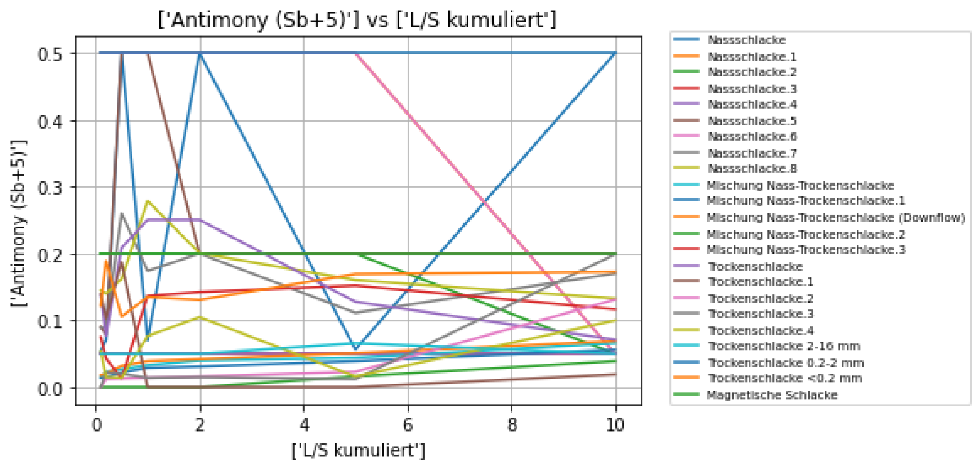


Figure A.16: Above: Concentrations over all LS values for Aluminium. Below: Except for the first three.



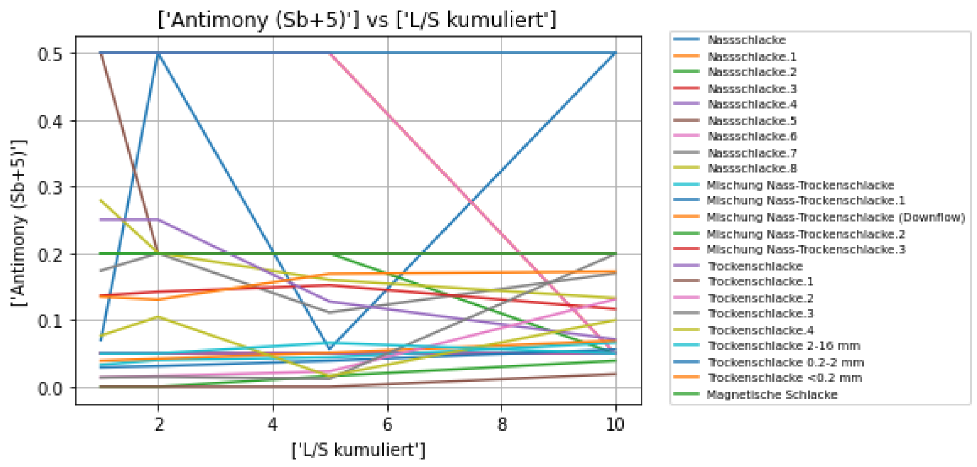


Figure A.17: Above: Concentrations over all LS values for Antimony. Below: Except for the first three.

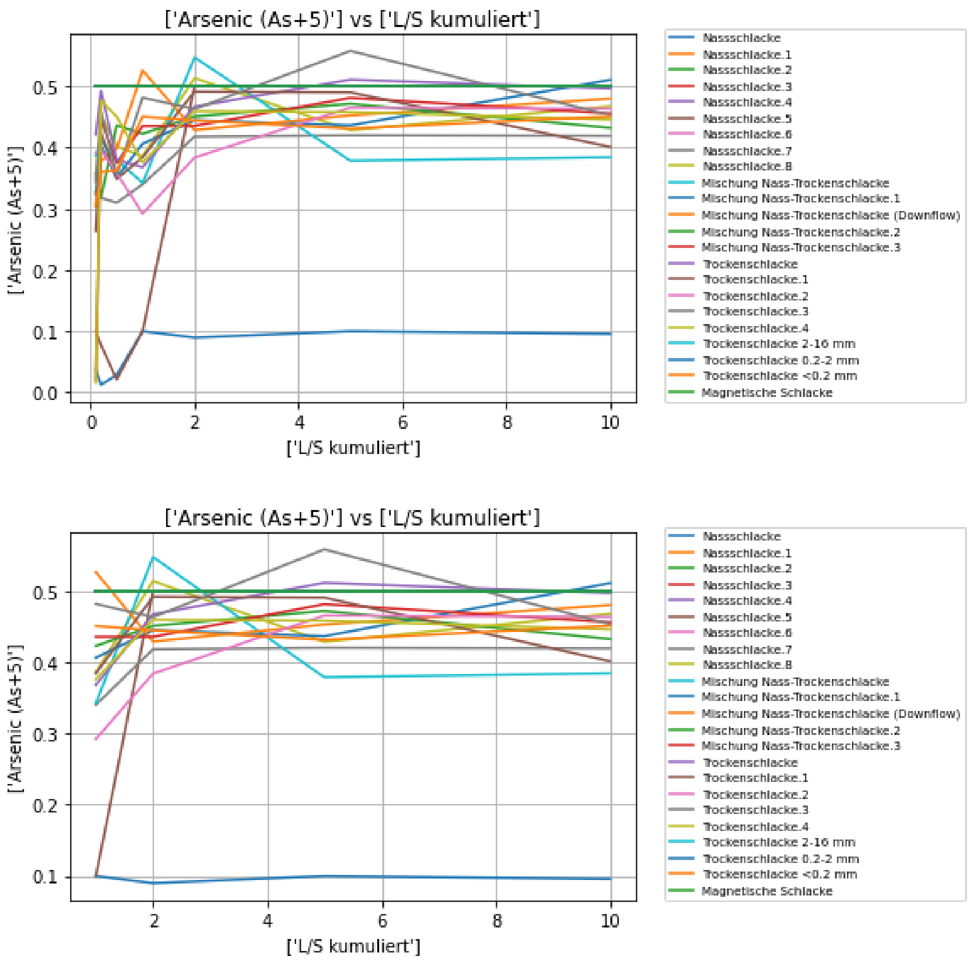
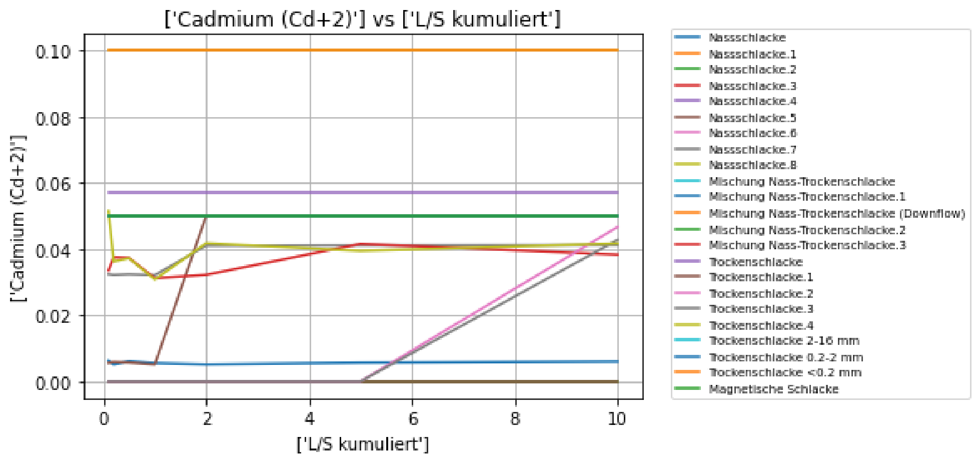


Figure A.18: Above: Concentrations over all LS values for Arsenic. Below: Except for the first three.



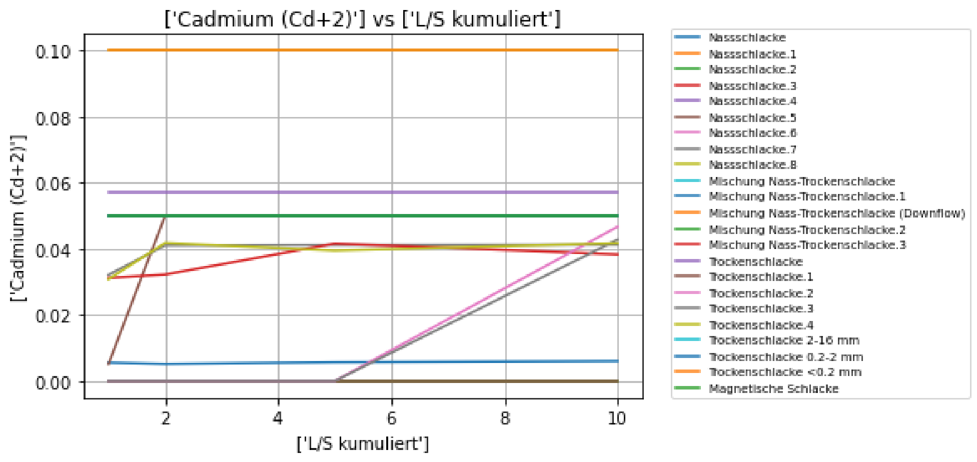
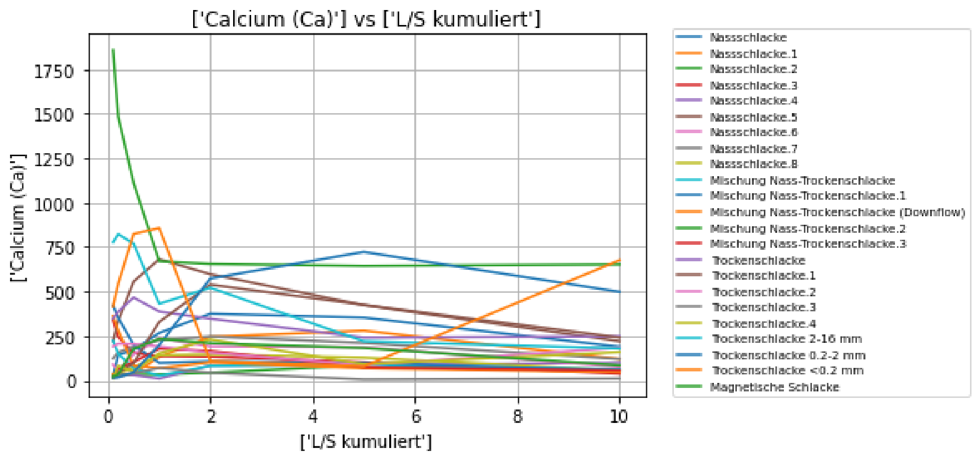


Figure A.19: Above: Concentrations over all LS values for Cadmium. Below: Except for the first three.



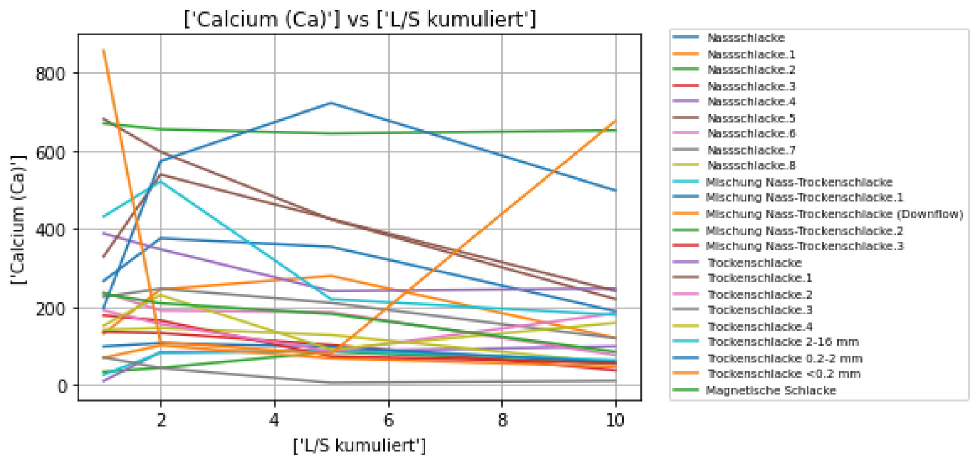
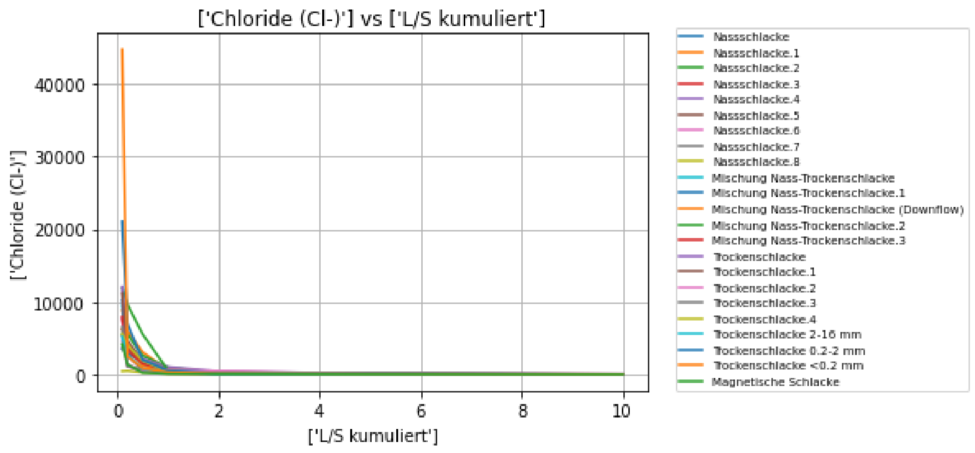


Figure A.20: Above: Concentrations over all LS values for Calcium. Below: Except for the first three.



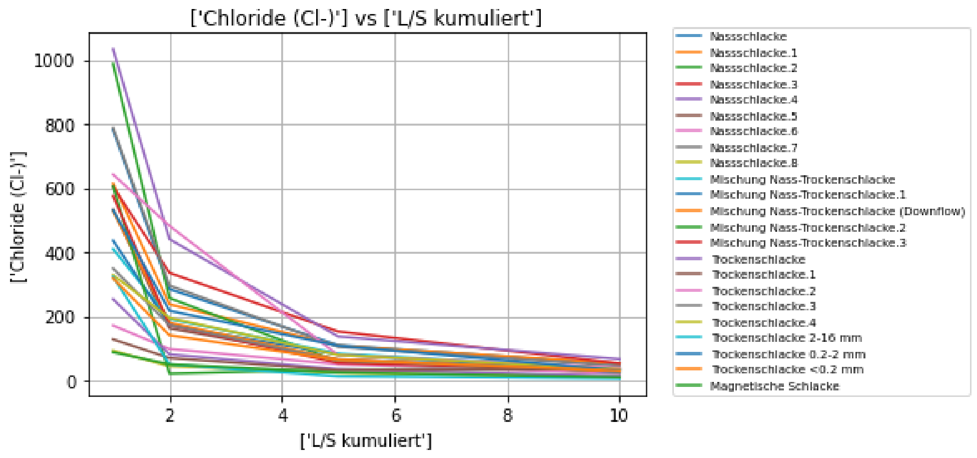
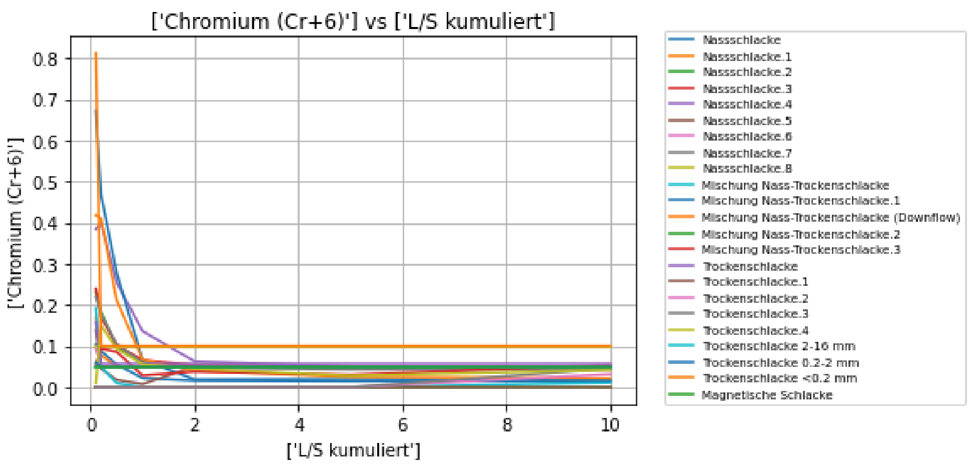


Figure A.21: Above: Concentrations over all LS values for Chloride. Below: Except for the first three.



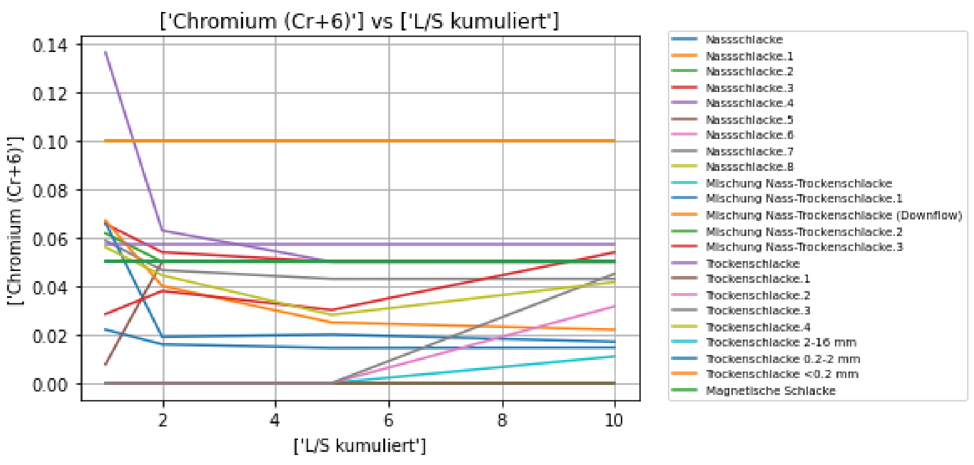
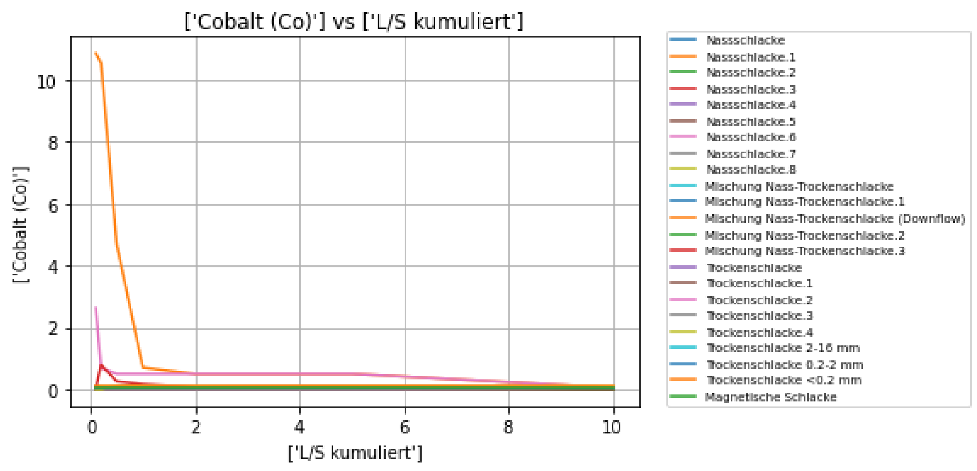


Figure A.22: Above: Concentrations over all LS values for Chromium. Right: Except for the first three.



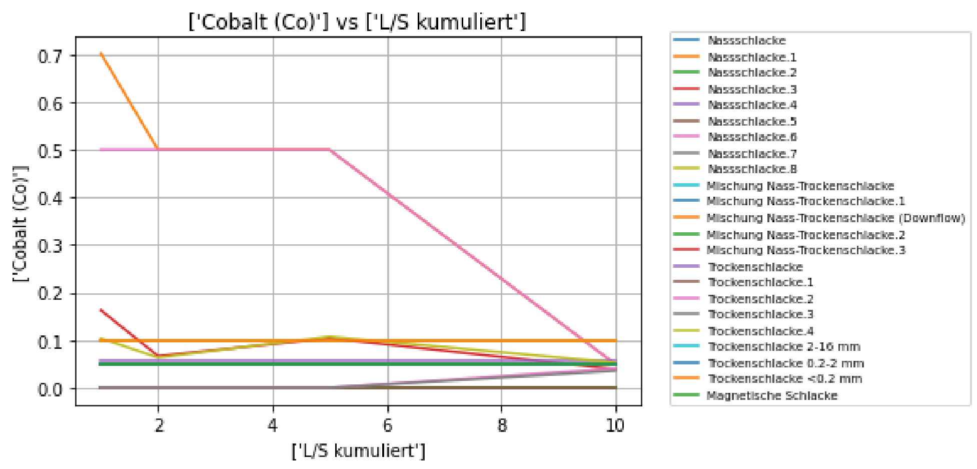


Figure A.23: Above: Concentrations over all LS values for Cobalt. Below: Except for the first three.

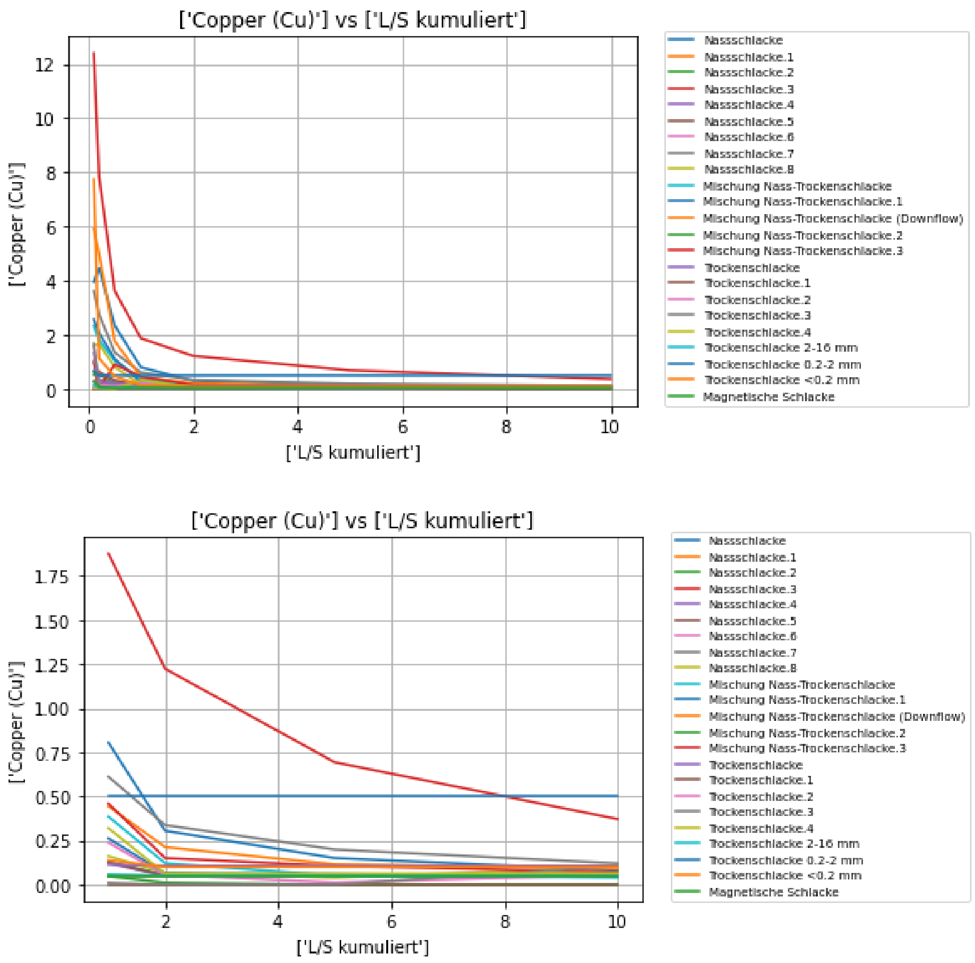
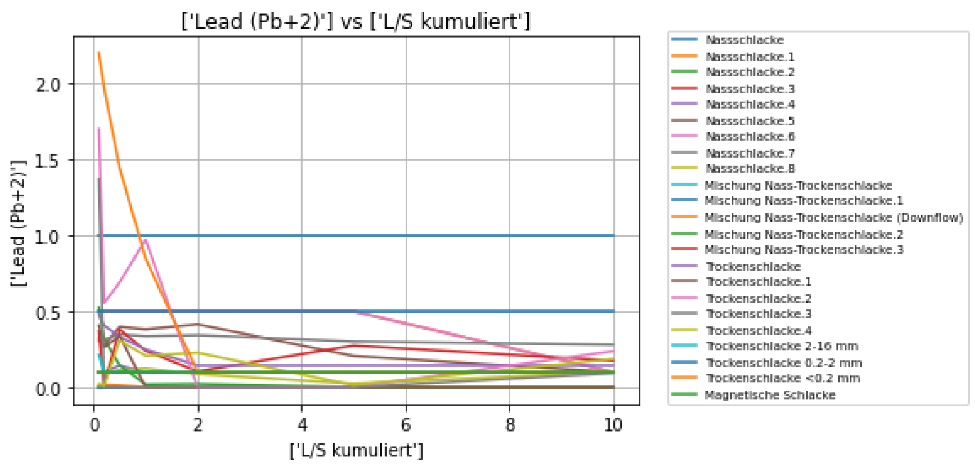


Figure A.24: Above: Concentrations over all LS values for Copper. Below: Except for the first three.



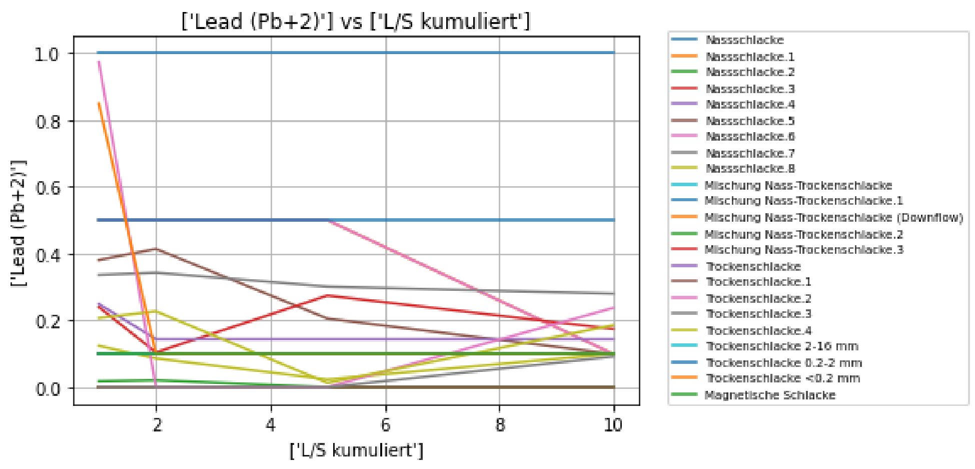
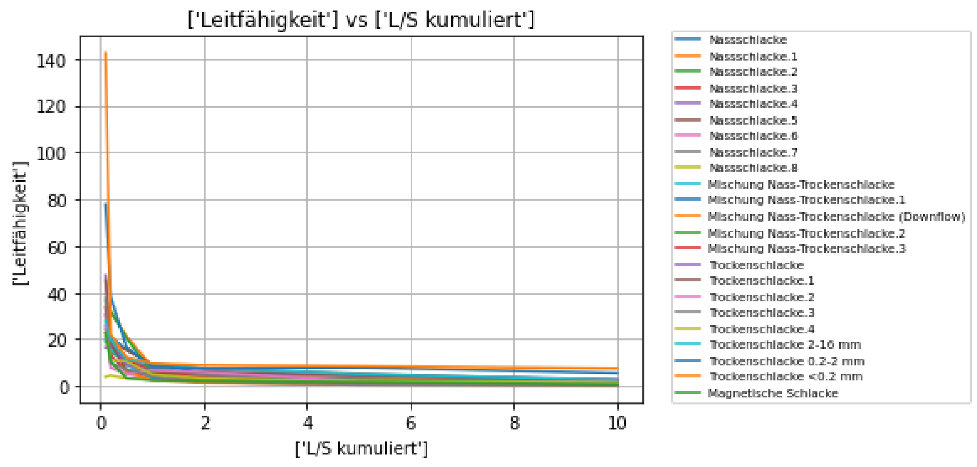


Figure A.25: Above: Concentrations over all LS values for Lead. Below: Except for the first three.



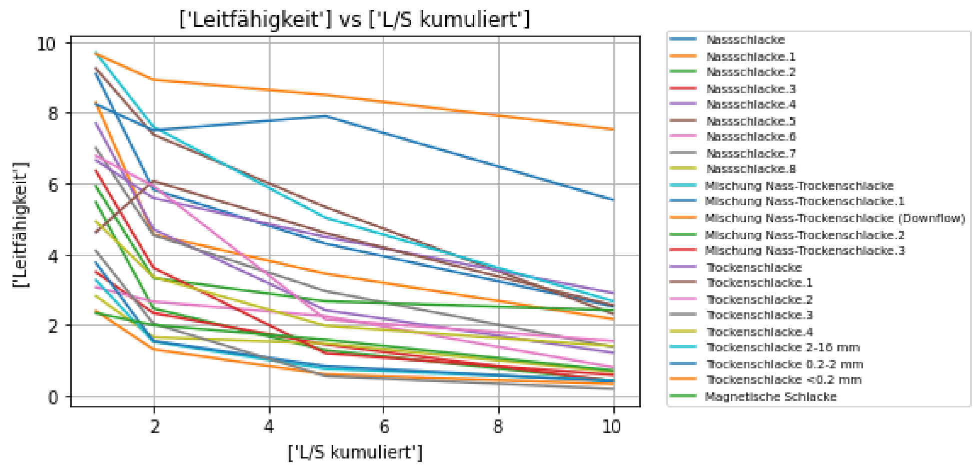
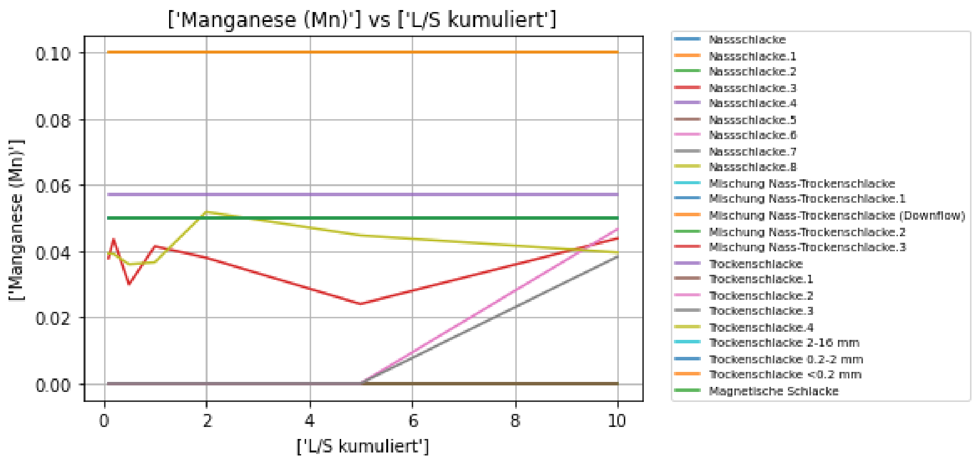


Figure A.26: Above: Concentrations over all LS values for Conductivity. Below: Except for the first three.



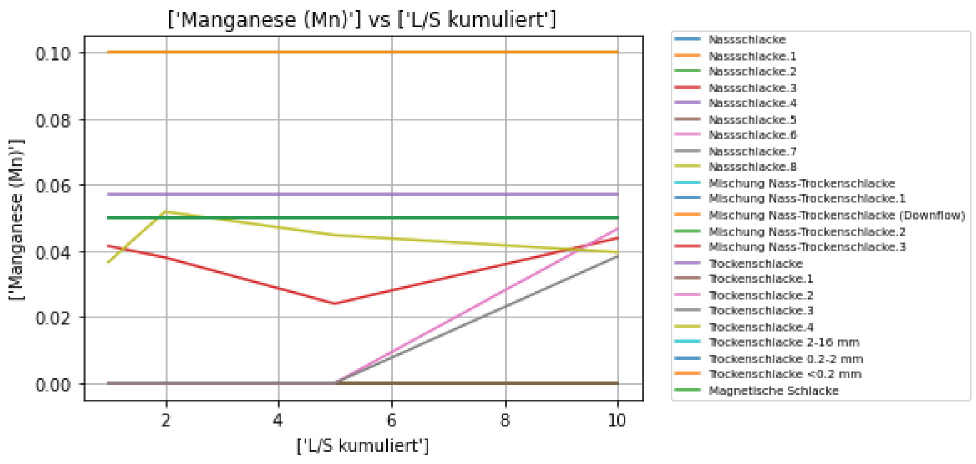
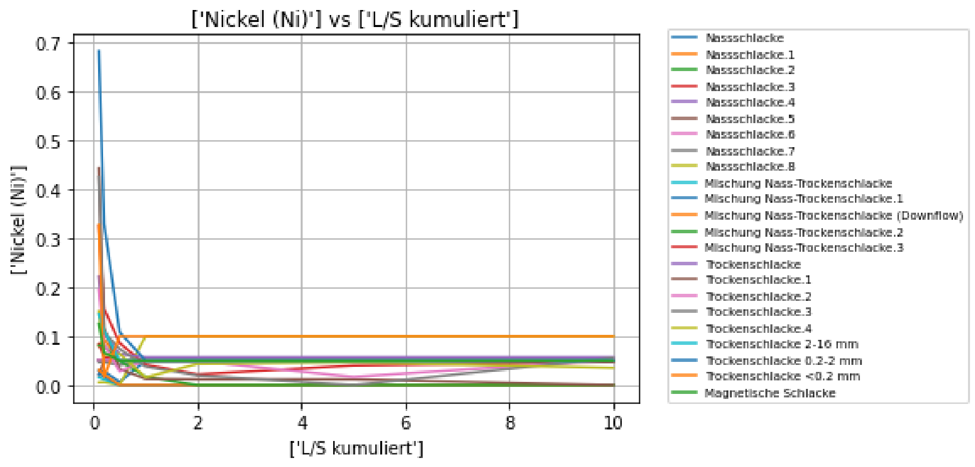


Figure A.27: Above: Concentrations over all LS values for Manganese. Below: Except for the first three.



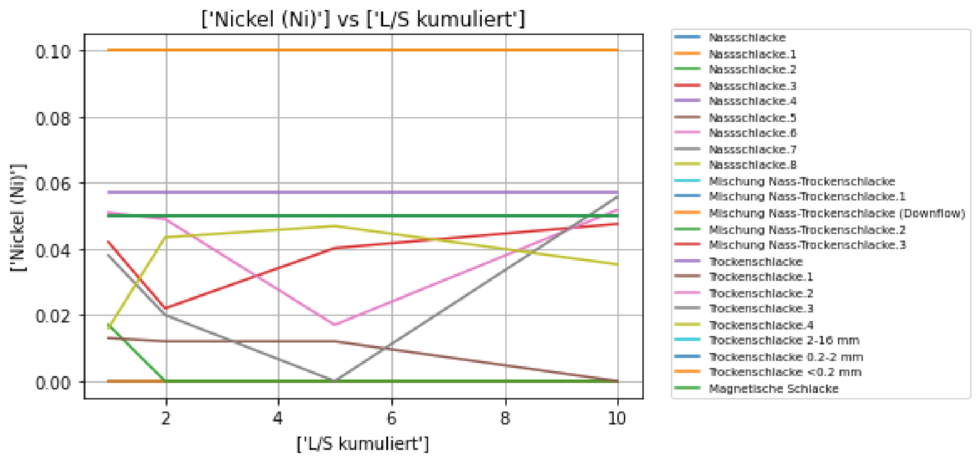
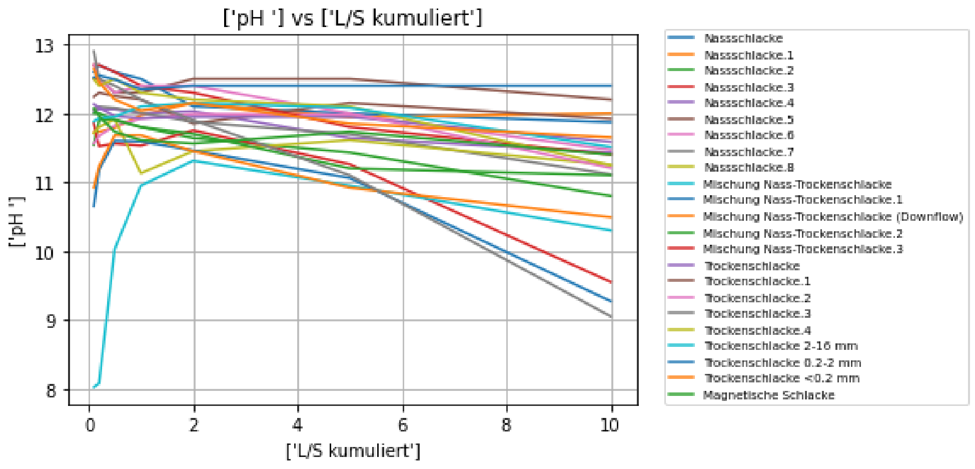


Figure A.28: Above: Concentrations over all LS values for Nickel. Below: Except for the first three.



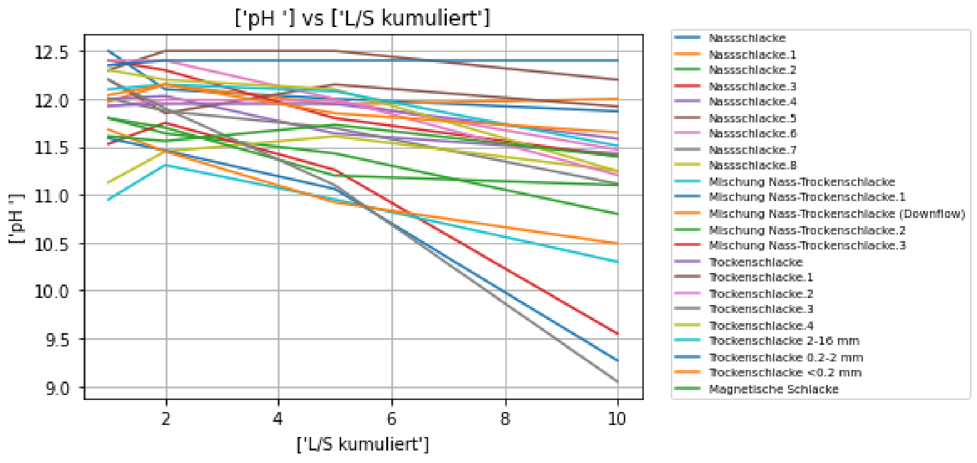
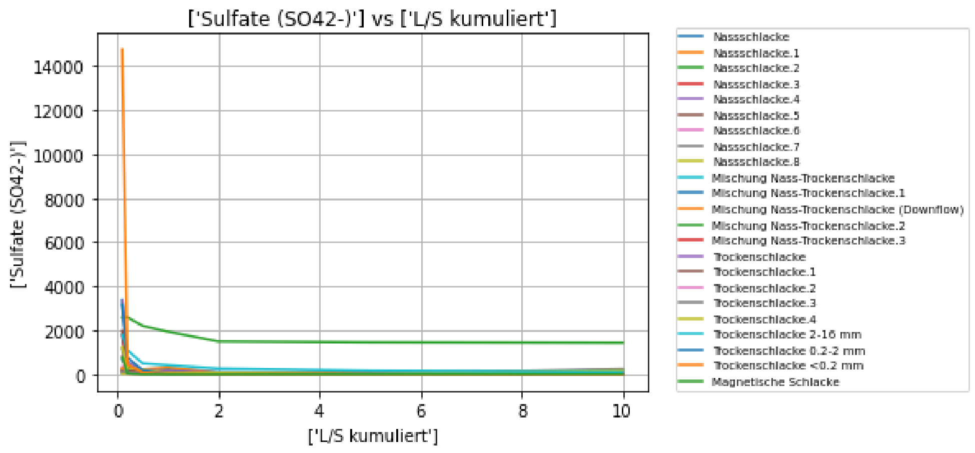


Figure A.29: Above: Concentrations over all LS values for pH. Below: Except for the first three.



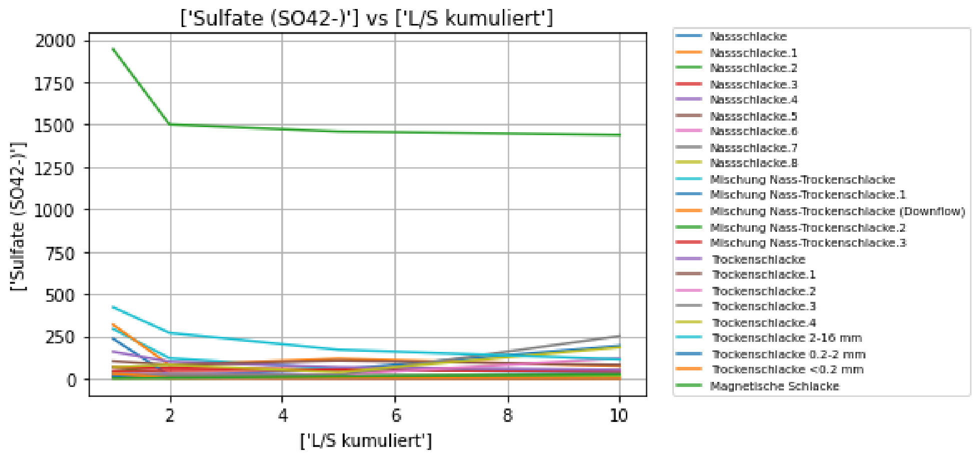
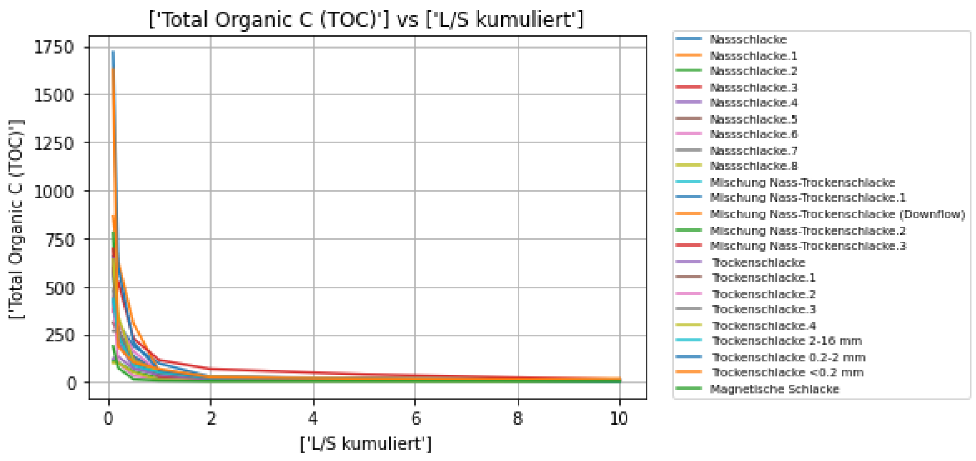


Figure A.30: Above: Concentrations over all LS values for Sulfate. Below: Except for the first three.



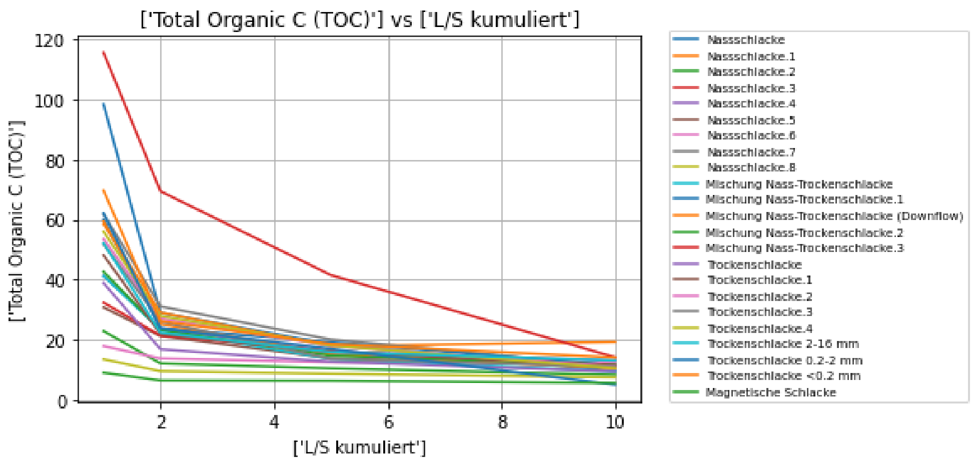
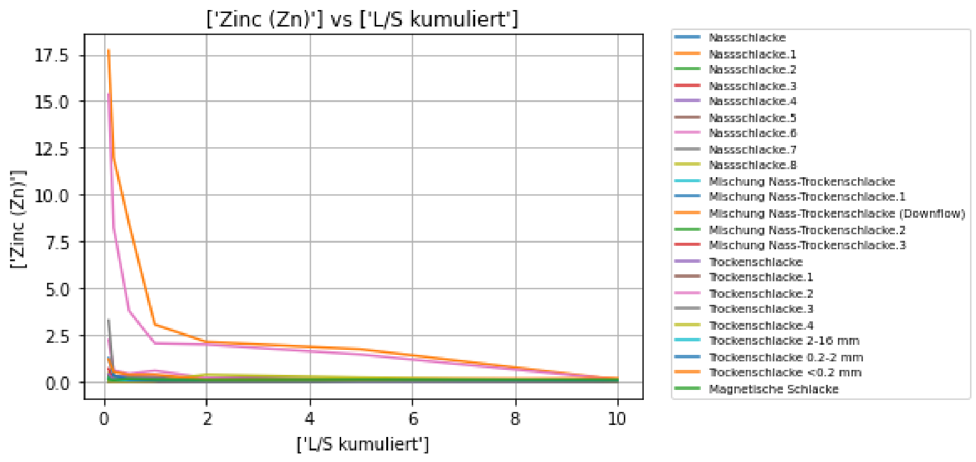


Figure A.31: Above: Concentrations over all LS values for TOC. Below: Except for the first three.



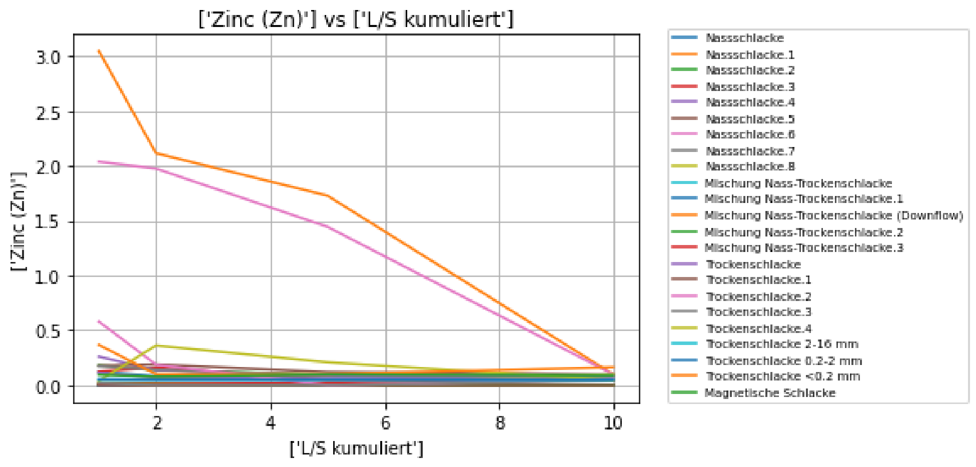


Figure A.32: Above: Concentrations over all LS values for Zinc. Below: Except for the first three.

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Declaration of originality

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor.

Title of work (in block letters):

MACHINE LEARNING MODELS FOR PREDICTING THE LEACHING OF CONTAMINANTS FROM INCINERATION BOTTOM ASHES

Authored by (in block letters):
For papers written by groups the names of all authors are required.

Name(s):	**First name(s):**
SCHUMACHER	WILLY

With my signature I confirm that
- I have committed none of the forms of plagiarism described in the 'Citation etiquette' information sheet.
- I have documented all methods, data and processes truthfully.
- I have not manipulated any data.
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for plagiarism.

Place, date	**Signature(s)**
PSI Villigen Ost, 30.08.2024	[illegible]

For papers written by groups the names of all authors are required. Their signatures collectively guarantee the entire content of the written paper.