AMI Milestone 3

Djuhera Aladin, Schuck Martin, Endres Aron, Soares Frutuoso Henrique, Brandner Michael, Griessel Alexander, Landerer Niklas, Montnacher Felix, Putz Maximilian

August 2020

Contents

1 The Data Processing Pipeline

For convenience, **all raw data sets** are gathered from the different sources into a single *all_raw.csv* file by executing the */src/data_management/proxy.py* script. The actual pipeline then **loads the data frame** and starts to process the **individual columns**. Because our data stems from **multiple sources**, columns from individual sources pose **different challenges**. Some of these need to be **interpolated** from weekly samples to daily samples for example, whereas others exhibit **high levels of noise and outliers**. After the individual processing, the pipeline also **scales all features** to zero mean and unit variance. It also **splits** and saves the data set into a **pre- and post-COVID 19 outbreak** data set. Data from within the time period of the current pandemic can be assumed to have been generated by significantly **altered mechanisms** compared to the pre-COVID 19 data. Consequently, the respective data sets **cannot be used to create a single model of the joint data**.

The pipeline is realized twice - as a **notebook** and an **executable python script**. The notebook was used to **gain insight** into the indiviudal features where **plots** allow for an initial inspection on the condition of the data and at the same time verify the **correctness of the measures** taken within the pipeline processing itself. Since we are dealing with **time series** in our data with many of them being **prone to noise**, **rolling averages** over the samples are employed to smoothen them prior to scaling. Below figure provides a graphical overview of the underlying **pipeline architecture**.

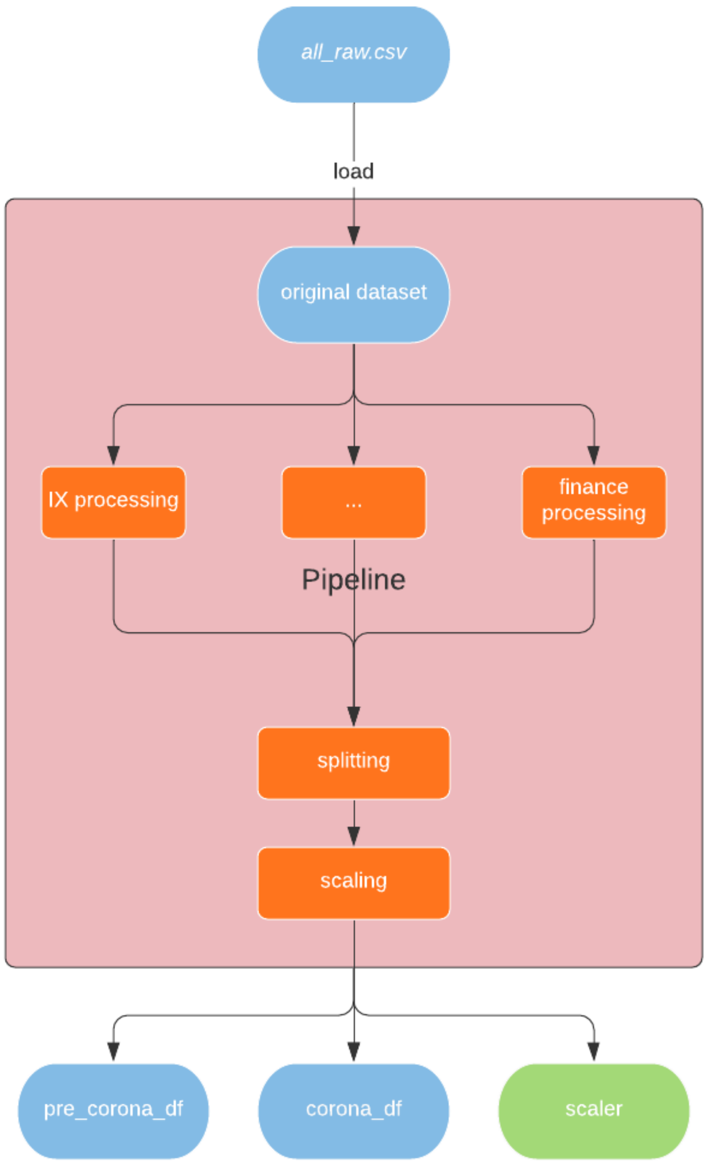


Figure 1: Pipeline Architecture

2 Candidate Machine Learning Models

2.1 Introduction

Predictive time series modeling is a more or less **standard problem** in the field of statistical modeling for which **many different approaches** exist. One of the main goals during the AMI project is to evaluate some of these **candidate model approaches** and to eventually decide for one that suits the project task at hand. This underlying chapter thus presents a quick & dirty analysis of some possible learning approaches and discusses their respective feasibility.

2.2 Extreme Learning Machines (ELM)

One very promising approach to combat **regression and classification problems** is the utilization of **Extreme Learning Machines (ELMs)**. While being not so prominent in traditional ML lectures, ELMs are widely known for **fast learning**. The latter follows from the fact that the respective **pseudo-hidden neuron parameters** (in the following just referred to as hidden) do not need to be tuned during learning and are thus independent of the underlying training data set. More so, the hidden neurons are rather **randomly generated** which consequently implies a **random initialization of its parameters** such as input weights, biases, centers, etc.

The probably most distinct property embedded in the ELM nature is the non-iterative linear solution for the respective output weights. This is mainly due to independent input and output weights, unlike in a backpropagation scenario. This ultimately renders ELMs to be very fast compared to similar MLP and SVM solutions.

Most of the discussed concepts below can be re-read in the following articles:

- High-Performance Extreme Learning Machines: A Complete Toolbox for Big Data Applications
- Extreme learning machine: Theory and applications
- tfelm: a TensorFlow Toolbox for the Investigation of ELMs and MLPs Performance

2.2.1 ELM Model Architecture

ELMs are **fast training methods for single layer feed-forward neural networks (SLFN)**. Once again, this follows from the fact that input weights w and biases b are randomly set and never adjusted. Consequently, the respective output weights β are independent. Furthermore, the **randomness of the input layer weights** aims to improve the generalization property w.r.t. the solution of a single linear output layer. The so induced orthogonality leads to weaker correlated hidden layer features.

In general, we can define an ELM model as follows. Consider a set of N distinct training samples (x_i, t_i) where i ranges between 1 and N. The SLFN output equation with L hidden neurons can then be denoted as

$$\sum_{j=1}^{L} \beta_j \phi(w_j x_i + b_j), \;\; i \in [1, N] \tag{1}$$

with ϕ being the activation function, w_i the input weights, b_i the biases and β_i the respective output weights. Consequently, the relation between the network inputs x_i, the target outputs t_i and the estimated outputs y_i is given by

$$y_i = \sum_{j=1}^{L} \beta_j \phi(w_j x_i + b_j) = t_i + \epsilon_i, \;\; i \in [1, N] \tag{2}$$

where ϵ denotes the noise comprised of random noise and certain dependencies on hidden variables excluded from the inputs x_i. This process can be re-examined in below figure.

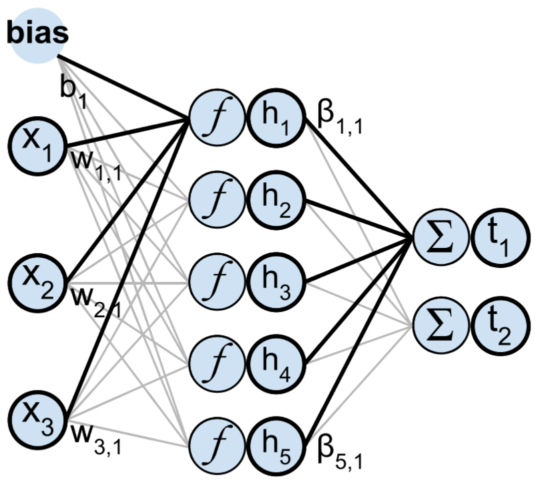


Figure 2: SLFN Process

2.2.2 Computation

Before discussing the simple computation technique behind ELMs, it is reasonable to first discuss the processes behind the respective hidden neurons as well as a compact matrix formulation of the problem.

2.2.2.1 Pseudo-Hidden Neurons

In general, the hidden neurons **transform** the underlying input data into a different representation. This is usually done in two steps:

1. The data is projected into the hidden layer using the input layer weights and biases.
2. The projected data is transformed using a non-linear transformation function.

In particular, using the above **non-linear transformation**, the learning capabilities of the ELM can be greatly **increased**. After transformation, the data in the hidden layers h_i can be used to find the output layer weights. Another **practical advantage** is that the respective transformation functions are not constrained by type, that is they can be selected to be very different and even non-existent. Furthermore, since the neurons are linear, they consequently adapt and learn linear dependencies between data features and targets which happens directly without any nonlinear approximation at all.

Note however, that other types of neurons have also found application in ELMs such as **RBF neurons** with nonlinear projection functions. These can be used to compute predictions based on similar training data samples in order to solve tasks with some more complex dependencies between data features and targets.

2.2.2.2 Compact Matrix Notation

ELMs exhibit a **closed form solution** in which the hidden neurons are comprised in a matrix H. The network structure itself though is not noticable in practice meaning that there is only a single matrix that describes the projection between two spaces. The projections for the input $(X \cdot W)$ and the output $(H \cdot \beta)$ are connected through a nonlinear transformation as follows:

$$H = [H_1 \,|\, H_2] = [\phi_1(XW_1 + b_1) \,|\, \phi_2(XW_2 + b_2)] \tag{3}$$

2.2.2.3 Solution Computation

In general, ELM problems are usually **over-determined** ($N > L$) with the number of training data samples being much larger than the number of selected hidden neurons. In all other cases ($N \leq L$) it is recommended to use regularization in order to obtain a better generalization performance.

Nevertheless, a unique solution can be found using the pseudoinverse:

$$H\beta = T \tag{4}$$

$$\beta = H^{\dagger}T \tag{5}$$

$$H^{\dagger} = (H^{\dagger}H)^{-1}H^{T} \tag{6}$$

2.2.3 Conclusion

ELMs have very promising and efficient properties. They have been proven to be very useful for regression tasks as needed in our project. Nevertheless, there have been some reports on negative effects such as

- Bad initial randomization
- Speedy performance but somtimes low accuracy

In particular, it has to be pointed out that ELMs only operate on **one hidden layer** in contrast to general DL approaches. In order to achieve a very high accuracy and approximation, this might not be necessarily the best performing choice considering the amount of COVID-19 data that we have gathered throughout the collection process.

Nevertheless, there are two very high-performance **Matlab and Python implementations** in form of **ready-to-use toolboxes** discussed in above articles. They promise an automatic model structure selection as well as the application of regularization techniques. Furthermore, many approaches using self-written Python code have emerged online.

However, the main argument that seems to disqualify ELMs initially - at least from our user point of view - is that only **one group member** has even had any experience at all using them. Furthermore, even though the concepts of ELMs seem promising and effective, most of the group's ML experience was gained in Python using more popular frameworks such as **TensorFlow and PyTorch**. However, we decide to continue pursuing an ELM approach as its operation on only one hidden layer seems very interesting w.r.t. the project scope and the amount of gathered data.

2.3 Gaussian Processes (GPs)

Another very promising approach which solves **regressional time series problems** is the use of **Gaussian Processes (GPs)** for which a handful of implementations is given in the **scikit ML library**. In particular, there have already been attempts to model and forecast CO2 emissions using GPs, as done *here*. This not only **motivates** to further investigate GPs for our project but also demonstrates a successful application, ultimately rendering this approach as **highly plausible** to achieve our specified project target.

2.3.1 GP Model Architecture

GPs are a very generic class of **supervised learning methods** which are designed to not only solve **regression but also probabilitsic classification problems**. In general, a GP is a **stochastic process** and thus a collection of random variables, e.g. in the time or space domain. Note that every such finite collection of random variables has a **multivariate normal distribution**, that is every finite linear combination of these random variables is **strictly normally distributed**. As such, every GP can be compactly described by the **joint distribution** of all those random variables and is thus strictly specified by its **mean and covariance functions**.

A GP can thus be described as a functional mapping of random variables x_i

$$f \sim \mathcal{GP}(m, k) \tag{7}$$

with mean function $m(x)$

$$m(x) = E[f(x)] \tag{8}$$

and covariance function $k(x, x')$

$$k(x, x') = E[(f(x) - m(x)) \cdot (f(x') - m(x'))] \tag{9}$$

Most ML algorithms that make use of GPs often apply **lazy learning approaches** in order to measure the **similarity** between the respective evaluation points. To this, the so-called **kernel function** is examined which aids to predict the value for a future, e.g. time series point. The so obtained prediction - in form of a **distribution** - not just provides an estimation but also contains some **uncertainty information** which is embedded in the **one-dimensional Gaussian distribution**. The same holds for multidimensional predictions where the GP is multivariate and for which the respective multivariate Gaussian distributions are the corresponding marginal distributions at the current evaluation point.

2.3.2 Pros and Cons

The advantages of GP models can mainly be summarized as follows:

- The model prediction interpolates between the observations for regular kernels.
- The prediction is probabilistic and allows for an analysis of confidence intervals which in turn aids to decide whether refitting is necessary. The latter one can thus be solved in an online fashion.
- There is a certain versatitlity due to the possibility to choose differently specified kernels.

However, there are also some severe disadvantages of GP models:

- Non-sparsity: The models use the whole sample space and every feature information in order to perform a prediction.
- Low efficiency in high-dimensional (> 12) spaces.

2.3.3 Conclusion

While GP models bring many different advantages and are also quite broady represented in the desired frameworks to be used during the project, once again **only few group members** have actively dealt with both the **theory and practical implementation** of such models. Thus, diving deeper in the more complex stochastic modeling theory will certainly **take up more time** than we initially desired to give for the ML core development. As such, we decide that more time should be spent **analyzing, optimizing and troubleshooting** the model output which is frankly the more **exhausting** part for a successful model application. Therefore, we live by the notion that the **more simple the model the better**. Since most of the group members have gained considerable experience in DL modeling scenarios, these are to choose and GP models will thus not be considered beyond an initial preliminary assessment.

2.4 LSTM RNNs

In summary, the work completed so far in terms of online models builds a good base for better online learning in the next iteration. Additionally, the completed work is sufficient evidence of the adapational power of online learning. There are many time-series modeling approaches using **recurrent neural networks (RNNs)**. However, while these seem to achieve some **very high accuracies**, they also suffer from severe drawbacks which might also affect our project scope. To be precise, one particularly dangerous disadvantage of conventional RNNs is their **short-term memory capacity**. In order to combat this drawback, **long short-term memory (LSTM) RNNs** have been introduced that incorporate a significantly **greater (longer) memory capacity**.

Unlike feed-forward neural networks, LSTMs have **feedback connections** and are not only able to process **single data points** but also **entire sequences** - a character trait especially needed for our project implementation. A huge proportion of LSTM associated model applications so far have been based on time series data. In particular, some models - as presented *here* - have been constructed that actively address the COVID-19 pandemic and use time-series data for accurate predictions. This once again **motivates** to further discuss LSTM RNNs as a valid approach for our project realization.

2.4.1 LSTM Model Architecture

In general, RNNs can be represented in a **chain-like form of repeating modules** which incorporate loops and interconnections. LSTMs in particular introduce a repeating module which has a more advanced structure containing **several neural network layers**. The general chain layout for the repeating module of such LSTMs is depicted below.

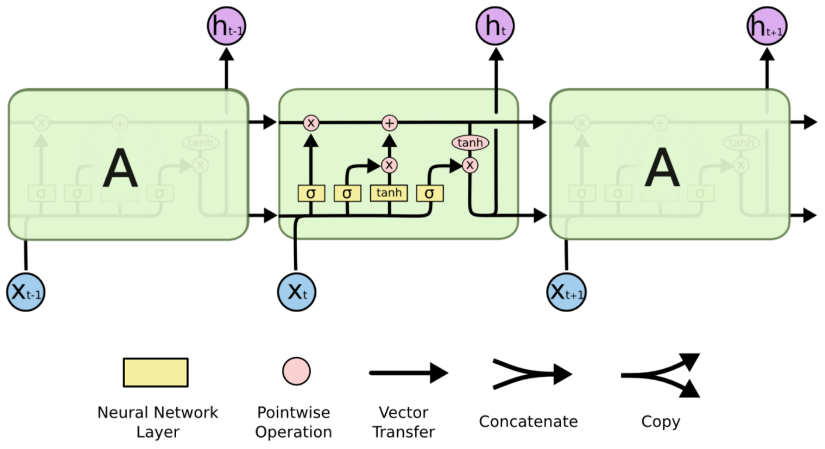


Figure 3: LSTM Layout

The core of LSTM based RNNs is the **cell state** C which is defined by the horizontal line on the top of the respective module. In particular, the cell state interacts **linearly** with other elements throughout its way.

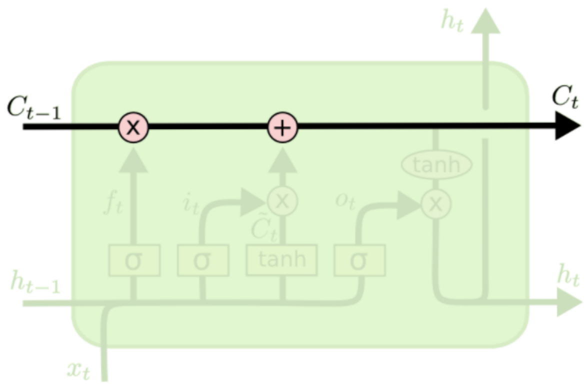


Figure 4: LSTM Cell State

The main feature of LSTMs is the ability to **add and remove** certain information w.r.t. the cell state, that is information can be **sequentially** added to and removed from it. This mechanism is regulated by so-called **information gates**. These gate structures are usually composed of **sigmoid neural net layers and pointwise multiplication operators**.

In general, we can classify three different types of information gates:

- **Forget Gates** decide what information will be thrown away from the respective cell state C.
- **Input Gates** decide what **new information** is going to be added.
- **Output Gates** decide what information has to be output. This is highly dependent on the cell state and essentially a filtered version of C.

2.4.2 Conclusion

To summarize, LSTM RNNs have a **very high and most importantly proven performance** w.r.t. time series modeling. In particular, there are also **many guides and tutorials** on how to realize and troubleshoot (optimize) LSTM RNNs using Python. Due to their **DL character**, they are furthermore very **familiar** to the group members as multiple layers can and should be constructed. In particular, there is also **no higher effort in understanding** the advanced LSTM architecture and its many variants as they can be **astractly considered** as "just elements in a RNN". As such, LSTM RNNs are currently the most promising solution to the time series model approach that we intend for our project scope. However, most of the aforementioned advantages can only be efficiently exploited if the underlying data is of good quality as LSTM RNNs perform predictions on predicted outcomes for which high quality data is essential.

2.5 Decision

After having reviewed above candidate ML models that **certainly would all suffice** to achieve the project task at hand, a final decision is hard to make. **LSTM-based RNNs** in particular seem to be a very powerful solution to the problem. Furthermore, the **matureness of LSTMs** compared to Extreme Learning and GP models as well as their **familiarity to every group member -** as they can be seen as an extension to RNNs - are two of the main driving factors fueling the decision for it. However, one major drawback that has emerged throughout the **data collection phase** is that not all of our data sets are of a particularly high quality due to **noise** and other factors. Thus, we believe that performing predictions using LSTM-based RNNs **will not exploit above high-feature advantages** nor will the predictions be of a particularly high accuracy due to the **low-quality data** in some cases. Furthermore, we believe that above candidate approaches need to be compared to more **traditional regression approaches** such as linear regression and decision trees which are well-known to the group members. In particular, necessary obstacles such as **troubleshooting, hyperparameter optimizing and model verification** are well understood for these regression model classes. In conclusion, very powerful model architectures have been discussed and examined. However, the preliminary performance assessment will determine whether complex approaches **outperform** standard regression models. A **final decision** on the model architecture can certainly only be made **after the assessment**.

3 Preliminary Assessment of Candidate ML Models

3.1 Model Assessment Framework

In order to compare the different models that we intend to employ on our acquired data sets, we implemented a **model comparison process** based on a **5-fold cross-validation** using scikit-learn which compares the **average MSE** of our models. The number of cross-validation intervals was chosen such that the time series has **enough data to process** with a certain variance. However, due to the fact that the data sets are more or less **very scarce**, we want to maximize the ability to exploit as much information as possible. Furthermore and due to the **inherent correlation of data points in time series** in general, we decided for a **non-shuffling cross-validation** attempt. We test on a single time series during the model assessment, which is the tech stock time series from 2017 to 2020. The respective code for model assessment can be found in */src/inference/model_comparison.py*.

Nevertheless, we have to consider - once again - the fact that the **quality** of our acquired data is **not good enough** for the utilization of above discussed, more advanced ML concepts. Therefore, a predetermined decision for much easier learning schemes such as **linear regression, decision trees and time series forecasting** using **FB Prophet**, a framework which implements trend estimtion with additive decomposition, has been made which inherently performs better w.r.t. scarce and noisy data.

Since the adaption of our currently most promising approach, the trend estimator by **additive decomposition**, to the scikit-learn model interface for cross-validation poses a significant challenge. We ran the model comparison process on standard sklearn regressors to verify its functionality. The results can be seen below where the different colors represent the respective different linear regression models.

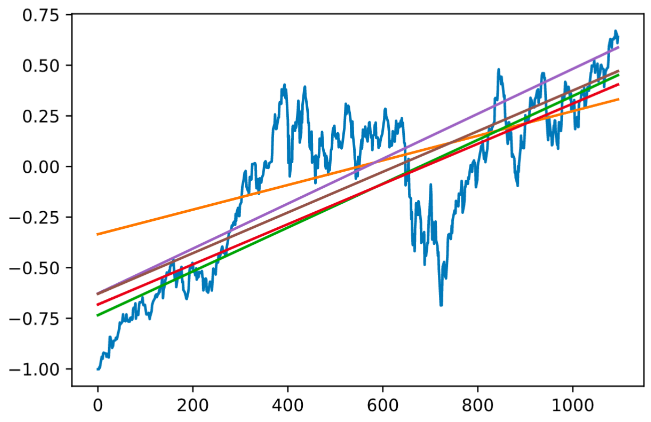


Figure 5: All 5 Linear Regression Models from the Cross-validation of the Tech Stock Data Set

Of course linear regression represents the **most primitive regressor** one could ever imagine. We are inclined to believe that the adoption of a **trend estimator** as seen in Figure 6 will signifficantly improve prediction quality.

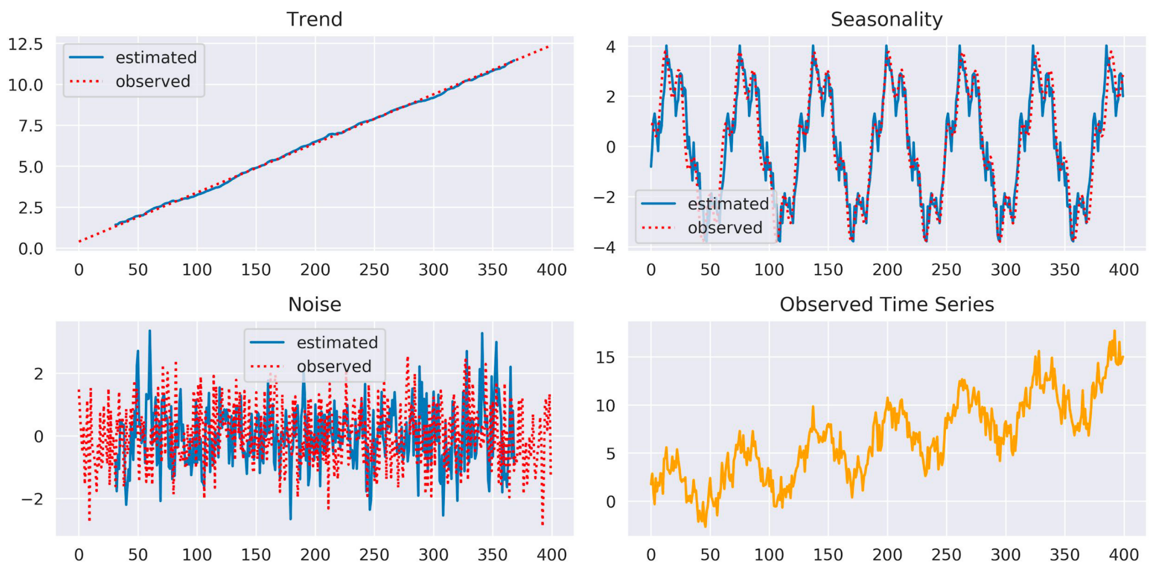


Figure 6: Trend estimator, Source: Applied Machine Intelligence Lecture Slides

4 Front End Developement

4.1 Design Goals

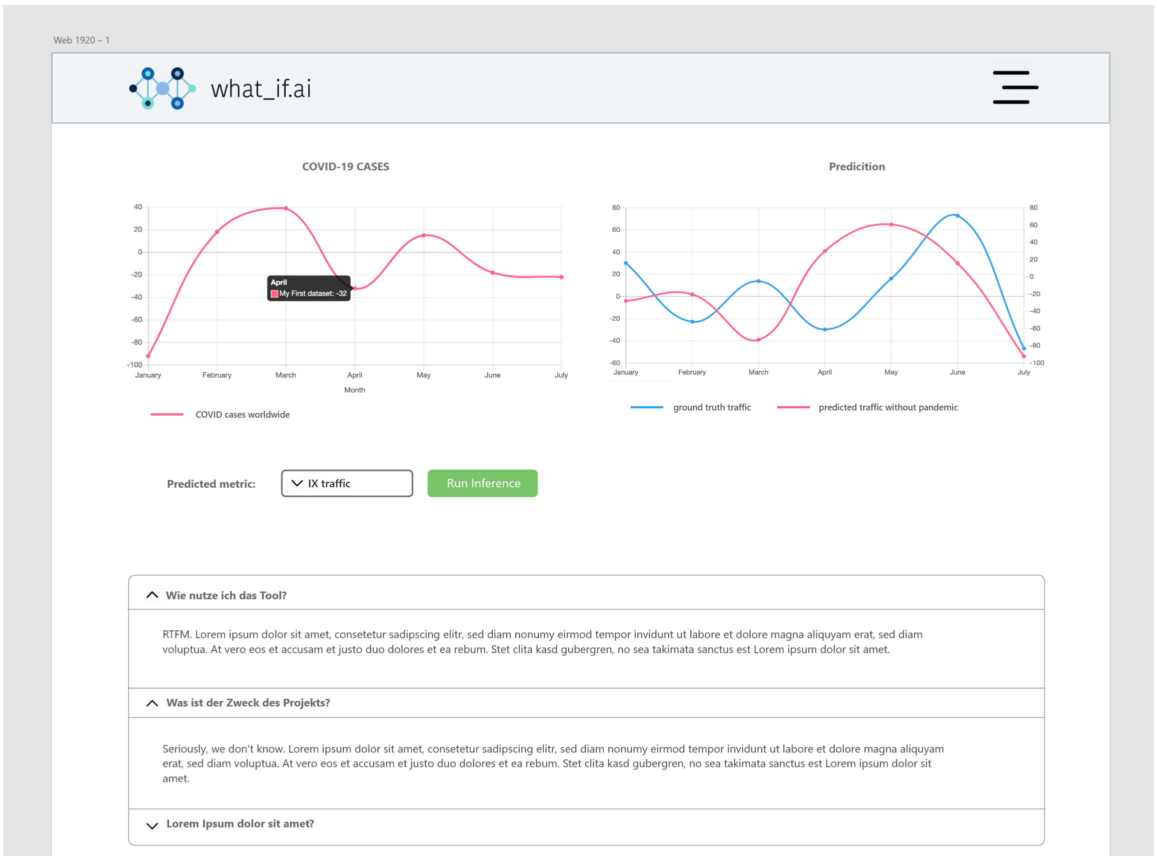
Before starting to work on the concept, we agreed on the following set of design principles:

- Ease of use
- Responsiveness and functionality **on all devices** - no matter the screen size
- Intuitive and smooth user input feedback using the interface with notifications for error messages, etc.

4.2 First UI Concept

Initially, we planned on showing two graphs in a simple overview as seen below.

- The graph on the left displays the number of COVID-19 cases over the respective time period and is **draggable**, ultimately allowing the user to **explore different scenarios**.
- The graph on the right hand side displays the **model prediction** compared to the **ground truth data**. Initially, it was supposed to update in a **real-time fashion** with changes being made to the input data of the graph on the left.



Wie nutze ich das Tool?

RTFM. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

Was ist der Zweck des Projekts?

Seriously, we don't know. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

Lorem Ipsum dolor sit amet?

Figure 7: First Mockup, created in Adobe XD

4.3 Refining the User Interface

However, as our project proposal has **slightly changed** over time due to insufficient data sets, so did our interface concept. We decided to **leave out** the possibility to **drag** above COVID-19 case count. This is mainly justified due to the **low general impact** that it has on the model itself. Furthermore, the case count is highly dependent on **political decisions** which vary greatly by country and are thus **hardly predictable**.

Consequently and due to the experience of several group members in web developement, we started our second design iteration with the goal to not just creating a **mockup** but rather a **fully functioning interface prototype**. In particular, we made use of the charting library **chart.js** for graphs, **axios** for asynchronous communication with our python backend which was built using **flask**. In order to visually glue everything together, we used **vue.js** and **material web components**.

However, as UI implementation is not particularly within the scope of the third milestone, this report will not cover any further technical details on the web interface implementation. The current **second prototype** as well as a sample **error notification** in case of a failed connection attempt are depicted below.

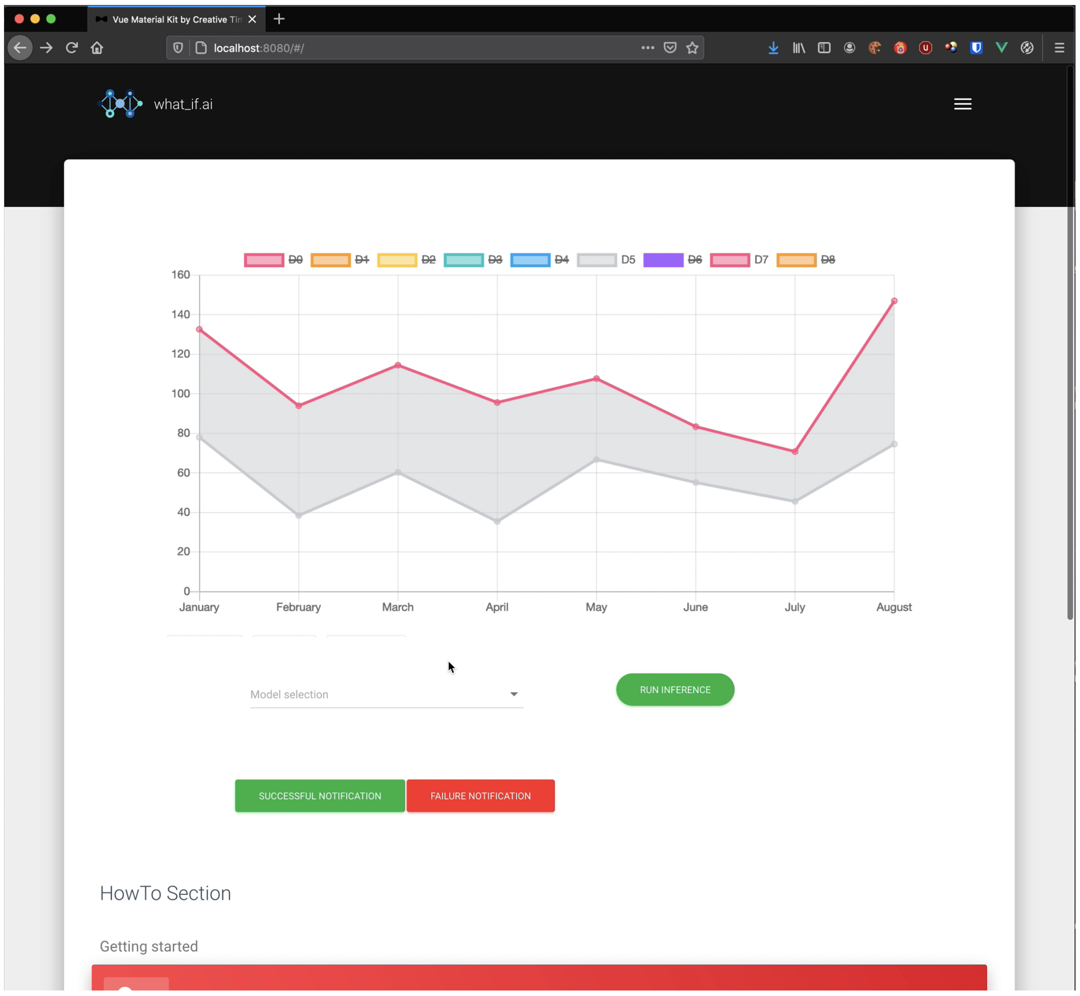


Figure 8: Second Design Prototype

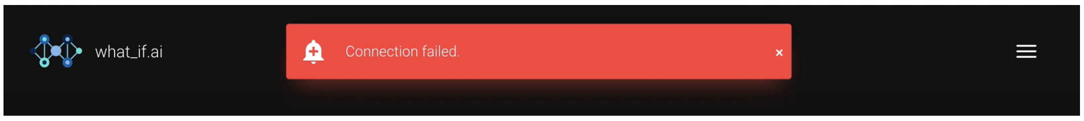


Figure 9: Sample Error Notification

5 Code Base

For convenience, this section provides **direct links** to all files of our gitlab repo used in the sections above.

- Data preprocessing pipeline (Link)
- Model comparison with cross-validation (Link)
- Trend estimator adaption (work in progress) (Link)
- UI development (Link)