TUM Data Innovation Lab

Munich Data Science Institute

Technical University of Munich

&

Amazon Fulfillment Germany GmbH

Project: Uncertainty Quantification and Probabilistic forecasting of big data time series at Amazon Supply Chain

Authors	B.Sc. Jan Marco Ruiz de Vargas Staudacher, B.Sc. Binlan Wu, B.Sc. Conrad Struß, B.Sc. Niclas Mettenleiter,
Mentor(s)	M.Sc. Filippo Lentoni, M.Sc. Aurelien Ouattara, M.Sc. Julien Virlogeux, M.Sc. Giacomo Calzoni, M.Sc. Sami Jullien, Amazon Luxembourg
Co-Mentor	M.Sc. Cristina Cipriani
Project Lead	Dr. Ricardo Acevedo Cabra (Munich Data Science Institute)
Supervisor	Prof. Dr. Massimo Fornasier (Munich Data Science Institute)

Abstract

Time series forecasting in Supply Chain has several applications such as predicting future inbound and flows of goods or the future demand of products.

Since business decisions (e.g. inventory replenishment strategy) may be based on those forecasts it is critical to ensure accurate, robust and interpretable predictions. Additionally, supply chain disruptions such as the Covid-19 pandemic, Suez Canal blocking, Shanghai lockdown or the current geopolitical/economical context have resulted in a higher uncertainty on inbound and outbound flows.

While single-point estimates only provide limited information on our forecasted signal (our conditional expectation), probabilistic forecasts model a representation of the conditional predictive distribution of our target and therefore provide a measure of uncertainty and variability of our variable of interest [1].

During the TUM Data Innovation Lab, we focused on researching probabilistic forecasting methods. By combining the state-of-the-art fixed quantile forecaster MQTransformer with an existing generative copula-based approach we create a new model, the **MQCopulaTransformer**, that achieves competitive results compared to the MQTransformer. In addition, our model prevents issues like quantile crossing and adds the capability of learning cross-series and cross-time correlations.

In addition to modeling the uncertainty, we also researched **Physics Informed Machine Learning** as a way to leverage prior knowledge about the system and variables we are trying to predict. This way we enhance a data-driven approach given laws of physics or economic properties specific to the supply chain environment.

Finally, we worked on increasing the interpretability of how these models estimate uncertainty by researching **Explainable AI** techniques and embedding them on top of our deep learning models pipeline. These techniques allow a researcher to understand the inputs (previous timestamp or exogenous variable) driving the models' predictions and assess how the models learn from data and whether their predictions are reliable.

Our research and package implementation work serves as a basis for further Amazon internal research on uncertainty quantification within the supply chain.

Contents

1 Introduction

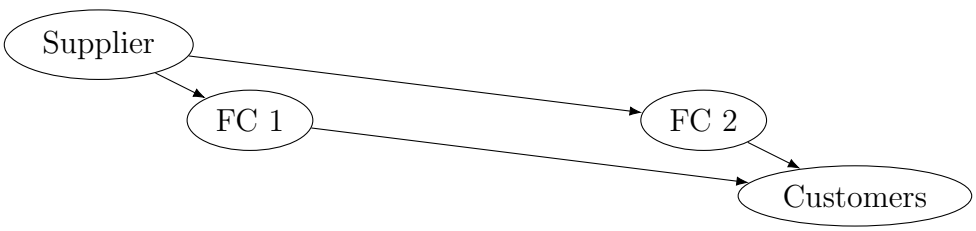
Every day, millions of packages travel through the Amazon Network, most of them using the "Fulfillment by Amazon (FBA)" service, which allows individual, third-party sellers to leverage Amazon infrastructure and supply chain. This poses a huge logistic challenge, as shipments created and shipped from these sellers are unpredictable for Amazon. At the same time, Amazon needs to plan capacities and inventory levels in order not to overload warehouses (called Fulfillment Centers or FCs) or underuse them.

Therefore, quantifying this uncertainty is a critical business issue to improve service performance, drive down costs and allocate resources effectively. This leads to the task of our TUM Data Innovation Lab Project: "Uncertainty Quantification and Probabilistic Forecasting".

1.1 Inbound Volume Prediction Problem for Amazon FBA

Figure 1: Problem Setup: Amazon FBA

Our problem setup is visualized in the sketch below.



The business problem for Amazon is to manage the inbound volume at each FC, e.g. how many items will arrive on a given day. Two key questions arise:

- Can the FC handle the arrivals?
- Can it store all items?

This leads to the requirement of forecasting inbound volume on an FC level. FBA sellers are individual agents into whose strategy and plans Amazon only has limited visibility - we do not control when a seller will create their next shipment. Additionally, the shipping from China and Europe introduces additional uncertainty (due to the longer lead time during which transportation can again be subject to different delays and bottlenecks).

1.2 Project Tasks and Achievements

The goal of our project has been to develop probabilistic forecasting and explainable ML models, by researching the inner workings of algorithms and investigating novel literature available, to estimate and reduce uncertainty for Amazon supply chain business applications.

For this, we began with **Deep Quantile Forecasting** in Chapter 2 and developed a new, state-of-the-art time series forecaster (MQCopulaTransformer). We developed a framework to test and benchmark multiple time series forecasters on several datasets. We present improvements on quantile learning to remove quantile crossing, a known problem in traditional quantile forecasters [2], as when this happens, they no longer output a valid representation of the distribution.

Next, we studied **Physics Guided Machine Learning** in Chapter 3, to incorporate known physical knowledge into the joint forecasting of several variables of interest. We propose a framework of general guided machine learning, usable to guide a data-driven machine learning model into a direction defined by a functional relationship, such as an economic property. Augmenting a data-driven approach with prior physical or business knowledge makes it less reliant on historical data only. This aims at improving the model's robustness and reducing uncertainty in face of distributional shifts, which are difficult to estimate from past data.

Lastly, to reduce uncertainty for Amazon business stakeholders and increase *trustability* and adoption of a given model, we researched **Explainable Artificial Intelligence** techniques in Chapter 4, focusing on LIME [3] (locally interpretable model agnostic implementations). This helps in explaining model predictions to non-technical business stakeholders as well as controlling whether the model's prediction makes intuitive sense and one can trust its output. We developed a pipeline to adapt LIME to multivariate time series forecasting, a setting in which it was not yet used.

2 Neural Models for Probabilistic Time Series Forecasting

2.1 Introduction and Related Work

Forecasting

In forecasting we want to predict the point estimate of our target variable in the future given the past information. Classical approaches like ARIMA [4] fail to deal with the complexity of real world time series datasets as those often include multiple covariates dynamically changing through time, correlated time series, long term dependencies and nonlinear relations between inputs and outputs [1]. Deep neural networks have been shown to perform well being competitive or beating classical approaches like ARIMA or Exponential Smoothing [5, 6] on current benchmarks [7, 8].

Autoregressive vs Multi-Step-Ahead

There exist two major types of approaches: autoregressive and multi-step-ahead forecasters. In the autoregressive setting we only predict one point ahead and then for the next prediction feed this point into our forecaster to obtain the next value.[1]
Let y denote the the value of the time series, t denote the forecasting creation time (FCT) from which we want to forecast the future value of the target, let C denote the number of past values we consider for our prediction and let f denote our forecaster. Then in the autoregressive setting we predict the future value of our target y_{t+1} as $y_{t+1} = f(y_{t-C:t})$ and $y_{t+2} = f(y_{t+1-C:t+1})$.
Chevillon [9] showed that directly predicting multiple steps in time ahead is less biased and more stable compared to the autoregressive strategy. Let K denote the number of future time horizons to predict then we retrieve our target prediction $y_{t:t+K}$ as $y_{t:t+K} = f(y_{t-C:t})$.

Probabilistic Forecasting

Probabilistic forecasting can be thought of an extension of forecasting where we are interested in the full probability distribution of our target variable instead of only a point prediction. The first approach assumes that the data generating process follows a parametric distribution, where the parameters are functions of the inputs. However, it is sometimes hard to fit on real data, as some datasets exhibit fat tail behaviors. [1]
Another non-parametric approach uses Quantile Loss (QL) to learn a fixed set of quantiles of the distribution for each forecasting point in time [1]. Since having access to a fixed set of quantiles does not lead to the marginal quantile function, Gouttes et al. [10] introduced the concept of Implicit Quantile Networks (IQN) which makes it possible to learn the marginal quantile functions using QL. Wen and Torkkola [11] added on top of that a Gaussian Copula to learn the joint distribution between the marginal quantile functions. We will discuss QL, IQN and the copula in greater detail in later sections (2.2.1, 2.2.2 and 2.2.3 respectively).

Related Work

Non-parametric Multi-Horizon probabilistic forecasters have been used in several previous

works. Wen et al. [1] introduced MQR(C)NN an encoder decoder neural network that uses a RNN or CNN to encode the timeseries data into a hidden state representation and a Multi-Layer-Perceptron (MLP) decoder that predicts the quantiles for each forecasting point in time. In a later work Wen and Torkkola [11] extend the MQR(C)NN with IQN and a Copula to estimate the full joint distribution over all the forecasting points in time. Lim et al. [12] introduced Temporal Fusion Transformer (TFT) as another multi horizon forecaster that uses a different encoder/decoder structure with a combination of LSTMs and Attention layers that outperforms MQR(C)NN. Eisenach et al. [13] extended the decoder of MQR(C)NN with two different purposed attention layers in the work to be competetive with TFT on some benchmarks and outperform it on others.

2.2 Theory

We focused our work on the non-parametric approaches where we learned our models on three distribution-learning methods.

2.2.1 Quantile Loss

For a random variable X, let $F_X(x)$ be its cumulative distribution function. The α^{th} quantile of X is defined as

$$q_X(\alpha) \coloneqq \inf\{x \in \mathbb{R} \colon F(x) \geq \alpha\},$$

where in the case of F being invertible the relation

$$q_X(\alpha) = F_X^{-1}(\alpha)$$

holds [14]. Quantile loss was introduced in quantile regression, in which the question of how to produce the desired quantile regression lines (or hyperplanes) was first addressed by [15] which takes the form

$$QL(y, \hat{y}, q) = q(y - \hat{y})_+ + (1 - q)(\hat{y} - y)_+, \tag{1}$$

where y denotes the true target and $\hat{y}$ denotes the predicted target. In our setting, let $k \in \{1, 2, ...K\}$ be the forecasting horizons and Q be the fixed set of sorted quantile indices (e.g. [0.1, 0.5, 0.9]). We are interested in the following loss that takes of the form

$$L(y, \hat{\mathbf{y}}, Q) = \sum_{k=1}^{K} \sum_{q \in Q} q(y - \hat{y}(q, k))_+ + (1 - q)(\hat{y}(q, k) - y)_+, \tag{2}$$

where y is the true target and $\hat{y}(q, k)$ is the q^{th} quantile of prediction at forecasting horizon k.

While the quantile loss allows us to effectively quantify uncertainty, fit underlying distributions and also provide full probabilistic predictions through joint learning over multiple quantile levels, it can happen that two or more predicted quantile functions may cross or overlap, especially when several quantiles are jointly learned. This is problematic since it

violates the monotonic consistency of the conditional quantile function.
For two quantile indices $q_i, q_j \in Q$ with $i < j$ we say two quantiles cross if $\hat{y}(q_i) > \hat{y}(q_j)$. Thus, we came up with the idea to sum over the differences $(\hat{y}(q_i) - \hat{y}(q_j))_+$ for $\forall q_i, q_j \in Q$, s.t. $i < j$ and weight each difference by $(j - i)$. This weighting penalizes too large quantile predictions in lower quantiles indices that cross several larger quantile indices very hard.
This leads to the Quantile Cross Loss (QCL):

$$QCL(y, \hat{\mathbf{y}}, Q, \lambda) = L(y, \hat{\mathbf{y}}, Q) + \lambda J_Q(\hat{y}) \tag{3}$$

where λ is the penalty strength and $J_Q(\hat{y})$ is a weighted penalty term:

$$J_Q(\hat{y}) = \sum_{k=1}^{K} \sum_{\substack{i<j \\ q_i, q_j \in Q}} (j - i)(\hat{y}(q_i, k) - \hat{y}(q_j, k))_+. \tag{4}$$

2.2.2 Implicit Quantile Learning

In MQR(C)NN, TFT and MQTransformers the output of the model is a fixed set of quantiles. This means that we only have access to this fixed set and if we want to query arbitrary quantiles or want to have access to the full quantile function we need to inter-/ extrapolate the quantile predictions [11].
The main idea of implicit quantile learning is instead of feeding our model the same fixed set of quantile indices each batch (e.g. [0.1, 0.5, 0.9]), we sample a set of Q^* different quantile indices from the uniform distribution for each batch, let the model predict the corresponding quantiles $\hat{\mathbf{y}}$ and minimize the quantile loss $QL(y, \hat{\mathbf{y}}, Q^*)$ across all sampled quantile indices and forecasting horizons. [10]
This allows us to query arbitrary quantile queries from the model. As discussed later Section 2.4, we could see that using implicit quantile learning decreased the crossing percentage for all models and for some yielded no crossing at all. However since we cannot guarantee the absence of quantile crossing we still have no valid quantile function.

2.2.3 Learning the Joint Distribution

When predicting K steps ahead and using a fixed set of quantiles or IQN to make predictions we treat the steps $k = 1, ..., K$ ahead as independent from each other. However, for real world datasets this independence assumption is unlikely to hold, hence we want to forecast a representation of the conditional joint distribution through time.
Wen and Torkkola [11] proposed to use a Gaussian Copula to learn the joint distribution across the K steps ahead. A Copula is the joint distribution function of a set of marginally distributed standard uniform random variables [11]. Sklars Theorem [16] states that we can decompose every N-variate distribution function into their marginal distribution functions and a unique Copula $C(\mathbf{u}), \mathbf{u} = u_1, ..., u_N = F_1(y_1), ..., F_N(y_N)$

$$F(\mathbf{y}) = \prod_{i}^{N} F_i(y_i) C(\mathbf{u}) \tag{5}$$

A gaussian copula assumes that $\Phi^{-1}(\mathbf{u}) \sim \mathcal{N}(0, R)$, with Φ the standard normal cdf and R a K-by-K covariance matrix. ($R \in \mathbb{R}^{K \times K}$)
The fundamental idea of their approach is to first learn for each step k the marginal quantile function QF_k parameterized as a IQN, followed by learning the inverse of it which is the marginal distribution function F_k by minimizing the reconstruction loss. After that we retrieve for each k $u_k = F_k(y_{t+k})$, yielding $\mathbf{u} = (u_1, ..., u_K)$ and $\mathbf{z} = \Phi^{-1}(\mathbf{u})$. Learning the copula reduces to learning a model $L(h^*)$ that outputs conditional covariance matrix R for some input h^*. h^* is the output of a decoder network that learns the relationship between time series input $x_{t-C:t}$ and known future information $x_{t+1:t+K}$. Due to computational stability the model outputs the Choelsky lower triangular matrix L such that $R = LL^T$. This can be learned by minimizing the negative log likelihood loss NLL

$$NLL(L, \mathbf{z}) = 2\log(|L|) + \|L^{-1}\mathbf{z}\|_2^2 + \text{const} \tag{6}$$

To sample from the joint distribution for some sample x we draw $\hat{z} \sim \mathcal{N}(0, I_{K \times K})$, then obtain $\hat{\mathbf{u}}^* = \Phi(L(h^*)\hat{z})$ and then get our sample time series as $\hat{y}_{t+k} = QF_k(\hat{u}_k^*)$.
We obtain our quantile predictions by sampling n time series $\hat{y}^{(i)}_{t+1:t+K}, i = 1, 2, ..., n$ from the joint distribution and computing the empirical quantiles on those samples. [11]

Additionally, Wen and Torkkola [11] show that one can retrieve the implied independent latent variables of a given observational series as $\tilde{z} = L^{-1}z_j$ for series $j = 1, ..., M$. Those can be stacked together into an $M \times K$ matrix from which the cross-series $\hat{S}_{M \times M}$ and cross-time $\hat{T}_{K \times K}$ correlation matrices can be estimated which allow insights into the underlying correlation patterns.

2.3 Model Structure

MQTransformer has shown to be equivalent or to outperform state-of-the art approaches [13]. However, it does neither output the marginal quantile functions nor joint predictive distribution. Thus, we combine concepts from MQTransformer together with concepts from IQN and the copula from Wen and Torkkola [11] to combine the strong predictive power of MQTransformer with the capability to output the predictive joint distribution through time.
We implemented our model, MQCopulaTransformer(MQCT), from scratch in pytorch [17] as it is easy to use and extend and currently has the largest share in the scientific community [18]. While there already exist implementations of MQCNN in gluon [19], a library using the mxnet deep learning framework [20], it has a smaller community and is more complex to use.
We structure our model into three different parts: the encoder, the decoder and the loss. The encoder processes the time series data $x_{t-C:t}$ and outputs a latent representation h_t which then serves as input into the decoder. The decoder receives h_t together with the known future information $x_{t:t+K}$(for example the information of a public holiday or the weekday) and then computes another dense representation h_t^*. h_t^* is then fed into the copula loss part which handles the computation of the quantile function, the inverse of it (distribution function) and the copula. We discuss these three component in greater detail below. In addition, with advent of recent development of transformer models in time series

forecasting, we build our models based on the transformer extension with different types of attention layers in the decoder, which then also take the latent representation h_t from the encoder to produce the prediction.

Encoder

For the encoder we implemented and compared several different variants including standard Recurrent Neural Networks (RNN), Long-Short-Term-Memory Network (LSTM), Gated Recurrent Unit (GRU), a Wavenet [21] like Convolutional Neural Network (CNN) and structurally simpler version of Wavenet. The simpler version omits the PixelCNN [22] like gated activation unit and consists of a stack of dilated causal convolutional layers which were introduced in this context by Chen et al. [23]. The simpler CNN encoder we implemented follows the architecture of the CNN encoder implementation in gluon [19]. We found that the simpler CNN performed best which is consistent with the literature [12, 1].

Decoder

Traditional MQCNN uses multi-layer perceptron(MLP) in the decoder to produce quantiles of predictions and by design of this sequence to sequence structure, we face the 'information bottleneck' problem where the decoder receives information from the encoder via a single hidden state. In recent years, with the advent of transformer models in time series forecasting, this problem was solved by Eisenach et al. [24], where they used two different types of attention layers in the decoder to produce predictions. In our model, we took the inspiration from the decoder of MQTransformer and further combined it with the copula.

Copula

While in 2.2.3 we discussed the theoretical foundations of the copula learning in the following we discuss the concrete decisions we took for implementation. The copula of our model consists of three different parts. For the first one, the IQN, we took inspiration from the implementation of the pytorch forecasting library [25], removed the cosine embedding and changed the sampling strategy such that each element of the batch receives different quantile samples instead of the same ones as originally proposed. We noticed that this improved the training and made the inversion of the function easier.
We implemented the other two parts from scratch as there is no public implementation available, while following the specification in the paper of Wen and Torkkola [11].
The first part is the IQN. For each batch we sample a standard uniform tensor u of quantile indices. Then we use a MLP $QF(h_t^*, u)$ that outputs the quantile predictions $\hat{y}$ and is then optimized by minimizing the quantile loss.
The second part is the inverse of the QF. We train it in an autoencoder fashion: first we freeze QF, sample quantile indices u, then we use QF to compute predictions $\hat{y}$. After that we train the inverse MLP by passing the quantile predictions together with h_t^*, to obtain the predicted quantile indices $\hat{u}$ as $\hat{u} = F(h_t^*, \hat{y})$. We learn the weights of F by minimizing the reconstruction loss $\|u - \hat{u}\|_2^2$.

The last part is the copula. We first freeze the weights of F and QF use a MLP $L(h_t^*)$ that outputs the cholesky lower triangular matrix such that $R = LL^T$ is the covariance matrix of the gaussian copula. We use the tricks from Wen and Torkkola [11] to increase the numerical stability and ensure that R is a proper covariance matrix. We learn the weights by minimizing the negative log likelihood as specified in equation 6. Note that the input to the copula x in section 2.2.3 corresponds to the output of the decoder h_t^* in this section.

2.4 Implementation and Benchmarks

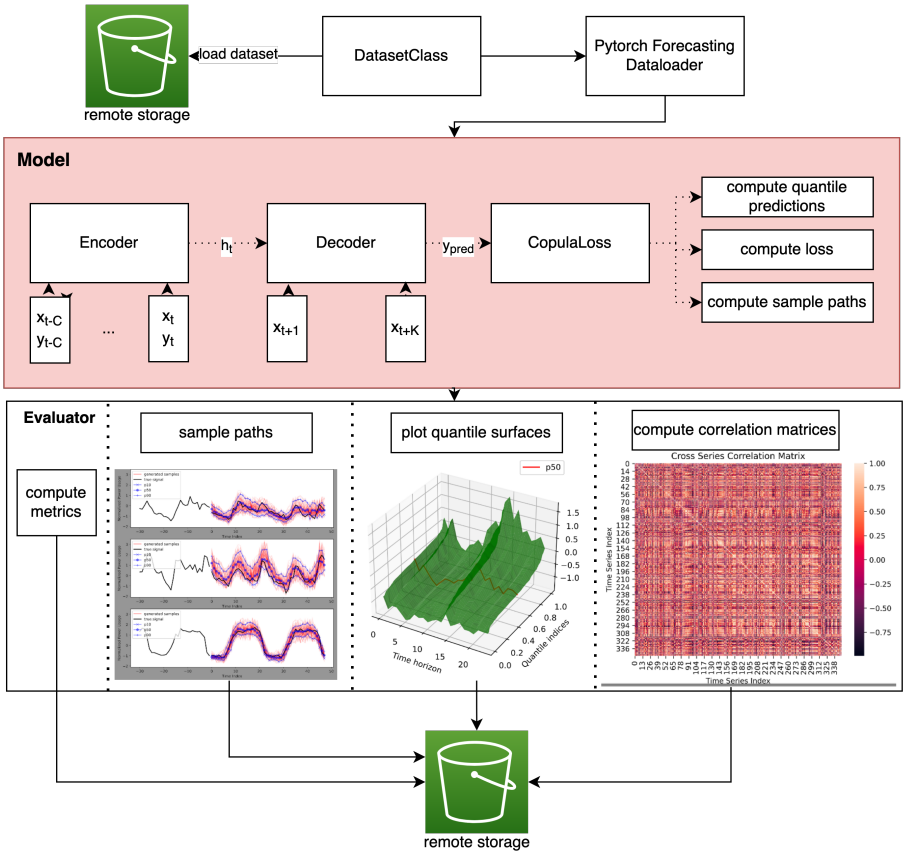


Figure 2: Experiment Running Overview. The figure shows the components that are involved during the running of our experiments. One experiment consists of a dataset, a model and an evaluator. The interactions between those components are orchestrated by the Experiment class.
Each dataset is implemented as a class that handles the dataset loading from Amazon Web Services (AWS) remote storage, the preprocessing and returns a data loader. The trainer class from pytorch lightning [26] then consumes this dataloader and the initialized model class and trains the model. After training is finished the Evaluator class saves the metrics and plots to a remote storage.

2.4.1 Implementation

We implemented a library that supports several models and makes running and evaluation of experiments easy.
While we only introduce our final model in detail, we implemented several ones that share the same basic structure: all of them consist of an encoder, decoder and a loss all implemented as python classes.
For the encoder we implemented RNN, LSTM, GRU, the Wavenet and simplified Wavenet CNN encoders. For the decoder we implemented a seperated MQR(C)NN style decoder, a simplified version only using a MLP as well as a MQTransformer style decoder and a custom Transformer encoder.
We implemented several different losses: standard QuantileLoss, QuantileCrossLoss, ExpectileLoss, ImplicitQuantileLoss and Copula Loss.
Due to the way we designed the library we can combine different encoders, decoders and losses easily. Those components are connected through their input and output dimensions.
To retrieve those dimensions we implemented class functions such that those can be set automatically without having to hard code changes in the code. This modular structure enables quick experimentation and extension.
Figure 2 shows a visualization that explains the experiment running flow.

2.4.2 Benchmarks

Dataset
We ran experiments on public datasets and hand-crafted synthetic datasets to reflect our model on different challenges. For completeness, we first evaluate our model on the electricity dataset[27] which focuses on simpler univariate time series forecasting, then we consider the hand-crafted synthetic datasets implemented on our own to see how our models handle changing signals. Last, we test our data on the Stallion dataset [28] which has much smaller size and is expected to be a harder task.

Metrics
One evaluation metric we already discussed is the quantile loss as defined in 2. We report the quantile loss on the set of quantile indices [0.01,0.02,...,0.99]. Another metric we use is the quantile crossing percentage

$$\text{CrossingPercentage}(\hat{y}, Q) = \frac{1}{N \times K \times |Q|} \sum_{n=1}^{N} \sum_{k=1}^{K} \sum_{i=1}^{|Q|} 1_{(\hat{y}(q_i - 1, k) > \hat{y}(q_i, k)} \tag{7}$$

with FCT t, N time series samples, K forecasting horizons, and a set of quantiles Q. The last set of key metrics are the Calibration and ExpectedCalibrationError. Calibration measures how well our quantile predictions are calibrated. To give an example, when we want to predict the median (p50 quantile) then 50 percent of the data should be smaller or equal than the predicted value. Calibration measures for each quantile prediction $\hat{y}(q)$ how much percent of the data are smaller or equal to it. The CalibrationError takes the absolute value of the difference between this percentage and the corresponding quantile

index. The ExpectedCalibrationError takes the mean of the CalibrationError over all quantile indices q.

$$\text{Calibration}(y, \hat{y}, q) = \frac{1}{N \times K} \sum_{n=1}^{N} \sum_{k=1}^{K} 1_{(y_{n,k} <= \hat{y}_n(q,k))} \tag{8}$$

$$\text{ExpectedCalibrationError}(y, \hat{y}, Q) = \frac{1}{|Q|} \sum_{q \in Q} |\text{Calibration}(y, \hat{y}, q) - q| \tag{9}$$

Experiment Results Below we show results on the electricity dataset on the metrics introduced above.

Electricity Results
In accordance with [7], we use the past week (168 hours) to predict the power usage of 370 users over the next 24 hours.

Figure 3: The left column shows the comparison across the following models: MQCNN Mxnet v.s MQCNN pytorch v.s MQTransfromer learned on quantile loss then evaluated on mean quantile loss, expected calibration error and quantile crossing loss, and on the right is comparison across MQTransformer learned on different loss: QL v.s QCL v.s IQL v.s Copula then evaluated on mean quantile loss, expected calibration error and quantile crossing loss.

As from the plots, with only the QL, the quantile crossing percentage is rather high at 18%. QCL reduces effectively to 11% while IQN significantly cut it down to 3%. But both fails to reach the crossing percentage of zero. MQCopulaTransformer achieved a crossing percentage of 0 as expected since we are using empirical quantiles on generated sample paths. IQN has the best mean quantile loss and expected calibration error among all models. However, it is only slightly better than MQCopulaTransfromer with respect to the loss while not guaranteeing zero quantile crossing.

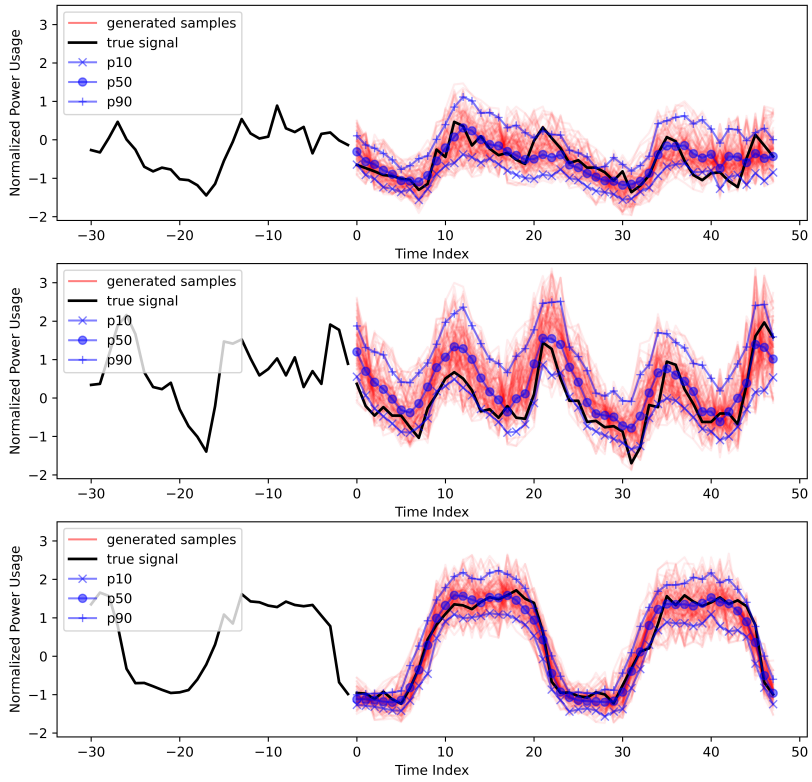


Figure 4: Three sample predictions of MQCopulaTransformer on the electricity dataset selected to show different peak behaviour. The black line shows the true signal, red lines show generated sample paths and the blue lines show the different quantile predictions of the p10, p50 and p90 quantiles

From the sample predictions in Figure 4 we see that even though our model fails to capture all of the strong signals we observe that it barely missed to capture the tendency of the target variable. And although the quantiles of predictions seems to be very close to each other, quantile crossing percentage is 0 as by design. Figure 5 shows the correlation matrices for the dataset.

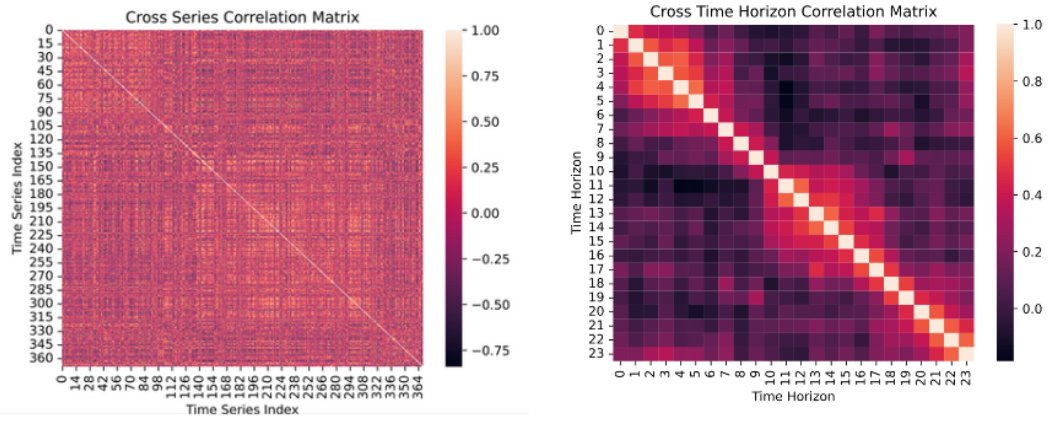


Figure 5: Sample Correlation Matrices created according to Wen and Torkkola [11] as discussed in section 2.2.3 on the electricity dataset. Bright values indicate high positive correlations while dark values indicate high negative correlation. The image on the left hand side shows the correlation between different time series while the one on the right hand side shows correlation between different time horizons

We achieved very good results on the MQCopulaTransformer which enables 0 crossing quantiles.

3 Physics Informed and Guided Machine Learning

In the context of global supply chain disruptions in recent years (Covid-19, Suez Canal blockage, Shanghai lockdown, War in Ukraine) increased attention has been on potential disruptions in the distribution of supply chain forecasts. These shifts are inherently unpredictable from historical data only, on which state of the art time series forecasting models, including our MQCopulaTransformer, rely exclusively. Therefore ways to make model training more robust and less reliant on data only are needed.

Traditionally a technique for improved generalization is regularization. It is a way to bias the models weights toward zero and has been observed to improve model generalization [29].

This lead us to investigate physics informed machine learning. This is a technique to regularize models based on a physical law that is believed to govern the variables we try to predict. Since physical laws don't change over time, our idea is to improve model robustness and reduce uncertainty with this approach by letting the model use both data and prior knowledge, the latter formulated in the regularization term. Traditionally the 'prior' knowledge formulated in regularization that biases weights to zero is that weights shouldn't be too high or volatile, but with a specific physical formula, we aim to provide better initial bias.

We furthermore expanded on this physics informed machine learning idea by using any formula describing an economic model or business insight as regularization. This introduces guided machine learning. Here we guide (or bias) the model into a certain direction by adding a regularization term to the loss function, which is 0 when our guidance function is zero. Any economic rationale, equation, any prior knowledge from business intelligence that can be expressed as a mathematical formula can be incorporated into model training in this way.

Of course economic and business regularization is not as hard a constraint as physical laws (which could be considered by a smaller penalty weight). The guidance could also turn out to be wrong and worsen performance. Given a good guidance function however the model is not only guided by data, but also by prior knowledge from a researcher or businessperson. They bring it to the table through their extensive knowledge of the given learning task.

3.1 Introduction and Related Work

One of the biggest advantages of many machine learning models is their ability perform well in tasks, where it is difficult or impossible for humans to formulate the underlying rules or principles of the task, or even give explicit instructions to solve them. One example for this is image recognition, where it is very challenging to define manual algorithms that lead to good results, but relatively easy to train a Neural Network to do it. This "black box" approach however, requires a lot of data to succeed. A lack of (good) data, or data whose distribution shifts quickly, is often a big problem and can be addressed

by incorporating prior knowledge about the problem. One such way is incorporation of physical rules that govern the system which we want to analyze.

We can easily construct an example where more knowledge of the physics lead to less need for data (see Figure 6). Suppose we want to estimate the trajectory of an object flying through $\mathbb{R}^2$ and that its starting position is $(0, 0)$. It is well known, that the trajectory of such an object is described by the equation

$$y = \tan(\alpha) \cdot x - \frac{g}{2v_0 \cos^2(\alpha)} \cdot x^2,$$

where $g \in \mathbb{R}$ is a constant known to us, but the launch angle α and the starting velocity v_0 are unknown. But since we miss only 2 parameters, in order to know the whole parabola,

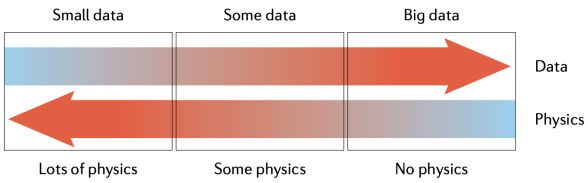


Figure 6: Data vs Physics [30]

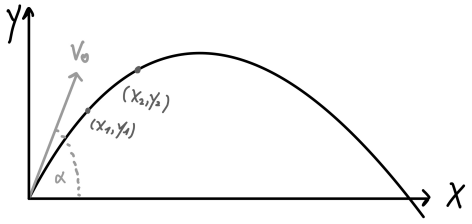


Figure 7: Trajectory of an object

we need to make only two distinct measurements $(x_1, y_1) \neq (x_2, y_2)$ of the trajectory. These two measurements would give us a system of two equations, which we could solve for v_0 and α, and thus obtaining all the points on the trajectory. Of course in this case machine learning is not needed at all, so the question arises what to do if we don't know all of the governing physics of a problem, but only some part of it or we only have a simplified model of reality.

If the laws of a physical system are well known and stable, one way of incorporating knowledge of the physical system is to directly design the ML model in a way that it better represents the physical laws. Examples for this are the very successful Convolutional Neural Networks (CNN) in image recognition, or Recurrent Neural Neural Networks (RNN) for sequential data [30], [31], [32].

Another approach, and the one will focus on in this text, is not to incorporate the physics in the design of the ML model, but instead in its training. In practice, both methods can and should be combined. We want to model physical flows for supply chains. A system of ODEs describing the flow of the supply chain network and a machine learning model, trained on both the data and the physical rules, should give better predictions and predictions more consistent with physical reality. In practice, changing the loss function is also much less work intensive than developing a new network architecture.

Our problem setup for physics informed machine learning is similar to the work of Maziar Raissi and Karniadakis [33]. We begin with a system of coupled ordinary differential equations dependent on time t. Let $y\colon \mathbb{R} \to \mathbb{R}^n$ be a vector valued n times differentiable function dependent on the time. Let $y, y^1, \ldots, y^n$ be its derivatives with

respect to the time, i.e. $y^i\colon \mathbb{R} \to \mathbb{R}^n$, $t \mapsto (\frac{d^i y_1}{dt^i}(t), \dots, (\frac{d^i y_n}{dt^i}(t))$ and $F\colon \mathbb{R} \times \mathbb{R}^{n^2} \mapsto \mathbb{R}^n$. Then the system of ordinary differential equation is satisfied, if

$$F(t, y, y^1, \dots, y^n) = 0. \tag{10}$$

Assume now in a first step that we know F and want to obtain the solution y of the ODE system. Since we know that neural networks can approximate any continuous function arbitrarily well [34] , we model y as a neural network

$$y = f_\theta\colon \mathbb{R} \to \mathbb{R}^n,$$

where $\theta \in \mathbb{R}^m$ are the weights, with $m \in \mathbb{N}$ being the number of parameters of the network. Since the structure of a neural network is well known the derivatives $f_\theta, f_\theta^1, \dots, f_\theta^n$ can be calculated through automatic differentiation present in modern day machine learning libraries such as pytorch.

Let now $\{(t_1, y_1), \dots, (t_N, y_N)\}$be a set of training data, with $(t_i, y_i) \in \mathbb{R} \times \mathbb{R}^n$. Ideally, for any $i \in \{1, \dots, N\}$ we wish to have

$$f_\theta(t_i) = y_i,$$

but also

$$F(t_i, f_\theta(t_i), f_\theta^1(t_i), \dots, f_\theta^n(t_i)) = 0.$$

The idea of physical informed neural networks is to solve the optimization problem

$$\arg\min_{\theta \in \mathbb{R}^m} \frac{1}{N} \sum_{i=1}^{N} |f_\theta(t_i) - y_i|^2 + \frac{1}{N} \sum_{i=1}^{N} |F(t_i, f_\theta(t_i), f_\theta^1(t_i), \dots, f_\theta^n(t_i))|^2. \tag{11}$$

Variations of this, such as scaling parameters for the two MSE's, or different loss functions than the MSE, are also possible.

In a second step, we now assume that we don't know F exactly, but instead we only know that it belongs to a family of functions $\{F_\lambda\colon \mathbb{R} \times \mathbb{R}^{n^2} \mapsto \mathbb{R}^n \mid \lambda \in \mathbb{R}^l\}$. This of course leads also to a family of ODE systems

$$F_\lambda(t, y, y^1, \dots, y^n) = 0.$$

But if our assumption is right, and the dynamics of our real life problem follow one of these ODE systems, then our training data $\{(t_1, y_1), \dots, (t_N, y_N)\}$ should approximately follow the solution y of this system for some specific parameter vector $\hat{\lambda}$. To try and find both $\hat{\lambda}$ and y, we now need to solve the optimization problem

$$\arg\min_{\theta \in \mathbb{R}^m, \lambda \in \mathbb{R}^l} \frac{1}{N} \sum_{i=1}^{N} |f_\theta(t_i) - y_i|^2 + \frac{1}{N} \sum_{i=1}^{N} |F_\lambda(t_i, f_\theta(t_i), f_\theta^1(t_i), \dots, f_\theta^n(t_i))|^2. \tag{12}$$

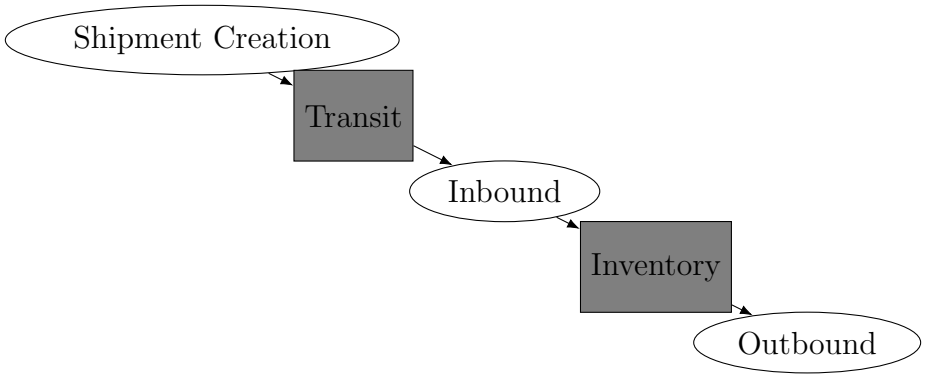
Assuming that F_λ is differentiable with respect to λ, we can again use automatic differentiation to solve (12).

It is important to note, that the idea of adding a second constraining loss into the training, is by no means limited to differential equations. Any law or principle, that the output of the neural network must obey, can be used. Although they didn't use feed forward networks, a good example of this is the use of conservation of energy in [35].

3.2 Applications in Supply Chain Forecasting

A supply chain is a system which can described as physical flow. Therefore, we can impose constraints in the form of an ordinary differential equation (ODE) to enforce that models which are predicting different parameters of interest independently from each other, together produce physically consistent forecasts.

We consider a simplified supply chain with 5 variables, dependent on time t: Shipment Creations by sellers $S(t)$, Shipments in Transit $Tr(t)$, Arrival Inbound at the FC $Ib(t)$, Inventory of the FC $Inv(t)$ and Outbound volume $O(t)$.



The Models learning the targets are parametrized by θ:

$$\begin{aligned} t &\mapsto (S(t), Tr(t), Ib(t), Inv(t), O(t)) \\ t &\mapsto (S_\theta(t), Tr_\theta(t), Ib_\theta(t), Inv_\theta(t), O_\theta(t)) \end{aligned} \tag{13}$$

Notice that we have two different types of nodes, whom we call buckets and gates. A bucket is a node which can fill up or empty, and whose derivative is defined by its gate nodes via the conservation of mass law, which states that the nodes must adhere to the ODEs:

$$\begin{aligned} \dot{Tr}(t) &= S(t) - Ib(t) \\ \dot{Inv}(t) &= Ib(t) - O(t) \end{aligned} \tag{14}$$

The gate nodes on the other hand measure strength of flow into and out of nodes.

From a business perspective this modeling is interesting as it not only enforces physically more realistic results (e.g. Inventory can't increase a lot more than Inbound - Outbound would allow), but also allows for simulations.

Consider the event of an economic shock happening and $S(t)$ decreasing significantly. Through the joint training, this decrease will propagate fast through the other networks and decrease the nodes downstream, adjusting all to the reality of the new situation. The same way a constraint imposed by Amazon on shipment creation would propagate quickly through the network. A bottleneck in Transit would reduce the amount of Inbound expected in the near future.

These variables are not only connected by the physics of a supply chain, but also by economic connections and expectations of businesspersons on the respective other variables. Therefore, we present more ways to incorporate expert knowledge into the joint modeling in the next chapter 3.3.

3.3 Guided Machine Learning

The idea of combining domain knowledge with statistical analysis is also present in Causal Diagrams [36]. There statistical variables are arranged in a graph with directed edges, called causal diagrams, and the edges represent causal relationships between variables. The graph can be estimated from data [37], but often expert domain knowledge is used to assist in the creation of the causal diagram. Then the combination of prior domain knowledge and statistical data allows deeper analysis, especially causal analysis in this case.

With a similar idea of combining domain knowledge and data, in Guided Machine Learning we expand on Physics Informed Machine Learning. We draw inspiration from the idea that forecasting can be improved by bridging data driven and physics driven modeling [38]. As seen in Chapter 3.1, a formula believed to govern a modeling system can be incorporated into training a machine learning model by moving all terms of the formula to one side (setting it equal to 0) and adding this as a regularization constraint.

Consider the general supervised learning task $f : \mathcal{X} \to \mathcal{Y}$, the estimation of the functional relationship f being f_θ, parameterized by weights θ. Whenever a functional relationship $\mathcal{X} \times \mathcal{Y}$ holds , $g : \mathcal{X} \times \mathcal{Y} \to \{0\}$, or a differential equation w.r.t. f holds (which we express via $g : ((f^{(1)}, ...f^{(n)}) \times \mathcal{X} \times \mathcal{Y}) \to R^d$, n being the amount of times f is continuously differentiable) and d being the number of equations that hold. We call this g the guidance function.

Then the usual loss function $L(x, y, f_\theta)$ (for example the squared error loss, $L(x, y, f_\theta) = \|y - f_\theta(x)\|_2^2$), which penalizes how well f_θ matches y, can be augmented with $\|g(x, y, f_\theta)\|_2$:

$$L_g(x, y, f_\theta, \lambda) = L(x, y, f_\theta) + \lambda \|g(x, y, f_\theta)\|_2 \tag{15}$$

In addition to matching the data, the model f_θ now is also encouraged to adhere to the equation described by g. Depending on the confidence in g or after validation testing, the parameter λ can be tuned.

Applications
During our theoretical analysis in particular we thought of two ways to improve training via the guided machine learning approach.

The **Economic Ordering Quantity** EOQ is a supply chain model giving a closed form solution to the optimal quantity of items to ship, considering s: shipment cost, h: holding

cost per year, T one year and demand for one year $\int_0^T O(t)$ [39]:

$$C(0) = \sqrt{\frac{2\int_0^T O(t)s}{h}} \tag{16}$$

with $N := \lceil \frac{\int_0^T O(t)}{C(0)} \rceil$ shipments created per year, at times $\frac{k-1}{N}T$, $k \in [N]$.

Furthermore the shipment creation process, which due to the FBA services structure is very unpredictable, is decided by seller behaviour, who in turn are influenced by expectations on how much they can sell, therefore demand pull shipment creation could be a formula to improve model training:

$$C(t) = \alpha O(t) - \beta Tr(t) - \gamma Inv(t) \tag{17}$$

with α, β, γ learnable parameters, which later can be used to assess strenght of the relationship and seller behaviour under different circumstances.

3.4 Conclusion

In this section we introduced how physics informed machine learning, and in general guided machine learning can be applied to supply chain forecasting as well as their theoretical frameworks. If a guided model can learn hyperparameters of the guidance function, e.g. α, β, γ in equation (17), while having good accuracy, these hyperparameters provide (model-specific) explainability into the guidance function. This desirable feature of gaining explainability is explored further in the next Chapter 4 for the model agnostic case.

(Physics) Guided Machine Learning is an open research field for Amazon and we encourage further research as well as empirical experiments to test practical applicability. For our project we decided to focus on other aspects, such as explainability, due to limited time.

4 Explainability to Understand the Uncertainty Estimator

Explainable AI (XAI) is the study of methodologies that help gain insight into complex non-linear ML models [40]. While having insight into the inner workings of a model does not directly reduce uncertainty, it is a useful tool to quantify and understand it. It also can be a stepping stone to reduce future uncertainty. Imagine a scenario where the input data of a model can only be of a certain size, for example due to memory constraints. Once we know which signals are important for a model to make inference on, we can then increase the quality of these important signals. Therefore we improve future forecast quality and reduce uncertainty, while still maintaining the same input data size.

A second important use case of XAI is the accessibility to Data Science it creates to non-experts of the field. Business stakeholders will have increased trust in the models, once they are more interpretable and understandable for a non-technical staff.

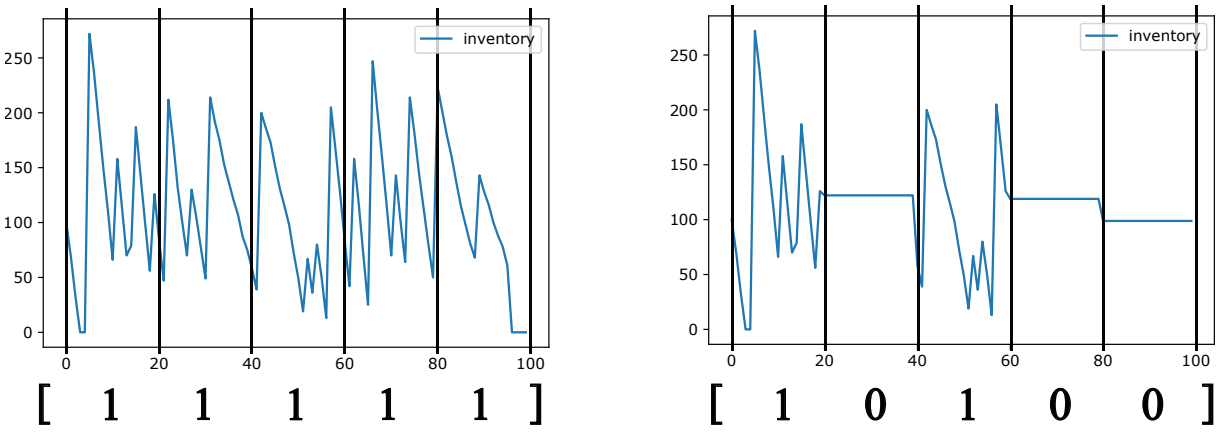
4.1 Introduction and Related Work

During this TUM DI Lab project we looked at two popular approaches in XAI. The first one is the Shapley value (SHAP), which was originally brought up by Lloyd Shapley in the field of game theory [41]. It is a numerical value assigned to each player in a cooperative game with a payout, that measures how much each player contributes to the payout. In the machine learning setting, the "game" is making a prediction, the "players" are the regressors, and the "payout" is the numerical prediction value [42].
The second approach we studied is Locally interpretable model-agnostic explanations (LIME) [3]. The premise of LIME is to transform the feature space of an ML model f to a much simpler surrogate feature space, and then train a linear explainable model g in a neighbourhood of a point in the surrogate feature space, to mimic the behaviour of the original model. The explainability of the linear model can then be used to explain the original model f.
Both LIME and SHAP have the advantage of being model agnostic, meaning that in theory they can be applied to a broad range of ML models under almost no further conditions. Both approaches also have publicly available implementations, namely [43] and [44]. In the end, we concentrated on LIME, as SHAP is very computationally expensive when dealing with a large number of regressors, which would be a problem when dealing with big time series data [42].

So far LIME has found applications in the areas of image classification [44], text classification, time series classification [45] and regression on low dimensional tabular data [44]. The innovation of our work during the TUM DI-LAB regarding explainability lies in applying LIME to time series regression tasks, as well as to multi outpout, multihorizon models. For that we implemented a data processing pipeline that provides an interface between LIME and our model implementations described in Section 2.4.

4.2 Theory

As already mentioned, we start the LIME algorithm with a trained ML model $f\colon X \to \mathbb{R}$, where X is the feature space [3]. A typical feature space can be images, time series, or a text data sace. Next, we pick an instance $x \in X$, where we want to explain the prediction $f(x)$. To obtain the surrogate space X', we divide the instance x into $d \in \mathbb{N}$ segments, like for example superpixels in image data, or window segments in time series data. Then our surrogate space is $X' = \{0,1\}^d$, where each dimension corresponds to one segment in x. Our original instance x corresponds to $x' = [1, 1, \ldots, 1] \in X'$ and through randomly turning entries of x' to 0, we obtain samples $z' \in X'$ in a neighbourhood of x'. Through a pertubation transformation $\varphi\colon X' \to X$, these samples are mapped to X. The only rule φ must obey is that an entry of 1 gets mapped to the corresponding unchanged segment of x, while an entry of 0 gets mapped to an altered version of the corresponding segment.



(a) Binary Representation

(b) Possible Pertubation Transformation

Figure 8: Pertubation Transformation

As shown in Figure 8 on the right, in the case of time series, a pertubation transformation can be to map a 0 to the mean of the corresponding segment of the original instance x [45]. Note that this yields $\varphi(x') = x$

Note that usually φ is only injective, not surjective. Thus, we can only define the inverse as $\varphi^{-1}\colon \varphi(X') \to X'$. In this way we obtained the training data set

$$\mathcal{Z} = \{(\varphi(z^{'}), z^{'}) \mid z^{'} \text{ sampled from } x^{'}\} \subset X \times X^{'}$$

for the local model. If even after the segmentation the number of features d of the surrogate space is too high, it is further reduced by taking only into account $K \in \mathbb{N}$, $K \leq d$, features. We call $K \in \mathbb{N}$ the *length of the explanation*. To be more precise, for an element $z^{'} \in X^{'}$ we denote with $z^{'}_{|K} \in \{0,1\}^K$ taking only K fixed entries of z'. Then the local model is going to be

$$\begin{aligned} g\colon \{0,1\}^K &\to \mathbb{R} \\ z^{'} &\mapsto \omega^{\top} z^{'} + \omega_0, \end{aligned}$$

where we determine the weights $\omega \in \mathbb{R}^K$, $\omega_0 \in \mathbb{R}$ by solving the linear regression problem

$$\underset{\omega \in \mathbb{R}^K, \omega_0 \in \mathbb{R}}{\arg\min} \sum_{(\varphi(z'), z') \in \mathcal{Z}} \pi_x(\varphi(z')) \left(f(\varphi(z')) - g(z'_{|K}) \right)^2 . \tag{18}$$

Here $\pi_x \colon X \to \mathbb{R}$ is some function that measures the inverse distance of any point $z \in X$ to x. In the case of time series we can base π_x on the l^2 norm, i.e. $\pi_x(z) = \exp\left(-l^2(z - x)\right)$. In the standard LIME implementation [44], the problem (18) can be solved with either L1 regularizartion (Lasso) or L2 regularization (Ridge) based on the scikit-learn package [46]. The objective function in (18) has to be changed according to the method used. In

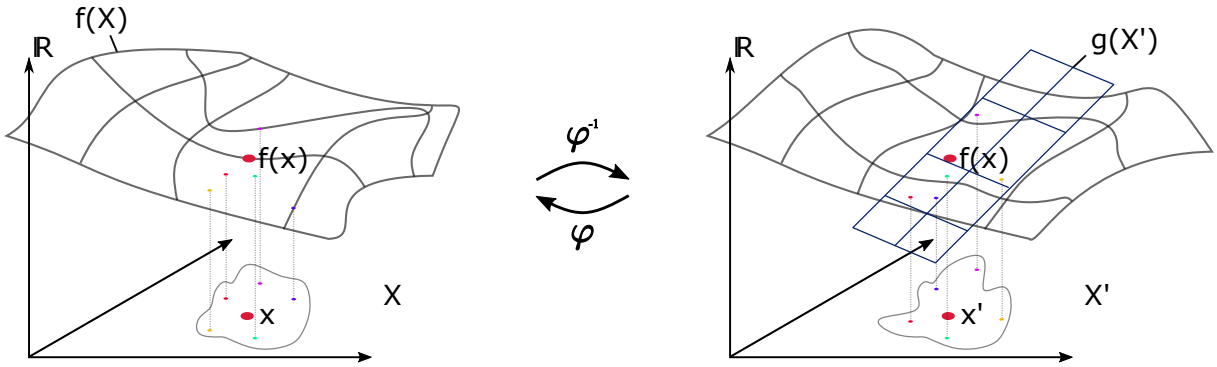


Figure 9: Idea behind LIME

Figure 9 the intuitive explanation of how we interpret the output of LIME, which are the weights of the linear model g, can be observed. The model g is the best approximation to f in a neighbourhood of our sample x' in the surrogate space X'. If now a feature in the surrogate space, which corresponds to a segment of data in the original instance x, has a high weight, that means if we move in the direction of the axis corresponding to this weight, g changes a lot, and therefore also f. This means that feature has a high impact on the predictions f makes in this neighbourhood. If on the other hand a feature has a low weight, that means a change of this feature has a low impact on f. Mind here that $X' = \{0, 1\}^d$, which means the original values of the data points in X, that correspond to the feature play no direct role in the surrogate space.

4.3 Predicting and interpreting an Inventory Simulation

In order to test if the LIME algorithm works as intended also for time series data, we created an inventory simulation that follows simple arithmetic rules. Our simulation models the time dependent development of an inventory of one item. The inventory belongs to a seller, who decides based on the state of all model variables, when and how much he restocks his inventory. Such a restocking we call *shipment creation* and after a certain lead time, the shipments arrive as inbound in the warehouse. The premise of our simulation is that a seller will create a new shipment whenever his inventory falls below a certain dynamical threshold. The quantity of shipments he creates is then calculated based on a formula based on the *economic order quantity* [39].
We then trained a random forest with 1000 estimators to predict the shipment creation

quantity of the next time step. As input features we gave the model the current demand of items, the shipment creation quantity, shipments in transit and inbound, as well as the historic data of inventory position of the past 20 time steps. As instance x to explain we

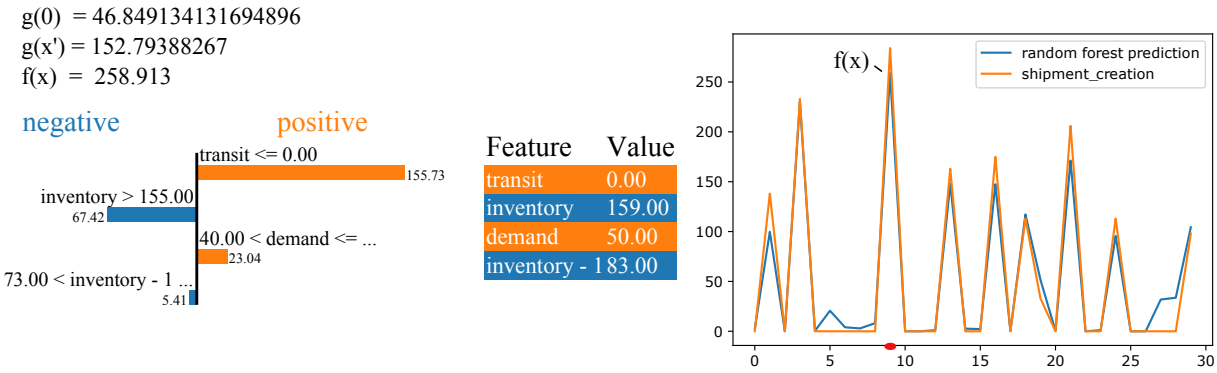


Figure 10: Interpreting a random forest prediction

pick the 9-th time step of the test as, as depicted in Figure 10 on the right. The actual shipments created at this time are 284, and the random forest predicted $f(x) = 258.913$. We then applied the "LIME for Tabular Data" approach from [44] to this prediction. Note that in this implementation of LIME there is no feature aggregation done, as it is intended for low dimensional tabular data. Therefore a LIME weight is assigned to each feature individually, instead of for example to windows in a time series. The results are displayed in Figure 10. As length of explanation we choose $K = 4$. In the table are displayed the feature and the value it attains in our instance x. In the bar plot on the left we see the rounded value of the weights each of the 4 features got. In other words the linear model is

$$\begin{aligned} g\colon \{0,1\}^4 &\to \mathbb{R} \\ z' &\mapsto 155.73 \cdot z_1' - 67.42 \cdot z_2' + 23.04 \cdot z_3' - 5.41 \cdot z_4' + 46.85 \end{aligned}$$

and we have

$$\begin{aligned} g(x') = g([1\ 1\ 1\ 1]) &= 155.73 \cdot 1 - 67.42 \cdot 1 + 23.041 - 5.41 \cdot 1 + 46.85 \\ &= 152.79. \end{aligned}$$

We see that according to LIME the 3 most important features are current transit, inventory and demand. This aligns fully with our simulation as in our algorithm the shipment creation quantity in the next time step, as well as if a shipments is created at all, will depend solely on the quantity of these variables at the next time step. The past inventory position, that we gave our model as additional features, play no role in the simulation, and LIME correctly attributes no importance to them. But not only the absolute values of weights, also their signs align with the simulation. A low number of shipments in transit leads to a higher number of shipments created, and a high inventory position leads to a low number of shipments created. In our instance x there are 0 shipments in transit, and the inventory is 159, which is high. Additionally the demand is high, which also increases the number of shipments created.

4.4 Adapting LIME to Multihorizon Timeseries Forecasting

In order to apply LIME and thus provide explanations for our quantile forecasting models, several questions had to be answered. One task to overcome was, that in the standard LIME algorithm the ML model $f\colon X \to \mathbb{R}$ maps only to the real numbers, but our time series forecasting models all are of the form $f\colon X \to \mathbb{R}^{n_1 \times n_2}$, where n_1 is the forecast horizon and n_2 is the number of quantiles. We solved this by fixing the timestep t and

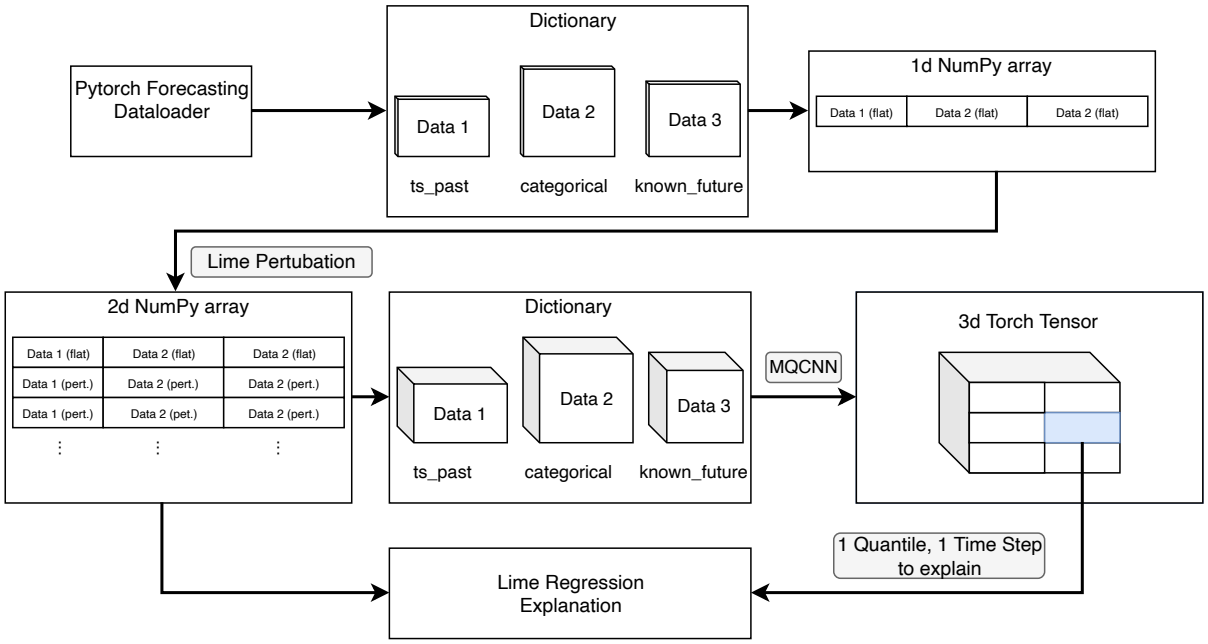


Figure 11: Data processing pipeline for LIME with PyTorch Models

quantile q and analyzing only the t, q-th output of our model at once. More precicely, let

$$\begin{aligned} h\colon \mathbb{R}^{n_1 \times n_2} &\to \mathbb{R}, \\ (a_{i,j})_{\substack{i=1,\cdots,n_1 \\ j=1,\cdots,n_2}} &\mapsto a_{t,q}. \end{aligned}$$

The model we give to LIME is then $\tilde{f} := h \circ f$, which maps into the real numbers, as needed.

As a first proof of concept, we decided to adopt the "Lime for Tabular Data" approach from [44], where each data point in our time series gets assigned a LIME weight and there is no feature aggregation done. Our models get their data in batches from the PyTorch dataloader. The data is stored in Python dictionary form, and we designed the data processing pipeline to do the transformations necessary for LIME, as seen in Figure 11. Our experiments showed, however, that too many feature in the "LIME for Tabular Data" approach lead to poor explainability results. Therefore we deem it necessary that our code is extended to incorporate feature aggregation similar to the window segmentation depicted in Figure 8, since this reduction in dimensionality is one of the big strengths of LIME.

5 Conclusion

During the TUM Data Innovation Lab at Amazon, we developed a modular and extendable framework to experiment and benchmark probabilistic time series forecasting models. We developed a new state-of-the-art model **MQCopulaTransformer**, which is especially suited for business applications since it is generative, provides consistent quantile forecasts and allows to create cross-series and cross-time correlation matrices.

We conducted theoretical research into ways to further reduce uncertainty via **physics or economic guided machine learning** and present a general framework to incorporate prior knowledge into training by only having to adapt the loss function.

In the same spirit, we adapted the **XAI technique LIME** to the new context of multivariate time series forecasting, which is especially applicable in the business context of earning trust with non-technical stakeholders who will use the models in practice. We provide the baseline for future LIME application in this new, multi horizon, multi time-series forecasting setting and identified relevant next steps for future research, namely being investigation into feature aggregation strategies.

With our work, we set the foundation for further theoretical research and empirical analysis of uncertainty quantification in a supply chain context at Amazon. Our package will be used and extended by Amazon Scientists to conduct further experiments and our research serves as a basis for future experiments and further theoretical research.

References

[1] Ruofeng Wen, Kari Torkkola, Balakrishnan Narayanaswamy, and Dhruv Madeka. A multi-horizon quantile recurrent forecaster, 2017. URL https://arxiv.org/abs/1711.11053.

[2] Ichiro Takeuchi, Quoc Le, Timothy Sears, Alexander Smola, et al. Nonparametric quantile estimation. 2006.

[3] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. "why should i trust you?": Explaining the predictions of any classifier, 2016. URL https://arxiv.org/abs/1602.04938.

[4] G.E.P. Box, G.M. Jenkins, G.C. Reinsel, and G.M. Ljung. *Time Series Analysis: Forecasting and Control*. Wiley Series in Probability and Statistics. Wiley, 2015.

[5] Everette S Gardner Jr. Exponential smoothing: The state of the art. *Journal of forecasting*, 4(1):1–28, 1985.

[6] Everette S Gardner Jr. Exponential smoothing: The state of the art part ii. *International journal of forecasting*, 22(4):637–666, 2006.

[7] Bryan Lim, Sercan O. Arik, Nicolas Loeff, and Tomas Pfister. Temporal fusion transformers for interpretable multi-horizon time series forecasting, 2019. URL https://arxiv.org/abs/1912.09363.

[8] Alexandre Drouin, Étienne Marcotte, and Nicolas Chapados. TACTiS: Transformer-attentional copulas for time series. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato, editors, *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 5447–5493. PMLR, 17–23 Jul 2022. URL https://proceedings.mlr.press/v162/drouin22a.html.

[9] Guillaume Chevillon. DIRECT MULTI-STEP ESTIMATION AND FORECASTING. *Journal of Economic Surveys*, 21(4):746–785, September 2007. ISSN 0950-0804. doi: 10.1111/j.1467-6419.2007.00518.x.

[10] Adèle Gouttes, Kashif Rasul, Mateusz Koren, Johannes Stephan, and Tofigh Naghibi. Probabilistic time series forecasting with implicit quantile networks. *CoRR*, abs/2107.03743, 2021. URL https://arxiv.org/abs/2107.03743.

[11] Ruofeng Wen and Kari Torkkola. Deep generative quantile-copula models for probabilistic forecasting. *arXiv preprint arXiv:1907.10697*, 2019.

[12] Bryan Lim, Sercan O. Arik, Nicolas Loeff, and Tomas Pfister. Temporal fusion transformers for interpretable multi-horizon time series forecasting, 2019. URL https://arxiv.org/abs/1912.09363.

[13] Carson Eisenach, Yagna Patel, and Dhruv Madeka. Mqtransformer: Multi-horizon forecasts with context dependent and feedback-aware attention. *CoRR*, abs/2009.14799, 2020. URL https://arxiv.org/abs/2009.14799.

[14] Galen R. Shorak. *Probability for Statisticians*. Springer New York, New York, 2000.

[15] Roger Koenker and Gilbert Bassett. Regression quantiles. *Econometrica*, 46(1):33–50, 1978. ISSN 00129682, 14680262. URL http://www.jstor.org/stable/1913643.

[16] Abe Sklar. Random variables, joint distribution functions, and copulas. *Kybernetika*, 9:449–460, 1973.

[17] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. URL http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[18] Ryan O'Connor. PyTorch vs TensorFlow in 2022. *News, Tutorials, AI Research*, May 2022. URL https://www.assemblyai.com/blog/pytorch-vs-tensorflow-in-2022.

[19] Alexander Alexandrov, Konstantinos Benidis, Michael Bohlke-Schneider, Valentin Flunkert, Jan Gasthaus, Tim Januschowski, Danielle C. Maddix, Syama Rangapuram, David Salinas, Jasper Schulz, Lorenzo Stella, Ali Caner Türkmen, and Yuyang Wang. GluonTS: Probabilistic and Neural Time Series Modeling in Python. *Journal of Machine Learning Research*, 21(116):1–6, 2020. URL http://jmlr.org/papers/v21/19-820.html.

[20] Tianqi Chen, Mu Li, Yutian Li, Min Lin, Naiyan Wang, Minjie Wang, Tianjun Xiao, Bing Xu, Chiyuan Zhang, and Zheng Zhang. Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems. *arXiv preprint arXiv:1512.01274*, 2015.

[21] Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. WaveNet: A Generative Model for Raw Audio. *arXiv*, September 2016. doi: 10.48550/arXiv.1609.03499.

[22] Aaron van den Oord, Nal Kalchbrenner, Oriol Vinyals, Lasse Espeholt, Alex Graves, and Koray Kavukcuoglu. Conditional Image Generation with PixelCNN Decoders. *arXiv*, June 2016. doi: 10.48550/arXiv.1606.05328.

[23] Yitian Chen, Yanfei Kang, Yixiong Chen, and Zizhuo Wang. Probabilistic Forecasting with Temporal Convolutional Neural Network. *arXiv*, June 2019. doi: 10.48550/arXiv.1906.04397.

[24] Carson Eisenach, Yagna Patel, and Dhruv Madeka. Mqtransformer: Multi-horizon forecasts with context dependent and feedback-aware attention, 2020. URL https://arxiv.org/abs/2009.14799.

[25] Jan Beitner. ImplicitQuantileNetworkDistributionLoss — pytorch-forecasting documentation.

[26] William Falcon et al. Pytorch lightning. *GitHub. Note: https://github.com/PyTorchLightning/pytorch-lightning*, 3, 2019.

[27] Uci electricity. https://archive.ics.uci.edu/ml/datasets/ElectricityLoadDiagrams20112014. [Online; accessed 29. July 2022].

[28] Stallion. https://archive.ics.uci.edu/ml/datasets/ElectricityLoadDiagrams20112014. [Online; accessed 29. July 2022].

[29] Bernhard Schölkopf, Alexander J Smola, Francis Bach, et al. *Learning with kernels: support vector machines, regularization, optimization, and beyond*. MIT press, 2002.

[30] George Em Karniadakis, Ioannis G. Kevrekidis, Lu Lu, Paris Perdikaris, Sifan Wang, and Liu Yang. Physics-informed machine learning. *Nature Reviews Physics*, 3(6): 422–440, May 2021. URL https://doi.org/10.1038/s42254-021-00314-5.

[31] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

[32] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[33] Paris Perdikaris Maziar Raissi and George Em Karniadakis. Physics informed deep learning (part i): Data-driven solutions of nonlinear partial dierential equations. 2017. URL https://arxiv.org/abs/1711.10561.

[34] Franco Scarselli and Ah Chung Tsoi. Universal approximation using feedforward neural networks: A survey of some existing methods, and some new results. *Neural networks*, 11(1):15–37, 1998.

[35] Anuj Karpatne Jordan Read Jacob Zwart Michael Steinbach Vipin Kumar Xiaowei Jia, Jared Willard. Physics guided machine learning for scientific discovery: An application to simulating lake temperature profiles. 2020. URL https://arxiv.org/abs/2001.11086.

[36] Judea Pearl. Causal diagrams for empirical research. *Biometrika*, 82(4):669–688, 1995.

[37] Markus Kalisch and Peter Bühlman. Estimating high-dimensional directed acyclic graphs with the pc-algorithm. *Journal of Machine Learning Research*, 8(3), 2007.

[38] Rui Wang, Danielle Maddix, Christos Faloutsos, Yuyang Wang, and Rose Yu. Bridging physics-based and data-driven modeling for learning dynamical systems. In Ali Jadbabaie, John Lygeros, George J. Pappas, Pablo Parrilo, Benjamin Recht, Claire J. Tomlin, and Melanie N. Zeilinger, editors, *Proceedings of the 3rd Conference on Learning for Dynamics and Control*, volume 144 of *Proceedings of Machine Learning Research*, pages 385–398. PMLR, 07 – 08 June 2021. URL https://proceedings.mlr.press/v144/wang21a.html.

[39] Bill Roach. Origin of the economic order quantity formula; transcription or transformation? *Management Decision*, 43(9):1262–1268, 2005.

[40] Wojciech Samek, editor. *Explainable AI: Interpreting, Explaining and Visualizing Deep Learning*. Springer Cham, 2019.

[41] Lloyd S Shapley. A value for n-person games. In Harold W. Kuhn and Albert W. Tucker, editors, *Contributions to the Theory of Games II*, pages 307–317. Princeton University Press, Princeton, 1953.

[42] Benedek Rozemberczki, Lauren Watson, Péter Bayer, Hao-Tsung Yang, Oliver Kiss, Sebastian Nilsson, and Rik Sarkar Sarkar. The shapley value in machine learning, 2022. URL https://arxiv.org/abs/2202.05594.

[43] Scott Lundberg. Shap (shapley additive explanations). https://github.com/slundberg/shap. [Online; accessed 25. Jul. 2022].

[44] Marco Tulio Ribeiro. Lime: Explaining the predictions of any machine learning classifier. https://github.com/marcotcr/lime. [Online; accessed 25. Jul. 2022].

[45] Emanuel Metzenthin. Lime for time. https://github.com/emanuel-metzenthin/Lime-For-Time. [Online; accessed 21. Jun. 2022].

[46] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.