

POLITECNICO DI TORINO

Master's Degree in Artificial Intelligence Data
Analytics



Master's Degree Thesis

Interactive Dashboard with Predictive system for Buffers Management Optimization

Supervisor

Paolo GARZA

Candidate

Marco CAPUSSO

November 2022

Summary

The advent of Deep Neural Networks has shown how powerful these kinds of models can be and how well they can perform in many different fields. Incredible results have been obtained in Image Classification and Natural Language Processing, but lately their usage was extended also to Time Series Forecasting. Deep Models are able to overcome some of the main limitations of the classical Statistical models and allow to discover hidden relations among different features.

Even though those models are widely used nowadays, they are rarely adopted in Time Series Forecasting.

In this thesis we will see a comparison between the most widely used models in order to predict the optimal stock quantities for the company products, as well as a descriptive Dashboard that aims to display all the most relevant information to the sales people at a glance.

The success of this project would bring relevant benefits to the company which will be able to drastically cut the stock management expenses.

Acknowledgements

“A tutte le persone che mi sono state affianco durante questo lungo viaggio”

Contents

List of Tables	VII
List of Figures	VIII
Acronyms	X
1 Introduction	1
1.1 Business Requirements	1
1.2 Overview to Dashboards	2
1.3 Overview to Time Series Forecasting	2
1.4 Research Goals and Methodologies	2
2 Forecasting Approaches	4
2.1 State of Art in Forecasting	4
2.1.1 ARIMA	4
2.1.2 SARIMAX	5
2.1.3 Prophet	6
2.2 Novel Approaches Proposed	8
2.2.1 NHiTS	8
2.2.2 DeepAR	9
2.2.3 Temporal Fusion Transformer	10
3 Descriptive Approach for Buffer Statistics calculation	15
3.1 Data Analysis	15
3.2 Architecture and Framework	17
3.3 Dashboard Data Update	17
3.4 Authentication	19
3.5 Dashboard details	19
3.5.1 Buffers	20
3.5.2 Analytics	22

4	Implementation of the Predictive Model for Buffer Forecasting	25
4.1	Data Analysis and Pre-processing	25
4.2	Framework and Architecture	28
4.3	Models Overview	29
4.4	Feature Selection	30
4.5	Models Comparison	32
4.5.1	ARIMA	32
4.5.2	Prophet	33
4.5.3	DeepAR	33
4.5.4	N-HiTS	33
4.5.5	Temporal Fusion Transformer	33
4.5.6	Final Comparison	34
4.6	Model Optimization	34
4.6.1	Hyper-parameters	34
4.6.2	Loss Function	36
4.6.3	Fine-tuning Considerations	36
4.7	Predictive Implementation	38
5	Results	42
6	Conclusions	45
	Bibliography	47

List of Tables

4.1	Quick trials for Models Comparison	34
-----	--	----

List of Figures

2.1	ARIMA parameters adjustment	5
2.2	SARIMAX parameters adjustment	6
2.3	N-HITS Structure from paper [2]	8
2.4	DeepAR Structure from paper [3]	10
2.5	TFT Structure from paper [4]	12
3.1	Dashboard data flow	16
3.2	Micro-services Architecture	17
3.3	Dashboard data Update Notebook	18
3.4	Dashboard Buffer Page	20
3.5	Buffer Statistics Popup	22
3.6	Buffer Details Popup	23
3.7	Buffer Filtered Critical	23
3.8	Buffer Analytics Section	24
4.1	Raw Data	26
4.2	Smoothed Data	27
4.3	Smoothed Averaged Data	27
4.4	Predictive Architecture	28
4.5	Sales Process Funnel	31
4.6	Static Variables Importance	37
4.7	Time Varying Variables Importance	38
4.8	Attention 64 steps	39
4.9	Attention 64 steps	40
4.10	SMAPE Comparison for Attention Sizes	41
4.11	Predictions Making Workflow	41
4.12	Predictions Table Example	41
5.1	SMAPE Cross-Validation	43

Acronyms

AI

Artificial Intelligence

LSTM

Long Short Term Memory

RNN

Recurrent Neural Network

CNN

Convolutional Neural Network

MLP

Multi Layer Perceptron

ARIMA

Autoregressive Integrated Moving Average

ETS

Exponential Smoothing

TFT

Temporal Fusion Transformer

ELU

Exponential Linear Unit

GRN

Gated Residual Network

GLU

Gated Linear Units

Chapter 1

Introduction

1.1 Business Requirements

The work treated in the following thesis was made in collaboration with Avnet, an international company that plays an important role in the distribution of semiconductors and passive components at a global level.

The company has to deal with a huge number of customers that can have significantly different needs and various problems. Some customers for example might need to order pieces once a year in bulk and for a predefined quantity, while some other customers might need to order some extraordinary and unexpected quantities. In order to be able to fulfill extraordinary requests from customers, a buffering system has been set up. It consists of private slots in the stock reserved for a specific customer. Whenever it seems that a customer might need a reserved amount of pieces due to possible extraordinary orders, a buffer is created and a certain quantity is allocated to it.

The quantity reserved to a specific customer should not be consumed from orders coming from other customers, therefore those pieces are basically locked for an undefined amount of time. Especially in a period like the current one, afflicted by semiconductors shortage, it is important to be able to deliver as many pieces as possible to the customers and avoid big amounts of materials sitting in stock for long periods, therefore it is important to handle buffers in the correct way and reduce as much as possible the reserved pieces in stock.

The current management of the buffers relies on fixed business rules based on a percentage of the previous year billings to choose the quantity to allocate to a buffer, which is not optimal and leads to situations where huge amount of pieces remain reserved for various months without ever being consumed.

The goal of this work is to provide to people that have to manage buffers, a simple

and intuitive dashboard that displays the most important features and a predictive approach that helps finding the optimal quantity to allocate to each buffer on a bi-monthly basis.

The predictive approach should allow to reduce the management costs of the stock in an important way by drastically decreasing the amount of pieces that remain reserved for long time, but it should also ensure that all the orders coming from customers can be fulfilled.

1.2 Overview to Dashboards

The Dashboard is a key element in Business analytics, it plays an extremely important role when it comes to visually display data in an intuitive way, and when it is necessary to display key information at a glance (such as KPIs).

Dashboards allow to take strategic decisions in an easier way by providing a general overview of the situation and they are usually linked to a source of data. In order to provide the highest possible flexibility, they should also allow users to select what kind of data they want to see by the mean of filters and selections.

1.3 Overview to Time Series Forecasting

Time series forecasting is the process of analyzing some historical data using statistics and artificial intelligence models in order to make predictions and enable strategic decision making. Time series analysis shows how data changes over time and a good forecasting can identify the direction in which the data is changing.

When dealing with Forecasting, it is always necessary to consider the amount of data at hand, the more points of observation are available, the better the understanding of the situation will be.

It is also important the concept of time horizon in forecasting, the time horizon is the amount of time steps that the model is trying to forecast in the future. Shorter horizons are much easier to forecast especially when the data is limited.

1.4 Research Goals and Methodologies

The goal of this work for the descriptive part is to provide a simple and intuitive way for the users to manage buffers and to see all the most relevant information at a glance, in order to help them being more efficient in the management and to

help their decision making by providing KPIs about market trends.

For the predictive part, many different models and approaches will be evaluated in order to find out the best performing one which allows to allocate the optimal quantity to buffers.

In the end, after evaluating all the results obtained during the training, some tests will be performed in order to estimate the possible cost reduction that would be obtained through the implementation of the predictive approach in some real case scenarios.

In the first section of Chapter 2 we will see a brief overview of the most popular predictive approaches and models (both statistical and deep) that are considered the state of art in classic time-series forecasting in order to get an idea about how the vast majority of predictions are performed nowadays.

In the second section of Chapter 2 we will see an overview of some proposed models that can better fit the described business problem, in this case we will evaluate the strengths provided by those models as well as some technical details that lie beneath.

In Chapter 3 we will dive into the details of the development of the descriptive Dashboard starting from the architectural choices, going through the frameworks used and explaining also the management of the users.

We will then take a look at the final prototype that has also been proposed to the stakeholders in the organization.

In Chapter 4 we will have a complete overview about the predictive model, from the analysis of the data-set and the data pre-processing to the training of the model with the hyper-parameters tuning.

In Chapter 5 we will evaluate all the results obtained from the different models on which the tests were performed and make some comparisons among different metrics.

After exploiting the best performing model, we will evaluate the real impact that the model would have weather it was applied to a real case scenario and we will evaluate in percentage how much we would be able to save if the model was implemented.

Chapter 2

Forecasting Approaches

2.1 State of Art in Forecasting

2.1.1 ARIMA

In time series analysis, an autoregressive integrated moving average (ARIMA) model is a generalization of an autoregressive moving average (ARMA) model. Both of these models are fit to time series data either to better understand the data or to predict future points in the series (forecasting).

ARIMA is an univariate model hence it can only learn and predict values considering the target feature. The parameters which are relevant in the ARIMA model are:

- p - which represents the number of autoregressive terms.
- d - which represents the number of nonseasonal differences.
- q - which represents the number of moving average terms.

In order to build an ARIMA model the first step is to stationarize the series, if needed, by differencing. Then the pattern of autocorrelations needs to be studied in order to determine if lags of the stationarized series and lags of the forecast errors should be included in the equation. After that, the model needs to be fit to the time series with the adjustment of the previously described parameters.

The AR (which stands for "Auto Regressive") part of ARIMA indicates that the evolving variable of interest is regressed on its own lagged values, the value of AR defines how many steps in the past we are taking into account in order to calculate the next value (as we can see in equation: 2.1).

$$X_t = c + \sum_{i=1}^p \phi_i X_{t-i} + \epsilon_t \quad (2.1)$$

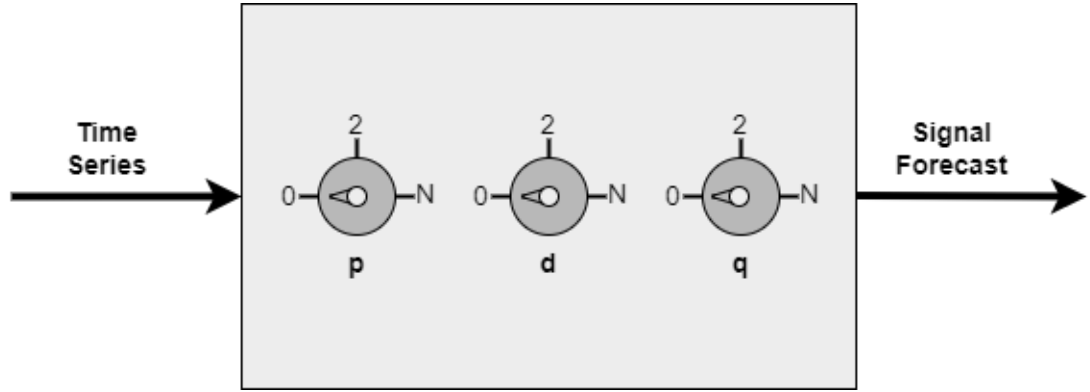


Figure 2.1: ARIMA parameters adjustment

The MA (which stands for "Moving Average") part indicates that the regression error is actually a linear combination of error terms whose values occurred contemporaneously and at various times in the past (as we can see in equation 2.2).

$$X_t = \mu + \epsilon_t + \sum_{i=1}^q \theta_i \epsilon_{t-i} \quad (2.2)$$

The I (which stands for "integrated") indicates that the data values have been replaced with the difference between their values and the previous values (and this differencing process may have been performed more than once). The purpose of each of these features is to make the model fit the data as well as possible.

2.1.2 SARIMAX

SARIMAX is used on data sets that have seasonal cycles (unlike ARIMA). The difference between ARIMA and SARIMAX is the seasonality and exogenous factors (seasonality and regular ARIMA do not mix well).

These are a bit complicated, but the most important thing to remember is that SARIMAX requires another set of p, d, q arguments for the seasonality aspect as well as an argument called "s" which is the periodicity of the data's seasonal cycle. When choosing an s value we need to try to get an idea of when the seasonal data cycles. If for example, your data points are separated by a monthly basis and the seasonal cycle is a year, then we need to set s to 12.

For all the values other than 's' is recommended to use a grid search approach in order to find the optimal parameters combination that better perform with the time-series.

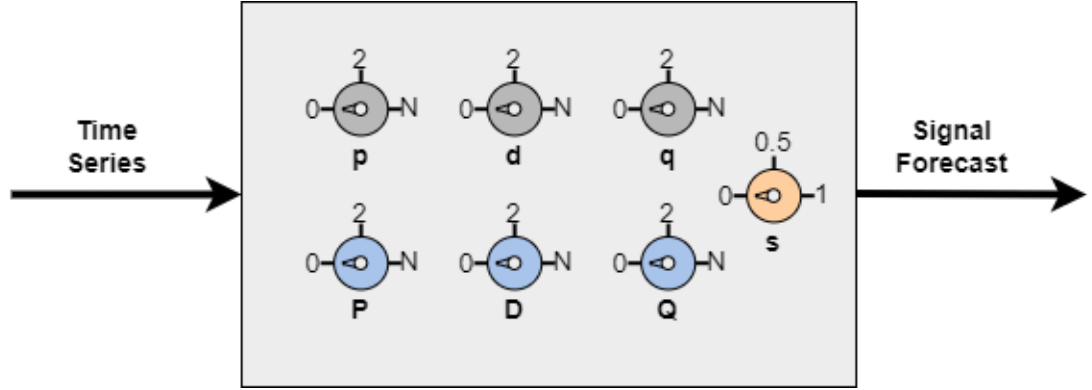


Figure 2.2: SARIMAX parameters adjustment

2.1.3 Prophet

It is a model introduced by Facebook in 2018, it was originally meant to forecast daily data with weekly and yearly seasonality by taking into account the holiday effects. Later on it has been extended in order to cover more types of seasonal data. It works best with time series that have strong seasonality and several seasons of historical data.

Prophet can be considered a nonlinear regression model that can be expressed in the following way:

$$y_t = g(t) + s(t) + h(t) + \epsilon_t, \quad (2.3)$$

In the equation 2.3, the term $g(t)$ describes a linear trend, $s(t)$ describes the various seasonal patterns, $h(t)$ represents the holiday effects and ϵ_t is a white noise error term.

- The knots for the linear trend are automatically selected if not explicitly specified. Optionally, a logistic function can be used to set an upper bound on the trend.
- The seasonal component consists of Fourier terms of the relevant periods. By default order 10 is used for annual seasonality and 3 is used for weekly seasonality.
- Holiday effects are added as simple additional variables.

As stated by Sean Taylor in [1], with Prophet we give up some important inferential advantages of using a generative model (such as an ARIMA), but we still provide some practical advantages:

- Flexibility: we can easily accomodate seasonality with multiple periods and let analytss make different assumptions about trends.
- The measurements do not need to be regularly spaced and we do not need to interpolate missing values.
- Fitting is very fast, allowing to interactively explore many model specifications.
- The forecasting model has easily interpretable parameters that can be changed to impose assumptions on the forecast.

2.2 Novel Approaches Proposed

2.2.1 NHiTS

In this section, we will take a look at the N-HiTS model, which tries to extend and improve the Neural Basis Expansion Analysis approach (N-BEATS) in many different aspects, making it more accurate in the predictions and more computationally efficient, especially in the context of long-horizon forecasting (as stated in [2]).

N-HiTS uses multi-rate sampling of the input signal and scales the forecast synthesis in order to produce a hierarchical construction of the forecast, which results in an important reduction in terms of computational requirements, and improvement in forecasting accuracy.

This model basically divides the input data considering it in different ways (i.e. different time scales), and at last it reassembles the various signals produced in a hierarchical fashion.

As we can see in figure 2.3 proposed in the paper [2], the model performs nonlinear projections onto different basis functions in order to try to extract different characteristics from the input signal.

Each projected input passes through a multi-layer perceptron (MLP) which produces the coefficients either for the forecast and for the back-cast. The forecasts are then summed up together in order to produce the final prediction. At the input

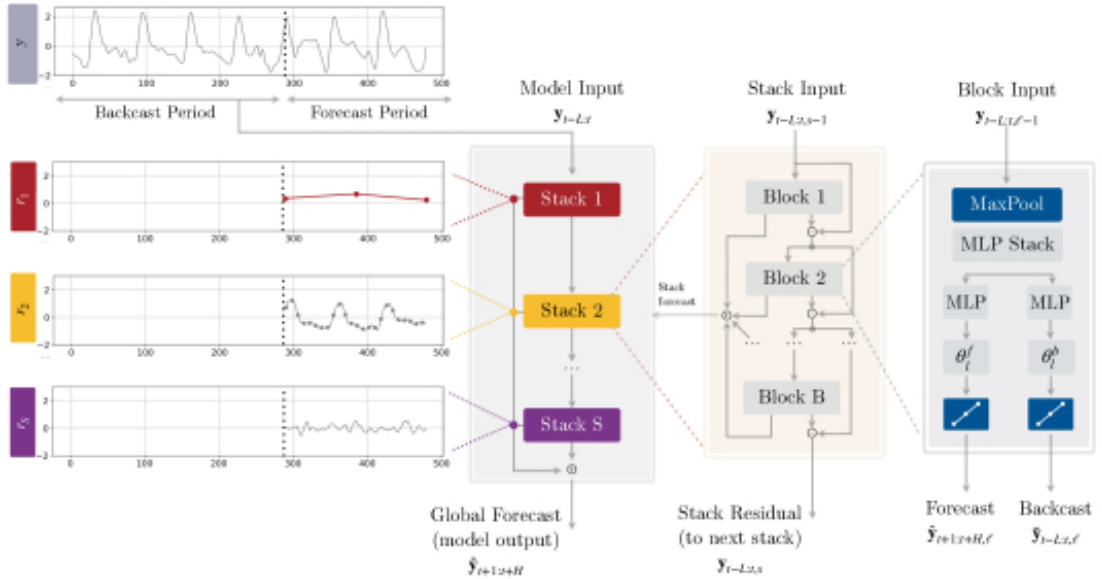


Figure 2.3: N-HiTS Structure from paper [2]

of each block, N-HiTS implements a MaxPool layer (similar to the ones used in

the very popular CNN) with a kernel size k , which helps to focus on analyzing the components of the input with a specific scale. Larger values for the kernel size tend to cut out small time-scale components from the input of the MLP, forcing the block to focus on analyzing the content on a large scale.

The term "multi-rate sampling" derives from the fact that each MLP faces a different effective input which is generated using a specific sampling rate.

Multi-rate processing is also very useful in reducing the width of the MLP input for most blocks, avoiding to excess in the usage of memory and the amount of computations. Furthermore, it reduces the number of learnable parameters, which allows to alleviate the effects of overfitting.

Given a specific block, the actions performed on the input are the following:

- MaxPool of the input is applied when the signal enters the block.

$$y_{t-L:t,l} = \text{MaxPool}(y_{t-L:t,l}, k_l)$$
- MLP calculations are performed after the subsampling through MaxPooling.

$$h_l = \text{MLP}_l(y_{t-L:t,l})$$

$$\theta_{l,f} = \text{LINEAR}^f(h_l)$$

$$\theta_{l,b} = \text{LINEAR}^b(h_l)$$
- Hierarchical interpolation is then performed with the results obtained through each MLP pass.
- With the constructed results obtained through Hierarchical Interpolation we are able to build the final predicted signal from which we can extract the future predictions.

2.2.2 DeepAR

DeepAR forecasting algorithm is a supervised learning algorithm which aims to forecast scalar (one-dimensional) time series using recurrent neural networks (RNN). Classical forecasting methods, such as autoregressive integrated moving average (ARIMA) or exponential smoothing (ETS), fit a single model to each individual time series. They then use that model to extrapolate the time series into the future and generate predictions.

Those kind of models however are not optimal when we have many similar time series across a set of similar units (for instance we have a different time series for each product), because in such cases it could be beneficial to train a single model over all of the time series.

DeepAR uses this approach and it is extremely effective when the dataset contains hundreds of related time series, since it is able to estimate predictions in a more robust way and can take into account relations between different materials.

DeepAR consistently outperforms the standard methods such as ARIMA and ETS and the trained model can be also used to generate forecasts on new time series. The input we provide to this model can be either one or more target time series that are somehow similar. According to the input dataset, the algorithm trains a model that learns the approximation of the time-series and uses it to predict how the target time-series evolves.

Each time-series can be eventually associated either with a vector of time independent categorical features (category) and with a vector of time dependent features (dynamic features).

As we can see in figure 2.4 taken from the paper [3], the training is performed

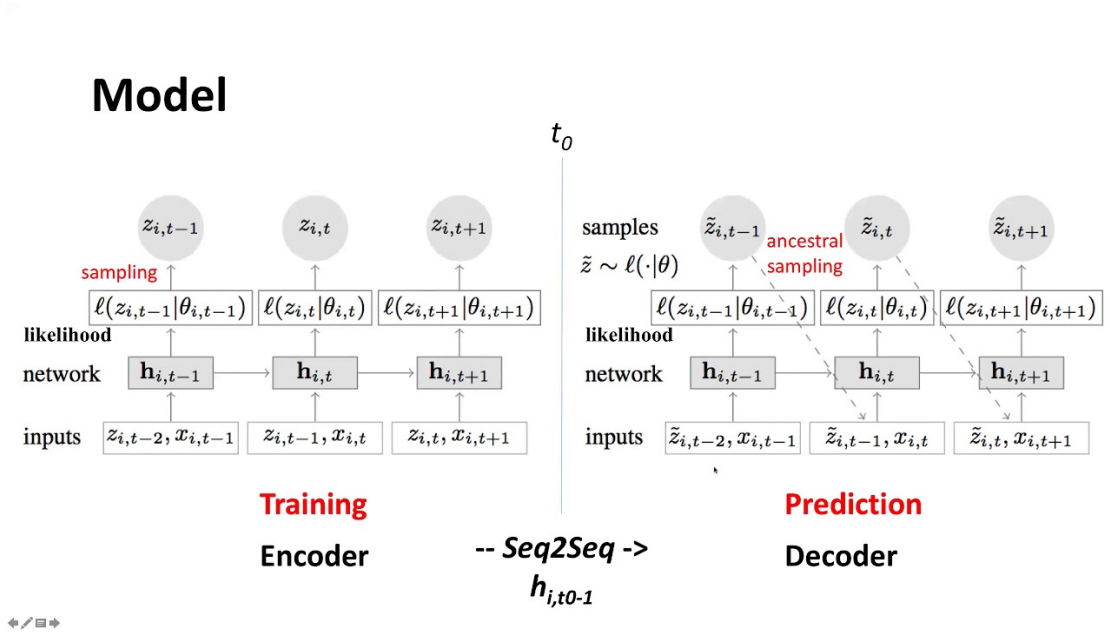


Figure 2.4: DeepAR Structure from paper [3]

by means of an Encoding model, the data provided in input is transformed into a sequence using a Recurrent Neural Network (RNN).

The prediction is then performed by a Decoding model which takes in input the sequence produced by the Encoder and generates the predictions.

2.2.3 Temporal Fusion Transformer

TFT is an attention-based Deep Neural Network architecture suited for multi-horizon time-series forecasting. It is able to achieve extremely high performances while also enabling new forms of interpretability of the data.

It implements a lot of novel ideas compared to the standard state-of-the-art architectures such as:

- Static covariate encoders - encode the context vectors in order for them to be used in other parts of the network.
- Gating mechanism - allows to perform a sample-dependent variable selection in order to minimize the contributions of inputs which are not very relevant for the predictions.
- Sequence-to-sequence layer - in order to process known and observed inputs.
- Self-attention decoder - which is trained to learn any long-term dependency within the dataset and it is able to generate the predictions from the encoded vector.

Attention mechanisms are often used in language processing, image classification or tabular learning since it is able to identify relevant portions of input for each instance using the magnitude of the attention weights. Recently those mechanisms have been adapted in order to be usable in the time-series, where they allow to find the most significant portions of the time-series which are more impacting in the future outcome.

With TFTs we can also evaluate the importance of the different variables in order to simplify the process of feature selection. We can directly see what is the contribution of each variable for the behaviour of the time-series.

As stated in [4], as we perform Multi-horizon Forecasting, let's say we have I unique entities in the dataset provided (in this case the different entities are the couples Material/Customer). Each entity \mathbf{i} is associated with a set of static covariates $\mathbf{S}_i \in \mathbb{R}^{m_s}$ as well as the inputs $x_{i,t} \in \mathbb{R}^{m_x}$ and scalar target $y_{i,t} \in \mathbb{R}$ at each time-step $t \in [0, T_i]$.

Time-dependent input features are subdivided into two categories $x_{i,t} = [z_{i,t}^T, x_{i,t}^T]^T$, which represent observed inputs ($z_{i,t} \in \mathbb{R}^{m_z}$), can only be measured at each step and are unknown beforehand, and the known inputs ($x_{i,t} \in \mathbb{R}^{m_x}$) which can be predetermined (day-of-week at time t ..).

In this model we adopt a quantile regression in order to make our predictions more reliable and take into account either worst and best cases. Each quantile takes the following form:

$$\hat{y}_i(q, t, \tau) = f_q(\tau, y_{i,t-k:t}, z_{i,t-k:t}, x_{i,t-k:t+\tau}, s_i), \quad (2.4)$$

where $\hat{y}_i(q, t, \tau)$ is the predicted q^{th} sample quantile of the τ -step-ahead forecast at time t and $f_q(\cdot)$ is a prediction model. In TFTs we simultaneously output forecasts for τ_{max} time steps.

The main components of the TFT model are:

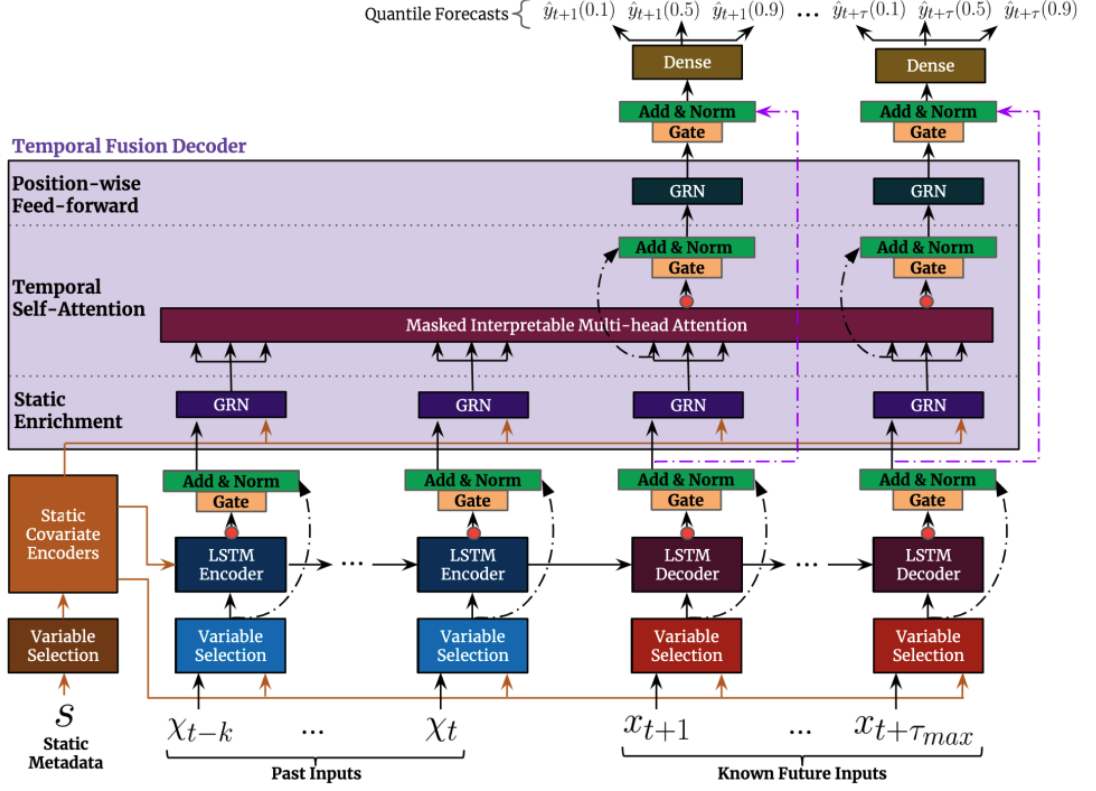


Figure 2.5: TFT Structure from paper [4]

- Gating mechanisms - allow to skip over any unused component of the architecture, providing varying depth and network complexity in order to adapt to different scenarios.
- Variable selection networks - allow to select relevant input variables at each time step.
- Static covariate encoders - allow to integrate static features inside the network, by means of context vectors encoding to condition temporal dynamics.
- Temporal processing - allows to learn long and short term temporal relationships among the data. It makes use of a sequence-to-sequence layer for local processing and a multi-head attention block in order to spot long-term dependencies.
- Prediction intervals - allow to generate forecasts for different prediction horizons.

The Gating Mechanisms are implemented through Gated Residual Networks (GRN), which takes as primary input a and an optional context vector c and yields:

$$GRN_w(a, c) = LayerNorm(a + GLU_w(\nu_1)), \quad (2.5)$$

$$\nu_1 = W_{1,w}\nu_2 + b_{1,w}, \quad (2.6)$$

$$\nu_2 = ELU(W_{1,w}a + W_{3,w}c + b_{2,w}), \quad (2.7)$$

where ELU is the activation function and $\nu_1 \in \mathbb{R}^{d_{model}}$, $\nu_2 \in \mathbb{R}^{d_{model}}$ are intermediate layers, LayerNorm is a normalization layer and w represents the weights.

When $W_{2,w}a + W_{3,w}c + b_{2,w} \gg 0$, the activation function would act as an identity function, if it is lower than zero instead the function would generate a constant output. The model performs component gating through GLUs to provide flexibility to suppress any part of the architecture. Given an input $\gamma \in \mathbb{R}^{d_{model}}$, the GLU takes form:

$$GLU_w(\gamma) = \sigma(W_{4,w}\gamma + b_{4,w}) \odot (W_{5,w}\gamma + b_{5,w}), \quad (2.8)$$

where $\gamma(\cdot)$ is the sigmoid activation function, $W_{(\cdot)}$ represents the weights and $b_{(\cdot)}$ represents the bias term.

GLU allows to control how much the GRN contributes to the original input a , potentially skipping entire layers if necessary.

We usually provide many variables to this model but they might have different relevance (in terms of contribution) in order to generate the predictions. TFT is designed to provide a variable selection mechanism that can be applied to both static covariates and time-dependent covariates. This mechanism allows to understand which variables are most significant for the predictions and also allow to remove unnecessary noisy inputs which could impact performances.

In addition, TFT employs the self-attention mechanism to learn long-term relationships across different time steps. In general attention mechanisms scale values $V \in \mathbb{R}^{N \times d_v}$ according to the relationships between keys $K \in \mathbb{R}^{N \times d_{attn}}$ and queries $Q \in \mathbb{R}^{N \times d_v}$:

$$Attention(Q, K, V) = A(Q, K)V, \quad (2.9)$$

where $A(\cdot)$ is a normalization function:

$$A(Q, K) = Softmax(QK^T / \sqrt{d_{attn}}) \quad (2.10)$$

in order to further improve the learning capacity, multi-head attention is proposed:

$$MultiHead(Q, K, V) = [H_1, \dots, H_m]W_H, \quad (2.11)$$

$$H_h = \text{Attention}(QW_Q, KW_K, VW_V), \quad (2.12)$$

where we have a set of weights for keys, queries and values that are head-specific. The model then linearly combines the output concatenated from all heads H_h .

Chapter 3

Descriptive Approach for Buffer Statistics calculation

3.1 Data Analysis

The Dashboard main objective is to display key data about the Buffers in an extremely simple and intuitive way as well as performing some basic elaborations on data in order to extract relevant KPIs such as trends.

The main data extracted for this purpose were:

- Buffers table: which contains all the data about quantity refilled, quantity consumed from the buffers, material contained in the buffer, customer associated and many other details regarding the buffers.
- Orders table: which contains all the orders of the whole company, used to extract data about past orders in order to display a list of previous orders as well as the trend of the orders in the past year.
- Billings table: which contains all the billings of the whole company, it was used to display a list of previous billings and again the trend of billings. Billings are orders for which the payment was successfully completed and the invoice was emitted.
- Quotes table: which contains all the quotes asked from customers. Customers might request a quote for a part number when they are interested in buying it. A quote is often followed by an order. In the Dashboard a list of previous quotes is displayed.
- Lead time table: which contains all the information about the Lead time. The lead time is the amount of time needed for the supplier to provide new stock

for a certain material. The higher is the lead time, the more difficult would be to retrieve a certain part number. Those cases have to be treated very carefully because it will be easier to remain out of stock for those materials.

- Forecast table: which contains Forecast made from customers. Customers can communicate with Avnet what amount of pieces they will be needing probably in the next months.
- Open Stock table: which contains information about the current available stock for each material. It is used to display to the user the available quantity remaining for the material contained in the Buffer.
- Materials table: which contains information about all the materials sold by the company. This table is used to retrieve information about quality and obsolescence of materials.

All the data was extracted from an internal source and needed to be cleaned in order to be usable. All the necessary queries have been prepared in order to extract the data, a cleaning process has been set up with the aim to reduce the size of the data and to have usable information. After that, some elaborations were performed in order to calculate some aggregate features and monthly trends.

For the calculation of the trends, a monthly aggregation was performed in order to have a data point for each month, and out of these points, the best fitting line was calculated in order to check whether the trend was growing or decreasing.

Out of all the elaborated data, a single table was prepared that contains all the information retrieved so that it can be queries at a glance to retrieve all the data from the Dashboard.

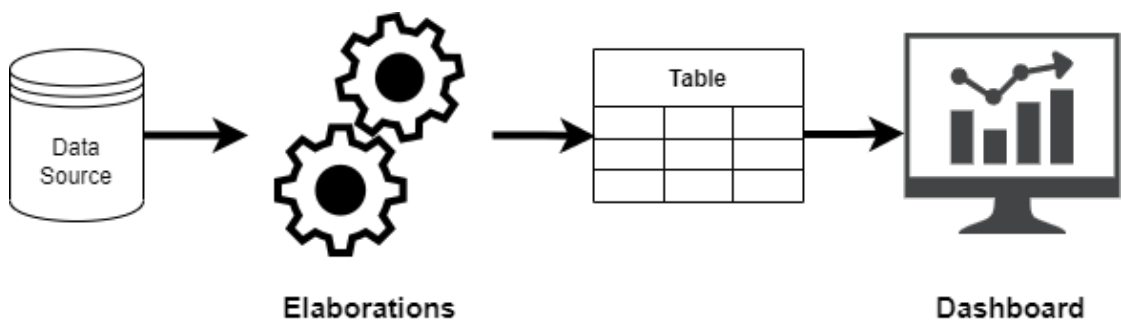


Figure 3.1: Dashboard data flow

3.2 Architecture and Framework

In order to be compliant with the best practises of Web Applications development, the Dashboard architecture relies on micro-services that interact both with the Data source and the Dashboard front end.

The main purpose of the micro-services architecture is to have well separated

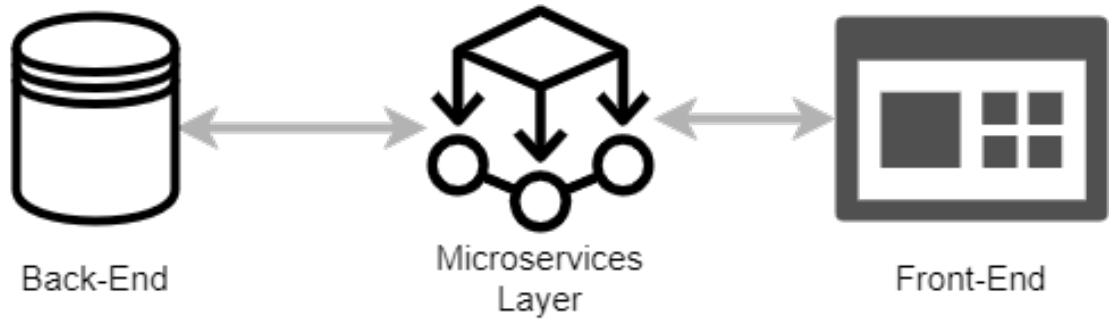


Figure 3.2: Micro-services Architecture

code in order to improve the maintainability of the code as well as the scalability. Furthermore, the APIs provided by the micro-services can be also used for other scopes beside the Dashboard.

The Back-end data is completely stored in some SQL tables and can be easily queries in order to retrieve all the needed information.

Micro-services instead were build with FastAPI, a Flask based framework that easily allows to build automatically documented APIs with Python. It is extremely simple to pick up and to develop routes (API calls) in a clean and organised way. For the Front-end part the framework chosen was React, a Javascript-based UI library created by Facebook which is supported by a huge community. React is very popular in web applications development nowadays and brings many benefits such as reusability of the code, high flexibility, scalability, a small learning curve and improved performances.

Besides all of the benefits given, the Dashboard will be implemented in the company's current most popular Application which is also developed in React.

3.3 Dashboard Data Update

All the data displayed in the Dashboard needs to be constantly updated, according to the actions takes from the users on the buffers and also according to the materials that are purchased every day.

A continuous update is certainly not feasible so another implementation should

be adopted in order to have data which is always up to date (we do not want to remove some pieces from a buffer using the dashboard and still see the same unmodified quantity after the page refresh).

For this data, a combination of approaches has been used in order to grant consistency, and it is divided into 2 main phases:

- Daily Data Refresh - there are some procedures that are run on a daily basis and they take care of extracting all the updated information that needs to be displayed in the dashboard.
- Delta Data Cache - all the actions that perform changes on the data are stored inside a cache database. Since there are not many actions happening throughout the day, the cached data will not be very heavy to manage and before displaying the information on the dashboard, we apply the delta operations to the data that we stored from the Daily Refresh.

All the operations that needs to be performed on the data on a daily basis are scheduled every night and are run by the means of Databricks, an online development environment that allows to create notebooks (similar to jupyter) and automatically trigger the run of those notebooks at a certain time.

All the operations are performed on dedicated clusters which are extremely powerful.

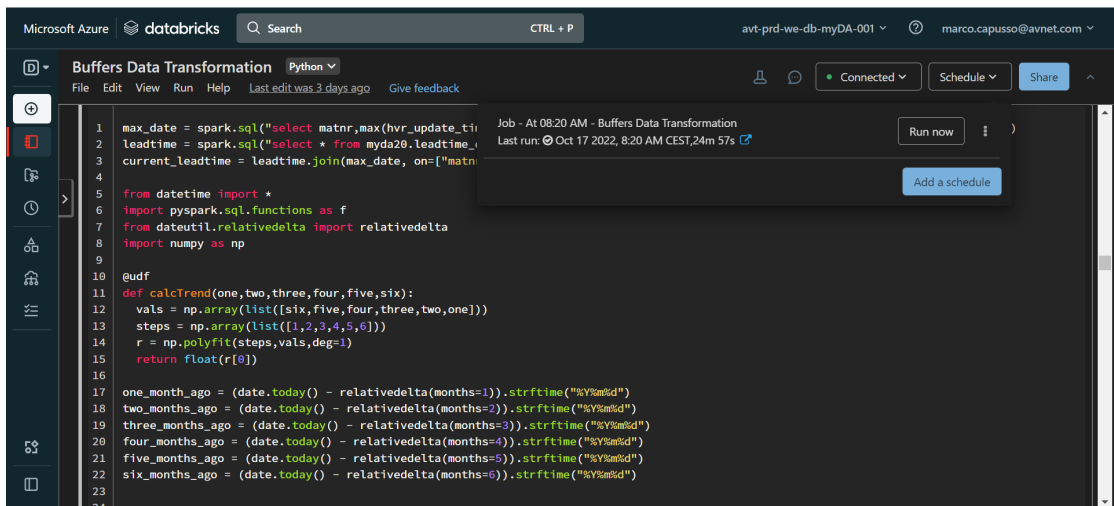


Figure 3.3: Dashboard data Update Notebook

3.4 Authentication

Authentication plays an extremely important role in business applications, in this case it was necessary to allow access to the application only to company members, and users needed to be able to see different information according to their email address.

In order to address all of this issues, and considering that the company uses a Microsoft Active Directory System in order to manage the identities within the organization, implementing a Microsoft SSO authentication was the best possible choice.

Due to the fact that it is not properly documented, the implementation of the Microsoft SSO has been very challenging but it was carried out through the usage of the 'ADAL' library available in Python.

Authentication phases:

- When a user access the Dashboard for the first time or after the lease time of the authentication token has expired, an automatic redirect is triggered, taking the user to a page where the classic Microsoft log in can be performed.
- After the Microsoft log in is performed, a token is created and the user is redirected back to the Dashboard.
- All the information about the user are then retrieved by using the generated token and all the subsequent interactions to the Dashboard need the user to be logged in.

In this way, the Dashboard automatically allows only the member of the organization and it is able to retrieve the email of the logged user in order to display the correct information.

3.5 Dashboard details

The Dashboard has been divided into 2 main sections which allow the users to retrieve all the key information that could help them manage the buffers in an optimal and more efficient way:

- Buffers - where the users of the Dashboard are able to see all the buffers assigned to them with all the most relevant KPIs, and can eventually take some actions.
- Analytics - where the users of the Dashboard are able to see aggregated data about the Buffers cost for each country.

3.5.1 Buffers

In this section there are some filters in order to allow users to find specific buffers (it is possible to filter according to the customer name, material, etc..).

The information is divided into 4 main categories (as we can see in fig. 3.4): Customer, Buffer, Analysis and Proposition.

In the Customer part we can find relevant information about the customer such as the name and the sales document.

In the Buffer part we can find relevant information about the specific buffer:

Customer	Buffer	Analysis	Proposition
PIETRO FIORENTINI SPA Sales Doc. 90326637	Validity: 02/03/2021 - 02/03/2023 Material: ST2STM32L152VDT6TR Remaining Qty: 190000 pcs Confirmed Qty: (190000) pcs Suggested Qty: 131200 pcs	Buffer Type: A Size: 1.80% Forecast: Lead Time: ABC: C Life: 19% Bill: Stock:	The Material is Customer Specific, the Buffer should be Deleted. The Buffer should be Renewed/Closed (Material Type C).
PIETRO FIORENTINI SPA Sales Doc. 90326637	Validity: 02/03/2021 - 02/03/2023 Material: ST2TS332IDT Remaining Qty: 160000 pcs Confirmed Qty: (100000) pcs Suggested Qty: 100000 pcs	Buffer Type: A Size: 1.43% Forecast: Lead Time: ABC: A Life: 19% Bill: Stock:	No Proposition Suggested

Figure 3.4: Dashboard Buffer Page

- Validity - represents the life span of the buffer in term of dates.
- Material - represents the target material.
- Remaining Quantity - represents the quantity that should be stored inside the buffer (theoretically).
- Confirmed Quantity - represents the actual quantity inside the buffer, it might differ from the theoretical quantity in some cases where the order coming from the supplier has not been confirmed yet.
- Suggested Quantity - is the optimal quantity that should be stored inside the buffer according to the AI model.

In the Analysis part we can find key information about the historical data of the material and the customer:

- Buffer Type - defines the type of contract attached to the buffer.
- ABC - represents the popularity of the target material (from A - very popular material, to F - very customer specific material).
- Size - metric that represents the current size of the buffer compared to the previous year billings.
- Life - represents the remaining lifespan of the buffer (in terms of a percentage).
- Forecast - represents the trend of the forecast for that specific customer in the last 6 months.
- Bill - represents the trend of billings for that specific customer in the last 6 months.
- Lead Time - represents the trend of lead time for that specific material in the last 6 months.
- Stock - represents the trend of stock for that specific material in the last 6 months.

In the Proposition part we can find a suggestion that the Dashboard gives to the user for that every single buffer. The suggestion is calculated according to some business rules and the output generated by the predictive model.

On the right of the Buffer page we can find 3 action buttons that allow to perform different operations:

- Email Button - allows to quickly send emails to the managers of the buffer.
- Link Button - allows to open the buffer directly in the management software used inside the organization in order to perform some modifications on the buffer.
- Statistics Button - opens a popup which displays historical data about orders, billings and quotes for that specific buffer (as we can see in fig.3.6).

Clicking on a single buffer card instead opens a popup which contains further details about the buffer:

- Customer Name - the name of the customer.
- Buffer Inactivity Time - how much time has passed since the last order has been issued from the buffer.

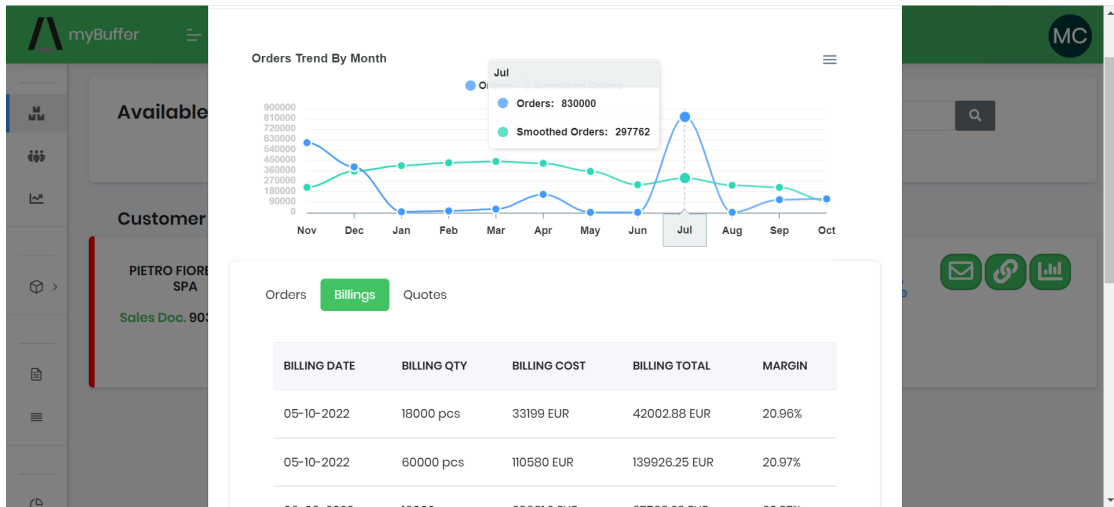


Figure 3.5: Buffer Statistics Popup

- Customer Number - customer code.
- Material - material stored inside the buffer.
- Account Manager - external sales person in charge for the buffer.
- Inside Sales - internal sales person in charge for the buffer.
- Consumed Quantity - quantity consumed from the buffer over the total quantity stored inside the buffer during its lifetime.
- Lead Time - lead time in days.

Besides filtering the available buffers according to the search bar, it is possible to select a specific portion of buffers according to their criticality (whether they need to be taken care of or not), by selecting the desired category in the switch button. As we select the Critical ones, we will obtain a situation similar to the one displayed in fig.3.7 (the critical buffers can be recognised by the red stripe on the side).

3.5.2 Analytics

In the Analytics section instead it is possible to have a complete overview of the status of the various buffers divided by country and office. As we can see in fig.3.8, in this section we can see:

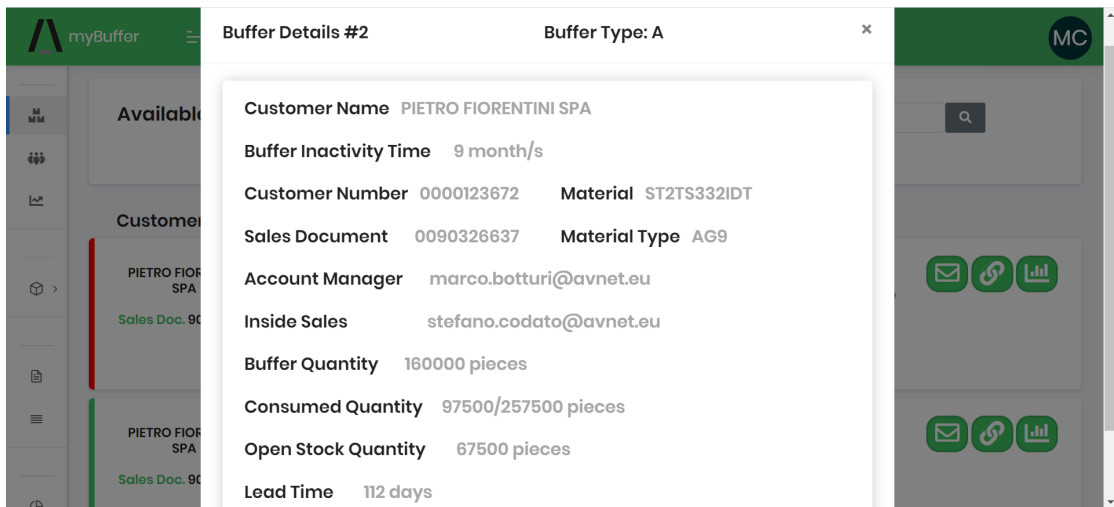


Figure 3.6: Buffer Details Popup

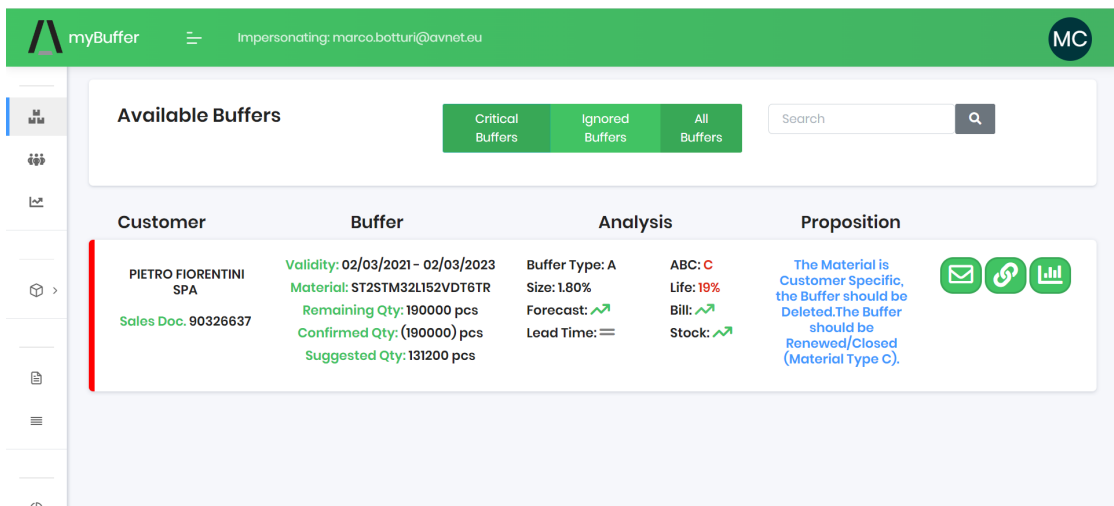
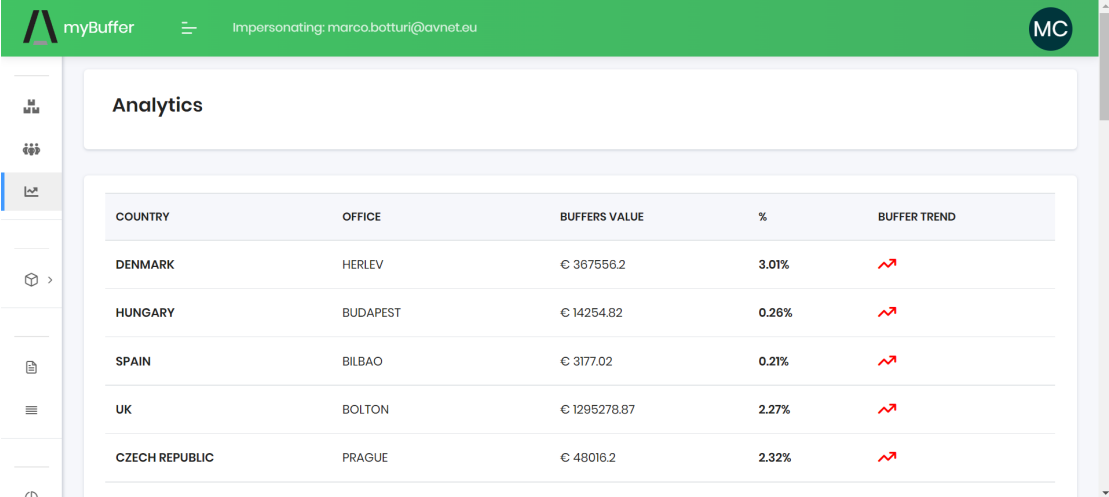


Figure 3.7: Buffer Filtered Critical

- Buffers Value, which is calculated for each office/country. It represents the sum of the value of all the pieces contained in all the buffers of the target office/country.
- Percentage, represents the value of the buffers for the target office/country divided by the value of the previous year billings for that same office/country. This metric helps us realize how much are the currently open buffers impacting.

- Buffer Trend, represents the trend of the open buffers in the last 6 months, indicating whether the value of open buffers is increasing or decreasing.



The screenshot shows the 'myBuffer' application interface. The top navigation bar is green and contains the 'myBuffer' logo, a user impersonation status 'impersonating: marco.botturi@avnet.eu', and a user profile icon 'MC'. On the left, there is a sidebar with icons for different sections. The main content area is titled 'Analytics' and displays a table with the following data:

COUNTRY	OFFICE	BUFFERS VALUE	%	BUFFER TREND
DENMARK	HERLEV	€ 367556.2	3.01%	↗
HUNGARY	BUDAPEST	€ 14254.82	0.26%	↗
SPAIN	BILBAO	€ 3177.02	0.21%	↗
UK	BOLTON	€ 1295278.87	2.27%	↗
CZECH REPUBLIC	PRAGUE	€ 48016.2	2.32%	↗

Figure 3.8: Buffer Analytics Section

Chapter 4

Implementation of the Predictive Model for Buffer Forecasting

4.1 Data Analysis and Pre-processing

The predictive approach's main objective is to provide optimal forecasts in order to be able to fulfill all the requests coming from customers. In order to be able to do that we extracted all the historical data coming from orders, billings and quotes as well as some information about the customers and the materials. The main data extracted in this phase were:

- Orders table: which contains all the orders of the whole company, used to extract all the information about past orders (in terms of quantity and price).
- Billings table: which contains all the billings of the whole company, it was used in order to extract all the information about past billings (in terms of quantity and price).
- Quotes table: which contains all the quotes asked from customers, used to extract all the information about past quotes (in terms of quantity and price).
- Materials table: which contains information about all the materials sold by the company. This table is used to retrieve information about quality of the materials which is gonna be used as a static feature during predictions.
- Customers table: which contains information about all the customers managed by the company, it is used in order to extract some static categorical features about the customers.

During the data analysis phase we immediately acknowledged various issues with the data that we needed to use, which led to the adoption of several approaches in order to reduce the impact of those problems.

The quantity that we wanted to predict with the models proposed was the orders value, which represents the values of the orders that we expect to receive in the next weeks.

Since we are working at a Buffer level, we needed to evaluate data at a very high granularity, we consider as a matter of fact every couple of Material-Customer separately.

Working at such as a high granularity obviously leads to lower accuracy and less robust predictions but given it was a Business requirement we could not bypass it. The most impacting problem caused by the high granularity is represented by the zero-data.

Since orders at a Material-Customer level are not performed very often, we face a lot peaks in the history of the orders as we can see in figure 4.1, because for several months we can receive no order for that specific Customer-Material combination. The solution adopted in order to deal with this kind of issue was to smooth the

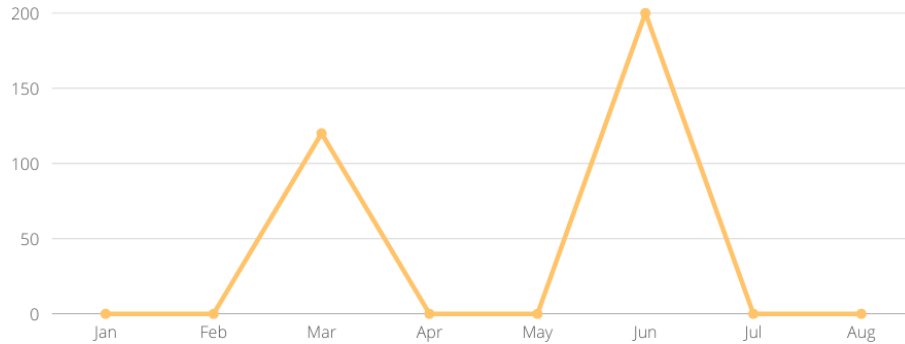


Figure 4.1: Raw Data

data using a sliding window. For each month, instead of taking the true value of that month, we used to take the averaged sum of the preceding and the subsequent month (as showed in figure 4.2), obtaining smoother data with less zero values allowing to train the model more efficiently and more precisely. Furthermore, if we would like to retain more information about the peaks without losing all the benefits provided by the windowing, another slight transformation could be performed on

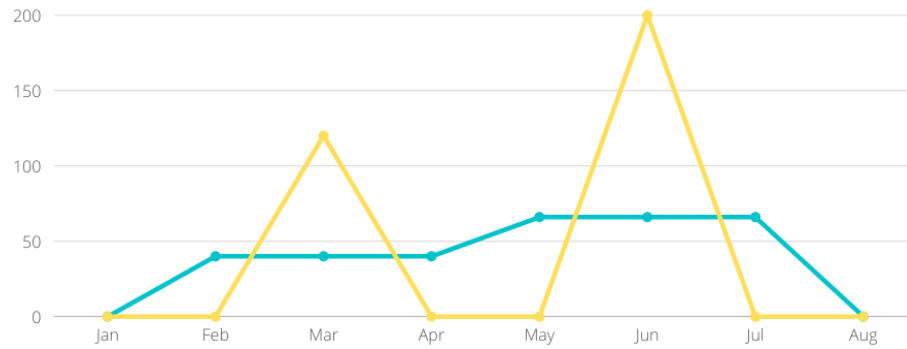


Figure 4.2: Smoothed Data

the data (as displayed in figure 4.3) which consists in averaging the Raw Data and the Smoothed Data, in this way we will retain more information about our data. With all these very simple transformations we have solved the main issue affecting

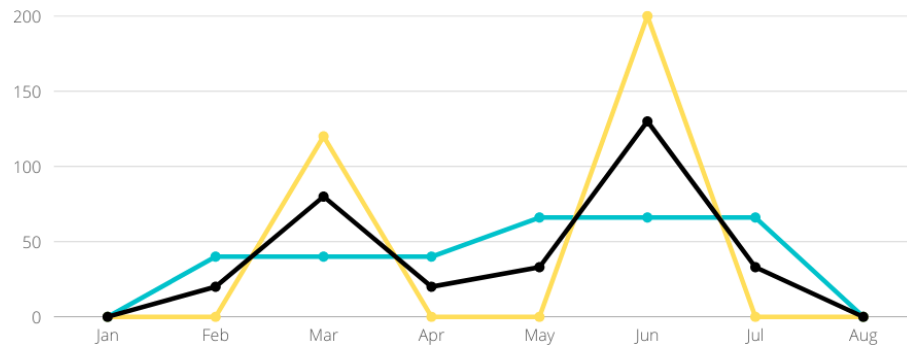


Figure 4.3: Smoothed Averaged Data

our data, however we had to face some other issue during the data preparation for

the predictive model:

- Null Data - in some cases we had to deal with null values in our dataset which were simply replaced with zero-values.
- Duplicated Data - in some cases we had duplicated values which could not be fed into the model, in that case we simply averaged the duplicated cases (which were often equal in terms of value) in order to get rid of duplicates.

4.2 Framework and Architecture

In order to set up the predictive system we made use of the tools provided by the company (Databricks) in order to prepare the predictive models and perform the data cleaning and the prediction calculation on a daily basis.

All the calculations are performed on separated notebooks which run on a dedicated cluster every day and, since we have many workers available, the data transformation was performed entirely using PySpark, which allows better parallelization over multi-worker environments. As displayed in figure 4.4, we have two different kinds of

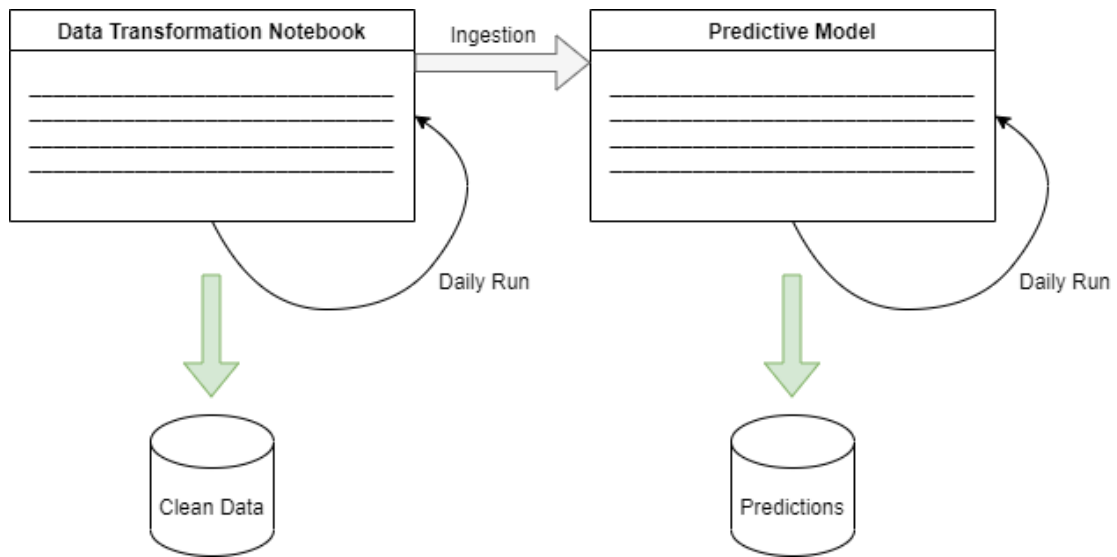


Figure 4.4: Predictive Architecture

notebooks being run on a daily basis that produce a specific outcome subsequently stored inside some SQL tables.

The Data Transformation notebooks are extremely important for the generation of new predictions because the output generated by those notebooks is directly fed into the predictive model which uses that data for the training.

The predictions notebook instead produces the daily prediction for each active buffer present in the database and stores the suggested quantity inside a table (the same table that will be used from the dashboard to display that information).

4.3 Models Overview

In this document we will compare and analyze 4 different models:

- ARIMA - the statistical uni-variate model extremely popular in time-series forecasting.
- Prophet - the statistical uni-variate model proposed by Facebook.
- N-HiTS - the multi-variate model that performs hierarchical interpolation relying on simple MLP.
- DeepAR - the recurrent uni-variate model that implements LSTMs in time-series forecasting.
- Temporal Fusion Transformer - a very sophisticated model that implements Transformer networks for time-series forecasting tasks.

Since there are many factors that concur in the change of the demand coming from a specific customer, uni-variate models (ARIMA and DeepAR in this case) can be quite limiting and extremely prone to lead to inaccurate predictions. Being able to feed only the target variable as input for the training might not be enough to understand the underlying information present in the time-series.

SARIMAX was discarded from the analysis because after a first evaluation on the data, there was no evidence about the presence of a seasonality in the extracted data.

In the other hand, multi-variate models can accomplish a higher degree of understanding of the data, at the cost of a higher computational demand. Multi-variate models allow to plug into the model several features and cross-validate the effectiveness and the relevance of those features in the resulting predictions.

DeepAR and TFT rely on Recurrent Neural Networks and are computationally more expensive than the simple N-HiTS which only performs projections of the input data over different basis and consequently passes the results into a MLP.

Statistical models such as ARIMA, SARIMAX and Prophet are extremely suitable for classical forecasting problems such as stock pricing.

ARIMA is the simplest model among all and it is very simple to implement and to understand what the different hyper-parameters represent (hence the huge popularity of this approach). Unfortunately this approach has some relevant down sides such as the fact it is an uni-variate model and the low level of generalization due

to the fact that we need to fit a different model for each time-series. SARIMAx is very similar to ARIMA but it takes into account seasonality as well. In our case we have noticed that time-series tend to have no seasonality at all hence the reason we immediately discarded this method.

Prophet is another quite simple model which can be implemented through several pre-made libraries, it presents the same down sides as ARIMA given it is a statistical model as well.

As we dive into the novel models proposed (N-HiTS, DeepAR, TFT), we face an important increase in terms of implementation complexity. All of these models rely on Neural Networks which come with a higher level of complexity.

Among the Deep Models, DeepAR is the older one and it is the most widely used Deep Model in time-series forecasting. Transformers, as DeepAR, is based on RNNs but should improve consistently the performances thanks to the more developed architecture and the implementation of an attention mechanism.

N-HiTS instead is completely different and does not rely on RNNs, making the training 50 times less expensive (in terms of computational effort) and in some cases it can reach higher accuracy compared to TFTs.

Model	Input	Train Time	Complexity
ARIMA	uni-variate	very fast	very low
Prophet	uni-variate	very fast	very low
DeepAR	uni-variate	slow	quite high
TFT	multi-variate	slow	very high
N-HiTS	multi-variate	quite fast	quite low

4.4 Feature Selection

When we talk about feature selection we necessarily talk about the multi-variate models since the uni-variate ones are only provided the target value as input for the training.

The first important thing to decide was whether trying to predict the orders or the billings. We refer as 'order' when we talk about a client submitting a request for a specific material. We refer as 'billing' when the order request is confirmed and an invoice is generated. We refer as 'quote' when a certain customer makes an informal request for a material in order to get to know what would be the price proposed according to the needed quantity.

As we can see in figure 4.5, we can think about the customer purchasing process as a 'funnel'. A customer can ask for a quote for a specific material and eventually, if the customer is satisfied from the price offered, an order is submitted. After

the order is submitted, it receives a confirmation and, unless an order cancellation occurs, it is delivered to the customer. Since we need to provide the correct quantity

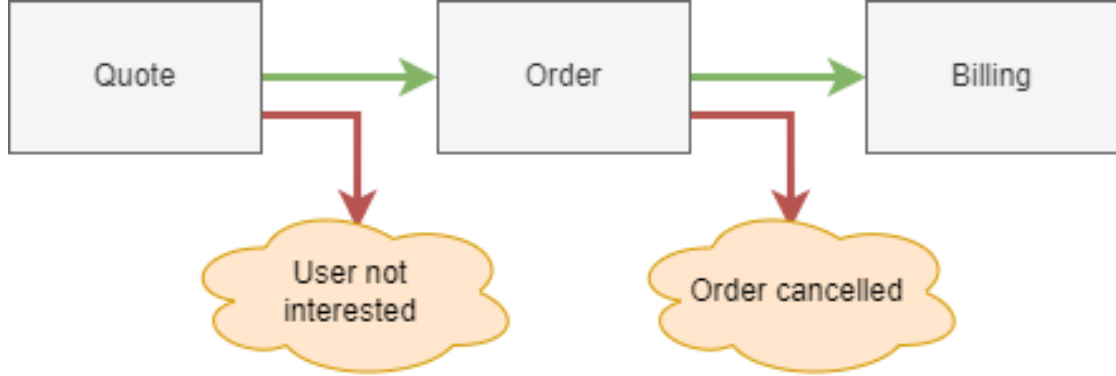


Figure 4.5: Sales Process Funnel

of materials in order to fulfill customers requests, we need to correctly predict the orders, hence the reason our target variable will be related to the orders.

Once we realise that we need to predict information about the orders, we need to understand what we want to predict. For each order we have information about dates, quantity, sale price and unit price, our target variable needs to be chosen among the ones proposed.

The order quantity could be at a first glance the best candidate since we want to predict how many pieces we want to store in our buffers but, doing so, we completely ignore the value of the target material. Since we cannot treat materials with completely different values in the same way (for instance a material could cost 1€/piece and another one could cost 10000€/piece), we directly try to predict the orders cost, which automatically takes into account both the unit price and the quantity purchased.

Another option could have been to use the unit price as a static variable and predict the quantity but this method could not be accomplished in uni-variate models so we will just stick to the first proposal.

Once we have decided which our target variable will be, we have to define our feature of interest for the multi-variate models. We need to find all the variables that might have a direct impact on changes on the orders in order to exploit a higher level of accuracy and reliability.

The data that have an immediate impact on the orders are definitely the historical data about orders, quotes and billings. As a matter of fact, if we receive a quote for a certain material, the customer is very likely to be submitting an order for that specific material soon in the future.

Besides that, we tried to add some other extracted features such as the total value

of the orders for each material without specifying the client (considering the orders at a Material level and not a Material-Customer level), some static features such as the value of that customer in our organization (how much does that specific customer impact on the sales compared to the others), and the popularity of the target material (how popular is the target material among our customers, how many of our customers buy that specific material).

4.5 Models Comparison

Since we did not want to proceed with a deep analysis and fine-tuning of the hyper-parameters for all the models, we performed some preliminary analysis in order to understand which could be the most promising model for our task.

In order to do that we prepared all the models mentioned before in order to evaluate the accuracy on the data extracted and cleaned for the training.

The first important thing that we needed to do in order to properly compare the different models proposed was to find a good evaluation metric.

In our case the choice fell on SMAPE, a reliable metric widely used in time-series forecasting.

$$SMAPE = \frac{1}{n} \sum_{t=1}^n \frac{|F_t - A_t|}{(F_t + A_t)/2} \quad (4.1)$$

Where F_t represents the predicted value and A_t represents the actual value. SMAPE fits very well our problem because it can manages cases where either the predicted value or the actual value are equal to zero (unlike other metrics such as MAPE). Furthermore, SMAPE tends to penalize more cases of under-forecasting rather than over-forecasting, which fits very well our scenario since under-forecasting a quantity means we are not able to provide the materials to the customers (that is extremely harmful for the sales people).

After we selected the proper metric, we started evaluating the performances of all the different models starting from the statistical models.

4.5.1 ARIMA

With ARIMA we performed some shallow grid-searching in order to find the optimal set of parameters that would fit our scenario. After some trials we were able to obtain a SMAPE around **0.94** which is not very promising in order to solve our task. Although the accuracy was not that good, the model was able to train and produce the predictions within 30 minutes only (very computationally efficient).

4.5.2 Prophet

With the Prophet model we performed some shallow grid searching as well in order to fine-tune the hyper-parameters and provide the best results. In this case we were able to reach with the best set of parameters, a SMAPE around **0.90**, which shows some slight improvements compared to the ARIMA model but which is not yet very promising as well. In this case the training process was quite slower and the predictions were carried out within 36 minutes.

4.5.3 DeepAR

As we started digging into Deep Models we acknowledged that we would have not been able to perform an accurate grid search for all the models given the wide number of hyper-parameters that they imply.

We therefore decided to perform very few trials in order to understand in general how well the model works with the proposed task without digging into each hyper-parameter.

After some trials with minor tweaking on dropout, RNN layers and some other key parameters, we obtained a minimum SMAPE of **0.84**, providing a further improvement compared to the statistical models. We assume this improvement is due to the ability of DeepAR to generalize information coming from different time-series in order to make predictions more reliable.

In this case the training time was much slower compared to the statistical models and was carried out in 3 hours and 32 minutes.

4.5.4 N-HiTS

For this model, as we did for DeepAR, we performed a very shallow grid search considering very few parameters in order to quickly estimate the performances of the model for the specific task.

In this case we have performed different trials varying the number of hidden layers of the MLPs and the dropout value. The minimum SMAPE obtained across the different trials was **0.72**, which represents a huge improvement compared to all the previous models.

The training time for this model was optimal as well in fact, in order to train the model, our cluster only took 43 minutes.

4.5.5 Temporal Fusion Transformer

Finally, we have performed the same operations also for the Temporal Fusion Transformer. This model was the most complex among all due to the extremely high number of hyper-parameters and the flexibility it comes with.

As before we selected some key hyper-parameters in order to perform once again a quick grid search and spot the general behaviour of the model.

We made some trials varying the size of the encoder and the depth of the RNN layers and the minimum SMAPE obtained was **0.70**, which is very similar to the results obtained with N-HiTS.

In this case the training time was much longer due to the fact that unlike N-HiTS which has a very simple architecture with simple MLPs, Transformers have a very complex RNN architecture which increases the training cost by almost 50 times. As a matter of fact, thanks to the high level of parallelization that our cluster can provide (due to the nature of Transformers which rely on a fully parallelizable architecture), we managed to run the whole training within 2 hours and 52 minutes.

4.5.6 Final Comparison

After performing those quick analysis on all the proposed model, we summarized the results obtained from the preliminary trials, as we can see in table 4.1, noticing that the best performances were obtained from the TFT model.

The N-HiTS model performed very well too and has extremely lower computational time but since we do not have any constraint in terms of computations (since we are provided of an extremely powerful cluster on the cloud environment) and we aim for the best possible accuracy, we decided to proceed with the analysis with the Temporal Fusion Transformer model in order to better fine-tune the hyper-parameters and obtain optimal results for the predictions.

Table 4.1: Quick trials for Models Comparison

Model	min. SMAPE	Training Time
ARIMA	0.94	30m
Prophet	0.90	36m
DeepAR	0.84	3h 32m
N-HiTS	0.72	43m
TFT	0.70	2h 52m

4.6 Model Optimization

4.6.1 Hyper-parameters

After we selected the model we wanted perform the deep hyper-parameters tuning process with, we analyzed in detail the meaning of each parameters along with

its contribute to the model. The most relevant hyper-parameters in the Temporal Fusion Transformer are:

- Static reals - numerical features that do not change over time. In our case we only had one static real feature which was generated through the extraction and the transformation of the orders performed by that specific customer. That feature represents the impact of that customer on the global sales compared to the other customers.
- Static categoricals - categorical features that do not change over time. In our case we have some static categoricals related to the target material and to the customer. For the material we have a feature that represents how popular the material is while for the customer we have a feature that represents the "Tier" of the customer.
- Dropout - it is used in order to reduce the phenomenon of over-fitting, according to the value selected (between 0 and 1), the model automatically ignores the calculations performed from certain nodes inside the model.
- Attention head size - number of attention heads used in the transformer blocks.
- LSTM layers - number of LSTM layers used in the model (usually 2 is a good choice).
- Hidden size - hidden size of the network.
- Loss - loss functions used to train the model.
- Max encoder length - maximum number of time-steps the model takes into account from the past.
- Time varying reals - numerical features that change over time used to train the model.
- Learning rate - defines the step size at each iteration while moving toward a minimum in a loss function.
- Reduce on plateau patience - defines the number of epochs after which the learning rate will be reduced.

In the proposed scenario, considering some of the main features displayed above, we have performed another round of trials. This time the round of trials was more specific and more time consuming since we needed to grid search among an extremely wide set of hyper-parameters.

4.6.2 Loss Function

Before starting the grid search process we certainly needed to define a proper Loss in order to train the model and evaluate the results. For this purpose, the literature comes in handy, and the most popular loss function used for this kind of models (Recurrent Forecasters) is the Quantile Loss (as explained in the paper [5]).

This loss function calculates different quantile losses for each observation points. Pairs picked from this range can be used to draw percentile bands around the forecast line to indicate the distribution of prediction errors.

Given a prediction y_i^p and an outcome y_i the regression loss for a quantile q is calculated as follows:

$$L(y_i^p, y_i) = \max[q(y_i^p - y_i), (q - 1)(y_i - y_i^p)] \quad (4.2)$$

Different quantiles favour or penalize over-prediction rather than under-prediction. For instance, with a quantile $q = 0.75$, over-predictions will be penalized by a factor of 0.75 and the under-predictions will be penalized by a factor of 0.25.

As soon as we selected the proper loss function, we plugged it inside our model and started with the training process setting the number of epochs for each trial equal to 100.

4.6.3 Fine-tuning Considerations

After performing a massive number of trials for our model we made some evaluations and hypothesis according to the results obtained during the training phase:

- Static reals and categoricals - in order to evaluate the impact of each static feature that we added to the dataset, we analyzed the importance of each variable in the prediction process.

As we can see in figure 4.6, the customer book normalized value that we extracted and fed into the model seems to have a very high relevance throughout the calculation of the predictions.

- Time varying variables - in order to evaluate the impact of each time-varying feature that we fed into the model, we analyzed the importance of those features as well as the static ones.

As we can see in figure 4.7, according to what we expected, the total quantity is the most relevant feature among all (given it represents the target variable). The relative time index certainly impacts a lot as well in the calculations given it defines the time range we are located in.

Among the features that we added to the model, the billing quantity seems to be the most surprising, representing a decent percentage over the total importance, while some other features that have very low importance could be removed from the dataset.

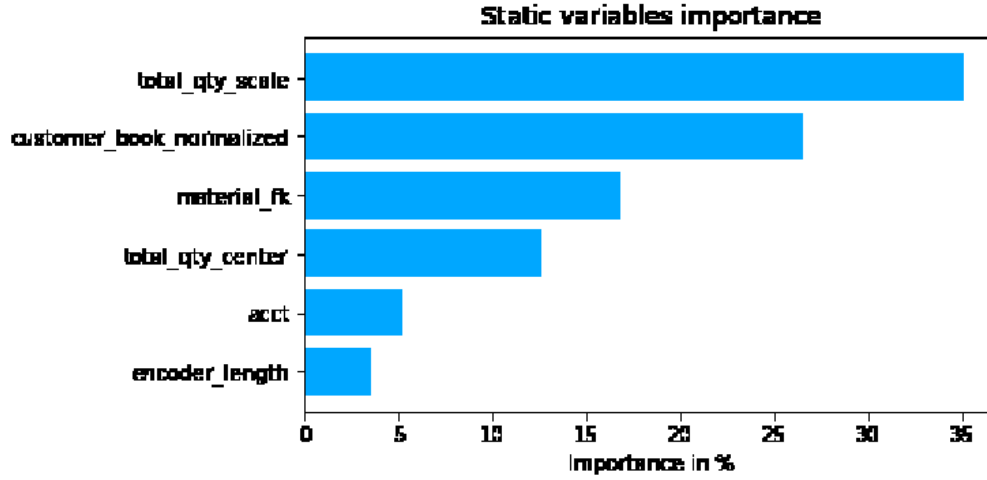


Figure 4.6: Static Variables Importance

- Learning rate - in order to estimate the optimal learning rate to use for the model we used a framework that performs a quick cross-validation in order to spot the best possible value to initialize the learning rate with.
- Encoder length - in order to evaluate the correct encoder length we took a look at the attention levels.
As we can see in figures 4.8 and 4.9 (in the first case using an encoder of length 64 and 132 in the second), we noticed that with a 64 steps encoder the attention was very high in the first time-steps, suggesting that there might be useful information in those time-steps.
In order to evaluate our hypothesis, we extended the encoder value to 132 and noticed that in that case the attention slowly decrease as further we go from the present (time index = 0).
- More complex features (such as Dropout, Hidden size, LSTM layers, ecc..) - in order to evaluate all of these features we necessarily needed to perform cross-validation due to difficulty to make assumptions feature-related. For example, as we can see in figure 4.10, as we compare different values for the Attention Head Size, we are immediately able to recognize that the optimal number of Attention Heads in our scenario is 1, which allows to reach a much lower SMAPE compared to the other trials.
The cross-validation across all the different hyper-parameters allowed us to reach an optimal set of values that we will evaluate in the Results chapter.

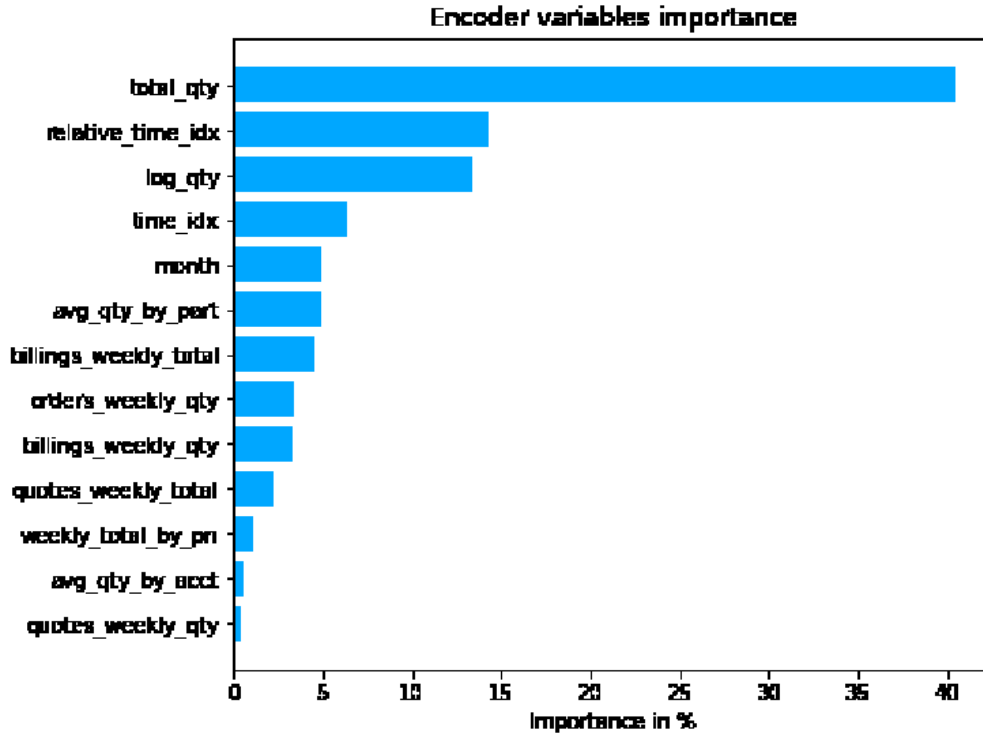


Figure 4.7: Time Varying Variables Importance

4.7 Predictive Implementation

In order to perform the predictions on the open buffers we needed to set up a workflow that provides all the predictions on a weekly basis.

To achieve our objective, we created an SQL table used to store the predictions calculated for each buffer on a weekly basis making them accessible from the dashboard.

In a normal scenario we could avoid to retrain our model every week but given the current world-wide situation caused by COVID and War, the market trend is quite unstable and the semiconductors shortage also heavily impacts on the sales making it necessary to retrain the model more often than usual.

In figure 4.11, we can see how the weekly prediction workflow is implemented. Every step of the process runs on our private cluster and updates all the interested tables accordingly.

As we can see in figure 4.12, for each couple Customer/Material (acct, material fk) several predictions are generated. The field "predictions" represents the quantity

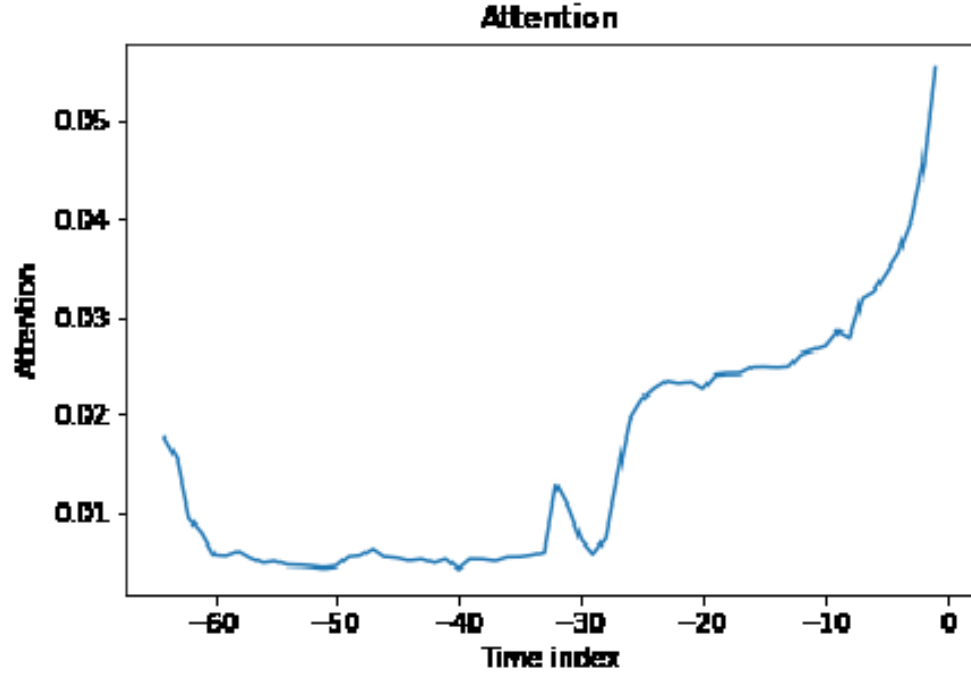


Figure 4.8: Attention 64 steps

directly forecast by the TFT model. The basic predictions then needs to be normalized in order to fit the current status of the buffer (true prediction represents the prediction applied to some business constraints while normalized prediction is calculated from the true prediction through a simple normalization step).

The balancing steps in this case are necessary since we are applying a test model to a real scenario where extremely wrong predictions could be very dangerous for our business.

Future improvements of the model and the data we feed will result in a lower impact on the results coming from the constraints.

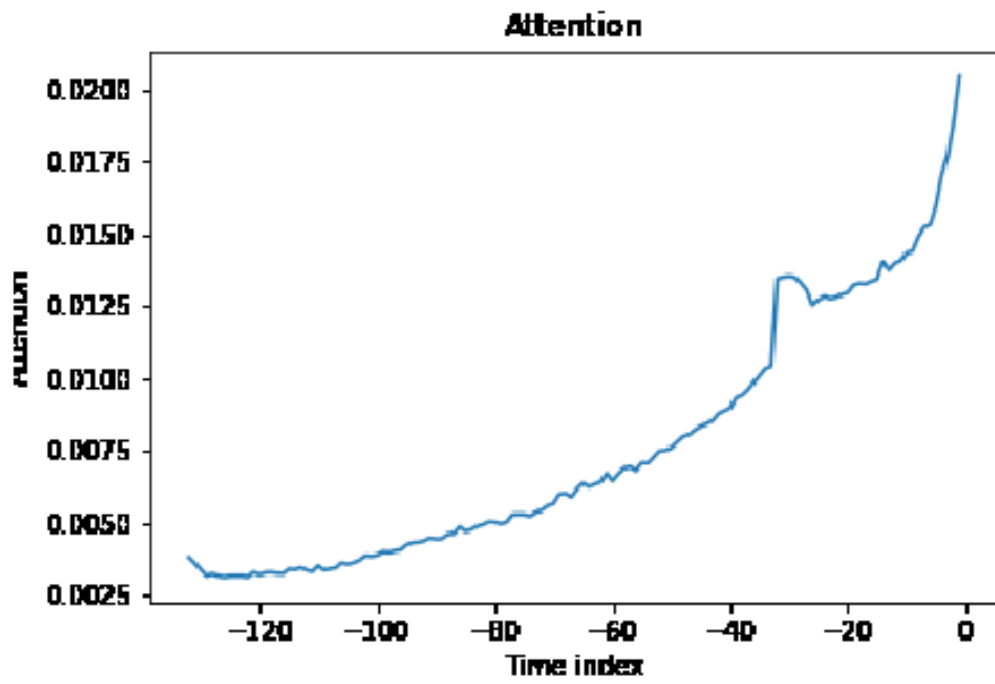


Figure 4.9: Attention 64 steps

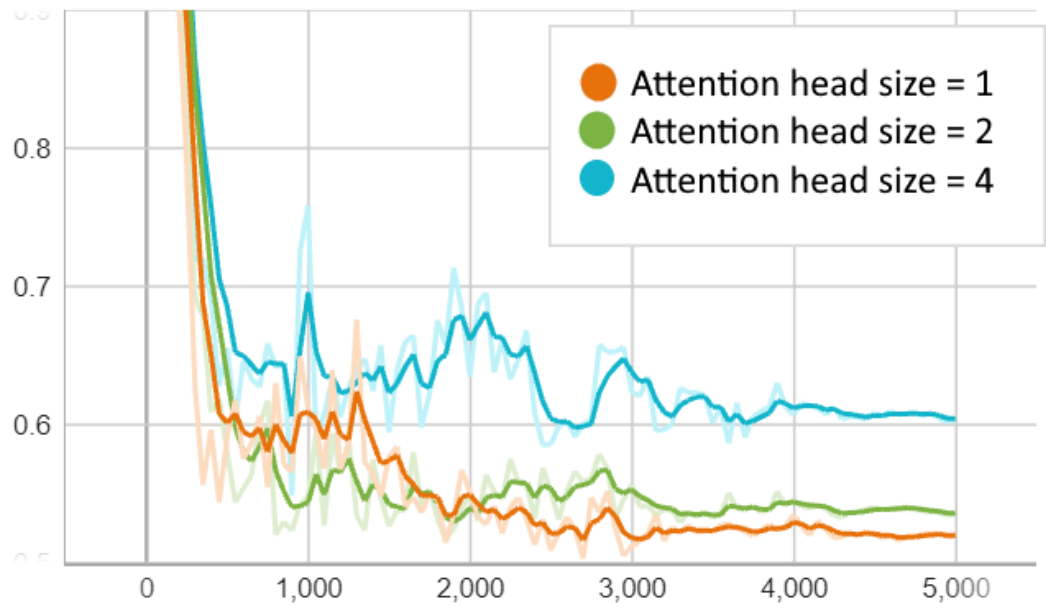


Figure 4.10: SMAPE Comparison for Attention Sizes

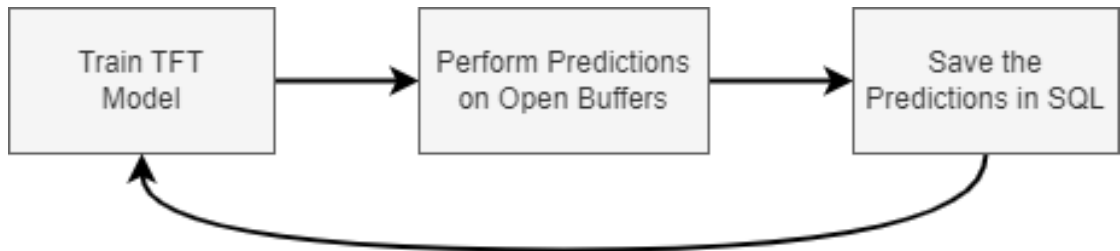


Figure 4.11: Predictions Making Workflow

	material_fk ▲	acct ▲	predictions ▲	true_prediction ▲	normalized_prediction ▲
1	ATTAF120GSTICCE1	0000280867	75	21	21
2	CHMS550HJ1Q02	0000316456	12	12	12
3	CIVFAX06010070016	0000142171	71	45	45
4	CIVFAX08030050008	0000379231	401	234	0
5	CLF1812FS104JLC	0000106342	2437	900	900
6	CLFDO3316P473MLD	0000102397	368	100	100
7	CLFLPS4018474MRC	0000101116	1782	600	600

Figure 4.12: Predictions Table Example

Chapter 5

Results

In this chapter we will evaluate the results obtained with the model trained and optimized in the previous chapter.

First of all, as we can see in figure (which represents only a portion of trials that we performed for the cross-validation), the SMAPE values obtained with the different sets of values for the hyper-parameters range between **0.7** and **0.5** (on the validation set).

The best set of parameters was able to reach a SMAPE of **0.502** therefore we decided to configure our model using the parameters chosen in the corresponding trial (purple line in the figure 5.1):

- Dropout = 0.11
- LSTM Layers = 1
- Hidden Size = 24
- Hidden Continuous Size = 8
- Attention Head Size = 1
- Max Encoder Length = 128
- Static Features = customer value, material type, unit price
- Variable Features = quotes, orders, billings
- Time Variables = month, time index, week
- Output Size = 7
- Learning Rate = 0.03

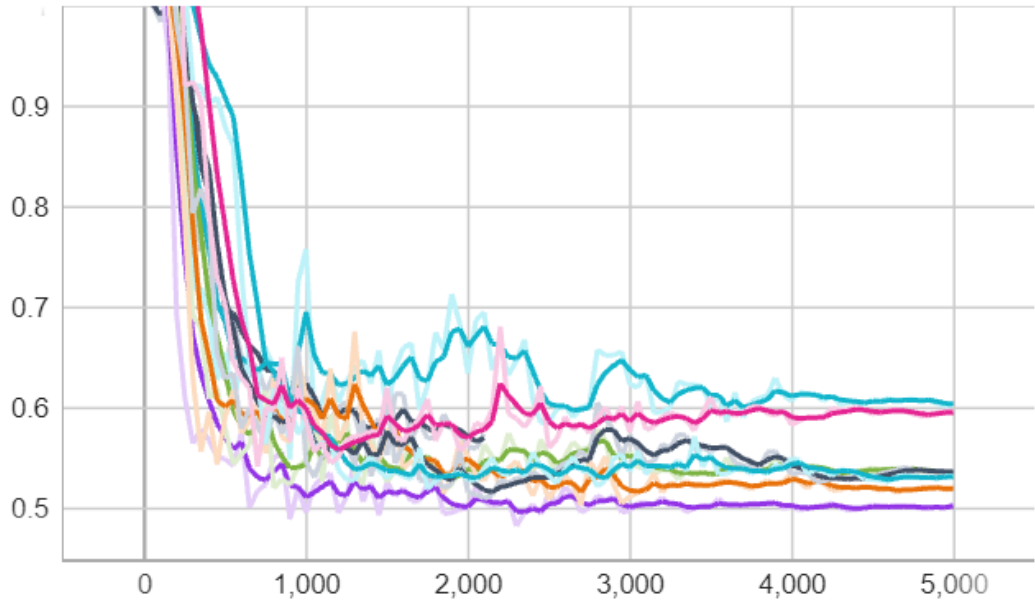


Figure 5.1: SMAPE Cross-Validation

- Epochs = 100

As we configured the model with the parameters mentioned before, we were ready to perform the simulation on our stock in order to evaluate the efficiency of our algorithm.

In order to evaluate the results we performed 2 different tests:

- Current Stock Improvement - in the first case we performed a shallow test where we applied the suggestions proposed by the dashboard to the current open buffers, in this case we noticed that the stock quantity (number of pieces we have in stock) would be reduced by **15.4%** if all the suggestions were accepted.
- In-Depth Historical Analysis - in this case we analysed how our model would have performed in the past 2 years checking also whether we were able to fulfill the requests coming from customers or not.

With this analysis we found out that by accepting all the suggestions for all the buffers in the past two years we would have reduced the impact of our inventory by **30.62%** with an accuracy of **99.42%** (meaning that in the **0.58%** of cases we were not able to fulfill the order request coming from the customer).

There is certainly room for improvement thanks to the data that will be fed into the model for the next few months but the results obtained from the trials previously described are very promising.

Chapter 6

Conclusions

In this work we proposed a dashboard in order to simplify buffers management, and a general overview over the most popular Statistical and Deep Neural models in order to compare their performances on a real case scenario, choose the most promising model and optimize it through the fine-tuning of all of its hyper-parameters.

The final scope of this thesis was to provide a complete tool able to empower user decisions through KPIs and a reliable way to forecast stock quantities in order to reduce the impact of static inventory and minimize inventory management cost.

After a brief introduction to the most popular statical models (ARIMA, Prophet), we provided a detailed overview about all the Deep Models proposed (DeepAR, N-HiTS, TFT) with the help of the most relevant literature about the topic.

After a first introduction to the models, we discussed about the implementation of the project through the descriptive part (dashboard) where we display KPIs and give users a simple way to perform management actions, and a predictive part which consists in the generation of a forecast which represents the quantity that we should store inside a specific buffer.

After illustrating the main features of the descriptive part we wanted to test all the models proposed in order to make a general comparison of their performances over the proposed scenario. The statistical model were consistently outperformed by the Deep Models with shallow grid searching (we did not look for the best hyper-parameters at this stage), guiding us toward the choice of a Deep Model rather than the most popular statistical ones, showing that RNNs and attention mechanisms can exploit much more information about the dataset through the learning of hidden relations between different features and different time-steps.

We then selected the most promising model (Temporal Fusion Transformer) in order to proceed with the fine-tuning process and generate some predictions to evaluate the results. With the TFT we were able to lower the SMAPE from **0.7** provided by the shallow grid search to **0.50** through hyper-parameters optimization.

As we finished fine-tuning our model we were ready to test it and evaluate its

performances, and in order to do so we have proposed an evaluation metric in order to calculate the impact reduction provided by the model.

The results showed that with the adoption of the suggestions provided by the model, the impact would be reduced consistently, resulting in a huge cost reduction for the entire company.

With this work we have highlighted the strength of Deep Models in the domain of time-series forecasting and how these models could be adopted in order to consistently improve the quality of predictions in several scenarios.

We presented a shallow comparison between the proposed models since our main objective was to fully implement a working predictive model in a real case scenario, but the analysis could be extended to several other promising novel models which have recently been developed and the performance comparison could be performed more in-depth in order to better evaluate all the proposed models with an intensive grid search on all the hyper-parameters.

Bibliography

- [1] Sean Taylor and Benjamin Letham. «Forecasting at Scale». In: 20 (Sept. 2017), pp. 7–8 (cit. on p. 6).
- [2] Cristian Challu et al. «N-HiTS: Neural Hierarchical Interpolation for Time Series Forecasting». In: 20 (Sept. 2022), pp. 1–5 (cit. on p. 8).
- [3] David Salinas et al. «DeepAR: Probabilistic Forecasting with Autoregressive Recurrent Networks». In: 20 (Feb. 2019), pp. 1–5 (cit. on p. 10).
- [4] Bryan Lim et al. «Temporal Fusion Transformers for Interpretable Multi-horizon Time Series Forecasting». In: 20 (Sept. 2020) (cit. on pp. 11, 12).
- [5] Ruofeng Wen et al. «A Multi-Horizon Quantile Recurrent Forecaster». In: 20 (June 2018), pp. 1–5 (cit. on p. 36).