POLITECNICO DI TORINO

MASTER's Degree in DATA SCIENCE AND ENGINEERING

MASTER's Degree Thesis

A Data Driven Approach to Remaining Time Prediction of Process Instances

Supervisors

Prof. SILVIA A. CHIUSANO

Candidate

MARCO DI NEPI

JULY 2021

Abstract

Large companies usually keep track of internal processes by continuously updating data in a database in the form of logs. They are crucial to carry out conformance checks and monitor whether a case is progressing as expected and similarly to what has happened in the past or, in alternative, detect any errors or unexpected loops that can negatively affect the performances of a system. Predictive process monitoring collects a set of techniques and methodologies to analyze event logs, with the purpose of making predictions on running cases. Being able to predict in real time the remaining time until the completion of a case is crucial to allow the user to intervene promptly. A fast response guarantees a reduction in the risk of delays and slowdowns in the entire workflow, which may occur in any moment, and an increased awareness on the presence of behaviors that differ from the normal trend. The problem can be treated as a supervised learning task and in this paper we propose a methodology based on neural network models. In particular, given the log structure as an ordered sequence of events, it comes natural to exploit architectures able to manage data sequences and very long dependencies, such as recurrent and attentions-based architectures. The goal is to integrate and optimize the application maintenance service provided by the company through machine learning algorithms. The case study, the forecasting of job completion time in an HPC system, covered the entire process from data acquisition to model deployment and the development of a dedicated web application to provide, in addition to the prediction, other useful features for improving the system.

Acknowledgements

Vorrei esprimere la mia più sincera gratitudine alle persone che mi hanno accompagnato in questo capitolo della mia vita offrendomi supporto in ogni situazione.

Ringrazio Reply, e in particolare Roberta, Monica e Sara per l'aiuto fornitomi in questo progetto e per avermi fatto sentire fin da subito parte del team.
Un rigraziamento alla mia relatrice Prof.ssa Anna Chiusano per i suoi preziosi consigli e al suo contributo per la riuscita di questo lavoro.

Vorrei ringraziare i miei genitori, mio fratello Edoardo, e tutta la mia famiglia per la presenza constante e per tutto il sostegno che hanno sempre rivolto nei miei confronti.

Un enorme grazie ai miei amici di sempre, al gruppo Doge, ai miei vecchi compagni di merenda Ruben e Jacob, ai data scienziati del Cluster con cui ho condiviso questo percorso e a tutti coloro che, tra Roma e Torino, non hanno mai smesso di starmi vicino.

Table of Contents

List of Tables

List of Figures

Acronyms

BPM
Business Process Monitoring

HPC
High Performance Computing

RNN
Recurrent Neural Network

LSTM
Long short-term memory

RSS
Residual sum of squares

GRU
Gradient Recurrent Unit

ETL
Extract Transform and Load

MAE
Mean Absolute Error

MSE
Mean Squared Error

NLP
Natural Language Processing

Chapter 1

Introduction

In modern era, companies regularly store in their database systems details about all the processes executed in the past. These data, which are called *logs*, are kept not only to preserve historical information, but are mostly used at run-time to check that ongoing traces are keeping on regularly and are not affected by problems. They are crucial indeed to carry out conformance checks and monitor whether a case is progressing as expected and similarly to what has happened in the past or, in alternative, detect any errors or unexpected loops that can negatively affect the performance of a system.

Processes are made up of an ordered sequence of tasks, each characterized by its execution timestamp. It is easy to think, for example, of a shipping company that has to keep track of the status of all orders in progress. A trivial sequence of tasks could be: 1) Preparing, 2) Sent, 3) Delivering, 4) Delivered. Each of these operations is then associated with a time indication. At this point, the customer will want to have a real time estimate of how long after the order will be delivered to him. This is just one of the possible applications of *process mining*, a discipline which aims to create interpretable models of processes and exploit them to obtain benefits in terms of insights, forecasting capacity and system optimizations.

The increasing popularity of machine learning and deep learning techniques has given a further boost to this field. Neural Networks, at the expense of the loss of explainability due to the black box approach, have provided new state-of-the-art results, achieving a far better accuracy than traditional methods.
In particular, in the last years, several papers have been published on three main tasks: Next Activity Prediction, Event Time Prediction and Remaining Time Prediction. This thesis will focus on the last challenge and will analyze and apply some machine learning algorithms to a dataset based on sequences of software processes that run daily on a high-performance machine.

The project has been developed with the support and collaboration of *Technology Reply*, one of the companies of the Reply group. Reply is an Italian company based on a network model, it is made up of a large number of companies, each operating on specific businesses and headed by a central holding. In particular, Technology is specialized in the development of innovative solutions based on Oracle technology, and it mainly deals with data infrastructures, databases and data modeling, also offering solutions based on the latest big data and machine learning technologies.

Among the activities of the company, an important collaboration with a large automotive company firm stands out. Technology Reply has developed a multitude of data pipelines and job sequences used to feed reports for the client. The latter are used every day by operating directors or engineers to understand the trend of the business, to organize daily activities and make evaluations.
Given the importance of having these data available with the right timing, all information related to the monitoring of these loading processes is constantly collected in a database in the form of logs, in order to keep track of delays and problems that may occur in any moment.
Subsequently, the need arose to provide the customer with a real-time estimate of the remaining time of these processes, as well as details and statistics regarding the impact of scheduling times, days of the week and the amount of data processed.

This thesis tries to fit into the context of process mining and in particular of business process monitoring, seeking to apply machine learning models in a business environment and attempting to integrate these advanced techniques into the standard workflow. The dialogue with the domain experts, who were able to provide explanations on the functioning of the processes, was thereby fundamental to the success of the work.

The objectives proposed by this project can be summarized in:

- Providing a literature review and studying the most recent machine learning and deep learning technologies applied to the remaining time prediction problem. After that, building and testing models and providing a benchmark of the performance of the various methods.
- Building an efficient pipeline to process and store event log data through the development of new tables and views in the database. Then proceed with cleaning and processing the data in order to make them suitable for use in machine learning tasks.
- Providing an accurate data analysis in order to obtain statistics and insights useful for improving the system and the service provided by the company.

The document is divided in several chapters and is organized as follows:

Chapter 2 provides an introduction of the goals and main tasks of process mining and predictive process monitoring. Moreover, a formal statement of the remaining time prediction problem is given together with an analysis of the state of art and recent publications.

Chapter 3 describes the theoretical background of machine learning and deep learning models used and is divided in three main sections: tree based algorithms, recurrent neural networks and transformers.

Chapter 4 explains the main features of the dataset used in the experiments. Also, is given an overview of the data acquisition and data cleaning phase. Finally, the results of a brief exploratory data analysis are presented.

Chapter 5 describes the proposed approach for remaining time prediction.
It is provided a report on the evaluation method, data preprocessing, developed architectures and training phase. The second part of the chapter is dedicated to the outline of the framework and how the prediction is inserted in the enterprise context together with additional functionalities such as anomaly detection and the construction of confidence intervals. Further space is given to the model deployment and release of the application.

Chapter 6 finally presents the results of the experiments and the performance of the tested models according to the metrics used.

Chapter 2

Process Monitoring

2.1 Process Mining

Process mining is a discipline that is proposed as a meeting point between data science and business process management (BPM). If the second is mainly oriented towards the modeling and optimization of processes within a company, data science allows for a more data-driven approach aimed at obtaining both powerful insights and predictions tools. Process mining analyzes business processes through logs, a popular kind of data indicating the status and progress of an ongoing activity providing above all a temporal indication of what is happening at a precise moment.

The starting building block of process mining is the *Event Log* or *Process Log*. This object contains information about *Cases*, single instances of the considered process, and a sequence of *activities* within each case.
The elements of an event log table must necessarily satisfy three fundamental requirements:

- Each event must be associated to a referring timestamp indicating the beginning or the end of the activity. This is a primary information when the measure of interest is the remaining time of a running case or the analysis of the performances of the system.

- Events within a case are ordered, there is a precise temporal sequence of what happened in the past and what will happen in the future. A disordered sequence would make the whole analysis lose its meaning.

- Events may have attributes or features, as instance the number of resources used, the priority, the cost, or many others.

Case-ID	Activity	Timestamp	NRows
101290	Loading Data	30-01-2020:11.02	12030
101290	Transformation-01	30-01-2020:11.25	12030
101290	Case Ended	30-01-2020:11.38	-
101291	Loading Data	31-01-2020:00.15	4
101291	Ended in Error	31-01-2020:03.19	-

Table 2.1: Trivial example of an event log

The previous concepts can be defined in a more formal way:

Definition 1 *(Event) An event is a tuple* $(c, a, t, (v_1, v_2, ..., v_m))$ *where c is the case identifier, a is the activity name, t is the timestamp and the* v_i *(where* $m \geq 0$*) are the values of the attributes.*

Definition 2 *(Trace) Let E be the event universe, the set of all possible events in the process, and let c be a case. A trace is a sequence* $\sigma_i =< e_1, e_2, ..e_m >$ *of events, such that* $\forall i \in [1, n], e_i \in E$ *and* $\forall i, j \in [1, n]$ *if* $e_i \in c$ *then* $e_j \in c$

Definition 3 *A trace is completed when each event in the trace has been completed and fully handled in the past. If a trace is still ongoing, it is called partial trace*

Process mining has applications in countless industries or companies that rely on log data to monitor the status of the internal processes. Examples of processes in which this type of data can be found are hospitals, following all the activities within the whole treatment path of a patient, such as the transition from booking to visit and final control with the doctor, or the management of a ticketing service considering information as open, in progress, pending or closed, or even more, as in the case of this project, a sequence of jobs concerning ETL operations running on a computer system.
In literature, three main types of analysis that can be applied to event logs are defined:

- *Discovering*: These techniques are used to build a model exploiting only the data logs. The result is typically a model in the form of a Petri net, as in the case of the $\alpha - algorithm$. Petri nets are a particular type of oriented graph, composed of places, transitions and arches, used to represent a discrete distributed system. Once the output is ready, it can be used to retrieve useful insights.

- *Conformance*: The goal of this type of process mining is to compare an event log to another of the same type or to an existing model in order to check the differences and reveal the presence of unusual behaviours or deviations with respect to the sequence of reference. This approach can be exploited in many applications, for example to build a complex rule based system.
- *Enhancement*: The last group of techniques includes a set of tools to improve and modify the extracted model, extending it with new event logs and finding, as instance, bottlenecks, correlations or performance indicators to make predictions. An example of enhancement is the repairing, that is the update of a model to better suit existing logs.

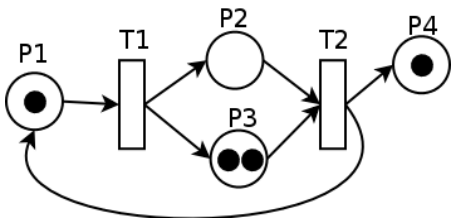

Figure 2.1: Petri Net example [1]

This thesis will elaborate deeper the last category, enhancement, and in particular the subfield known in literature as predictive process monitoring, concerned about forecasting and predictions on the ongoing cases. The modeling component is therefore lost to the benefit of the individual instances, on which analysis and statistics can be made. Thanks to Deep Learning, in addition to the classical machine learning techniques, this field has gained a lot of attention in the last years, and many challenges as well as many different approaches have been studied.

Besides the remaining time prediction, which will be discussed in more detail shortly, the most common problems that predictive process monitoring tries to solve [3] are all related to some aspects of a partial trace. The prediction of next activity of a running case, when the order of the events is not fixed and there exist different paths with different outcomes, can certainly be an interesting task, especially in cases where there are complex systems such as logistics or the management of a ticketing system. A variant of this last task is the prediction of the entire suffix of the case, but it is a more advanced problem and good results are difficult to be obtained. In the same way, the next timestamp prediction, that is the ending time of the following activity, can be crucial in a system strongly

[1]Source: https://upload.wikimedia.org/wikipedia/commons/f/fe/Detailed_petri_net.png

based on timing optimization and is widely studied in literature. Even more, it is also possible to define a classification problem regarding the outcome of a trace based on the sequence of the logs, for example accepted or rejected in the case of an online request. In general, the goal is to identify the processes which create problems and slowdowns and which can also become expensive to maintain.

A further problem lies in the fact that, since the prediction must be done online and give results in the shortest possible time, the model must be ready and fast in the inference and able to analyze even large amounts of logs in a short time to reduce risks. A static analysis, based only on historical data, would not be sufficient and would not allow humans to intervene promptly.

2.2 Remaining time prediction

When dealing with processes in a dynamic system, it is essential to be able to accurately estimate at what time of day the results will be available. A generic prediction based solely on past day averages, as instance providing only a range of the typical execution time, is usually not accurate enough. On the contrary, it is necessary to take into consideration many variables: On the one hand, there is the sequence of events, which gives a clear indication of how a process is taking place. On the other hand, the attributes of the case and features previously mentioned, which cannot be ignored, also play an important role.

Let $e_i = (c_i, a_i, t_i, v_i)$ be an event and consider an operator π that extract an element of the tuple in a specific event: As instance $\pi_a(e_i) = a_i$ $\pi_c(e_i) = c_i$ and so on. Given a trace $\sigma =< e_1, ..., e_n >$, the total execution of the case would be the difference between the last recorded timestamp, and the timestamp associated to the first event. Formally: $et(\sigma) = \pi_t(e_n) - \pi_t(e_1)$.

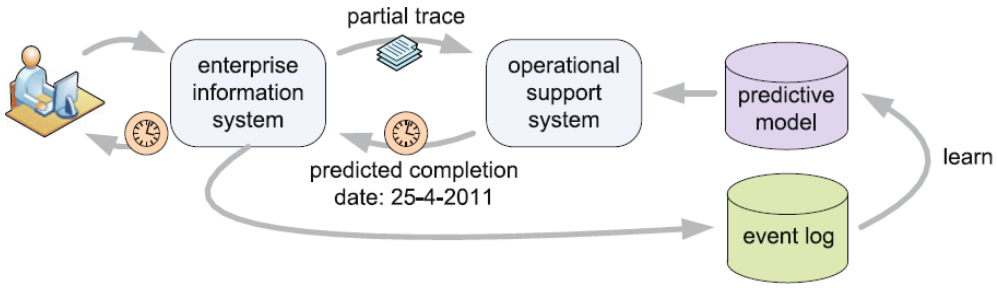

Figure 2.2: Scheme of training and inference phases[2]

Definition 4 *(Head and Tail) Given a trace σ, the head operator of length k $hd^k(\sigma)$ is defined as the sequence of the first k events of the trace such that $hd^k(\sigma) =< e_1, ..., e_k >$. The tail operator $tail^k(\sigma)$ defines the last k events in the trace: $tail^k(\sigma) =< e_{n-k+1}, ..., e_n >$*

We can now consider a partial trace, that is a trace that has started but is not ended yet $e^* =< e_1, ..., e_k >$. Our goal will be to predict the remaining time of the case, that is the difference between the timestamp of the event e_k and the one of the unknown event e_n.

Definition 5 *(Remaining time of a running case) Given a trace σ, the remaining time of a running case is the difference between the the execution time of the complete trace and the time that has passed since the first event. $et(\sigma) - et(hd^k(\sigma)) = et(tail^{n-k}(\sigma))$.*

This information has a high value for a twofold reason. On one hand, it allows the customer to know how much he should still wait, on the other hand, managers and engineers can look for delays or unexpected behaviors and try to react in time in order to avoid low performances of their systems. Note that we are not interested in predicting what the next activity will be, but only measuring the time until the final completion.

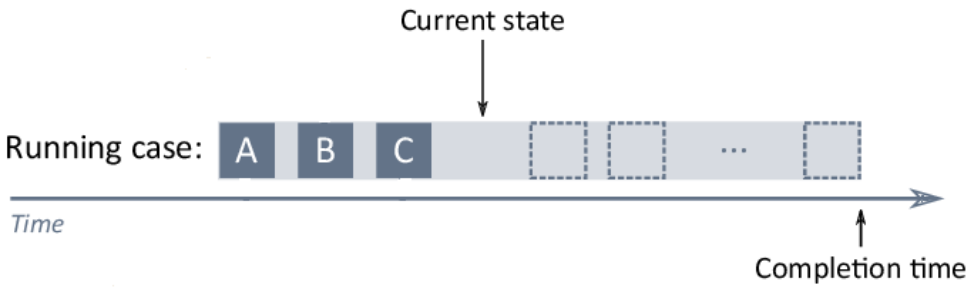

Figure 2.3: Remaining time prediction [4]

There exist various traditional methods to generate predictions, for example the ones based on Petri Nets or annotated transition systems and extraction of explicit

[2]©Process Mining, Data Science in action. Wil van der Aalst 2011,2016

models for the logs. However, the remaining time prediction task can be defined as a regression problem, hence a supervised learning task, having a well defined target variable. The results, as a consequence, are evaluated through well known metrics like Mean Absolute Error and Mean Squared Error. Considering the natural sequential structure of traces, the use of neural networks is worth investigating. Natural Language Processing, in particular, makes heavy use of recurrent neural networks, such as long-short term memory networks, but also new transformer-based architectures are also receiving a lot of attention in recent times. In the next section, some of the papers published in recent years will be illustrated in depth.

2.3 Related Works

Several authors tried to address the problem of remaining time prediction. One of the first contributions is from Wil Van der Aalst et al.[5]. This paper from 2008 shows how to make a prediction using a non-parametric regression, taking into account the duration of all the activities within a case together with other case-related data. Differently, Polato et al. in [4] proposed an explicit model for the process given by a transition system based on log abstractions. The prediction is computed joining a regression model combined with the likelihood of the next events given the data until a certain moment. Updates to overcome the limitation of transition systems are presented in [6] and [7].

Interestingly, Guo et al in [8] analyzed jobs logs for a HPC (high performance computing) system using standard machine learning algorithms such as random forest and boosting techniques, with a particular focus on the prediction of jobs whose duration has been underestimated and which will then be terminated automatically by the machine. A similar approach was followed by Di Francescomarino and Senderovich in [9], which also made use of inter-case features. Their intuition was mainly based on the fact that cases run concurrently and are not independent. As a consequence, they introduce additional features such as the case type, in order to categorize cases with similar behaviors, and the number of cases currently running in parallel.

The great breakthrough in the state of the art came with the introduction of deep learning techniques. Inspired by successes achieved in the fields of time series analysis and computer vision, recurrent neural networks have seen their popularity grow rapidly to establish themselves as the new standard for predictive monitoring. The following papers have made experiments on the BPI challenge dataset, a set of real life log form different domains used to test process mining and machine learning approaches for time based and event based predictions.

Among the different alternatives, remarkable is the work of Tax [10], which exploited LSTM not only for the remaining time prediction but also for suffix and next activity prediction through a multi-layer multi-task architecture. Similarly, the model proposed by Camargo et al in [11] is trained to predict the execution time of each event separately and then joins them together to predict the total remaining time. Although the results obtained with this procedure are satisfactory, they have a strong limitation with cases that have repeated attributes and seem to be worse than those obtained by Navarin in [12]. The main difference between this last paper and the previous ones is that the network is trained to directly predict the remaining time of a case, thus avoiding accumulating errors at each step and achieving a lower mean absolute error.

Considering the architecture used, in [13] have also been used the memory augmented neural networks, which exploit an external memory unit to overcome the limitations of LSTM and better remember long dependencies. Notwithstanding the advantages, these networks are typically hard to train and slow. Other solutions have been explored by Pasquadibisceglie in [14], that proposed a set of operations to convert historical log to spatial data and threat them as images in order to make predictions with a convolutional neural network. Taymouri et al. [15] adapted a Generative Adversarial Networks with an encoder-decoder architecture to generate a sequence of events and related timestamps. Finally, Bukhsh et al. in [16] developed a transformer, a purely attention based network, for remaining time prediction and next activity prediction.

As far as the encoding of the variables is concerned, a machine learning algorithm must be trained with a fixed-size input vector. There exist in literature two main strategies. The first is the one-hot encoding, used in [12] and [13], which consists in representing the feature as a binary vector of length equal to the number of possible values for that feature. A possible alternative is a more advanced entity embeddings[17], as proposed by [11], [18], [19], where categorical attributes are mapped into an Euclidean space whose dimensionality becomes a new hyperparameter of the model. The mapping can be learned with back propagation together with the rest of the network, with the advantage of reducing memory use and speeding up the training phase. Moreover, the learned embeddings have been shown to be able to extract the semantic meaning of words, mapping attributes that have a similar meaning close in the space.

The works mentioned above are also distinguished by how they manage the sequences: LSTM and RNN networks are conventionally trained with prefixed padded, hence for each trace every possible set of prefixes is considered and the resulting

vector is padded with zeros when its length is shorter than the maximum possible length of a case ([12], [16], [10], [13]). [20] proposed instead a timed state encoding that represents the inner state of an existing model, like a Petri Net, after being applied to a partial trace. [18], in alternative, takes into account only a single event at time, obtaining a simpler architecture but losing the dependencies within a case. Another encoding commonly used in language processing and adapted in this context in [21] consists in treating the event logs as if they were text and using a sliding window to extract events even from different cases. This allows for a faster training but long dependencies are not properly recognized.

Chapter 3

Machine Learning Algorithms

This chapter will provide the main theoretical background behind the use of some popular machine learning algorithms. Starting from tree based methods, we will then move on to the deepening of recurrent neural networks. Finally, transfomers will be introduced.

3.1 Tree based algorithms

3.1.1 Random Forest

Random forest is a simple supervised learning algorithm commonly used for both classification and regression tasks. The procedure, proposed for the first time by Ho in 1995 [25], was then formalized in the paper [26] written by Breiman. The algorithm is based on a ensemble of decision trees, trained with the *bagging*, or *bootstrap aggregation*, technique.
Decision trees model a set of sequential and hierarchical decision rules so that the predictor space is divided in non-overlapping regions (boxes or rectangles). If the relationship between dependent and independent variables is not linear, they outperform methods as Linear Regression.

The main intuition behind bagging is that a learning method reduces its variance if the prediction is made taking the average of n different models trained separately on different datasets. In fact, considering n independent observation of a population, the variance of the sampling distribution is computed as $\sigma_m = \sigma/n$ where σ is the variance of the population. Since getting n different datasets is not trivial, what is done in practice, instead, consists in generating B bootstrapped

datasets from the original one, using a resampling with replacement strategy. A rule of thumb is taking $\frac{2}{3}$ of data, leaving out the remaining part. The final result can be computed taking the average of the B models as follows:

$$f(x) = \frac{1}{B} \sum_{i=1}^{B} f_i(x) \tag{3.1}$$

The remaining $\frac{1}{3}$ of data that is not used to train a specific tree is commonly defined as *out-of-bag*. These data can be used to make an estimate of the error: For each sample, we can compute the response of each tree for which that sample has not been used yet and average them. This would be a good estimate and would likely replace the use of a validation set or a more expensive cross validation.

In order to obtain a good reduction of the variance, it is also needed to avoid high correlation between trees: when a split in a certain tree is considered, only few of the predictors are taken into account. In other words, if there exist m predictors, at each split p of them are randomly chosen, typically $\sqrt{m}$. With this simple trick, the trees become different from each other and, as a consequence, the performances are improved.

Another advantage of tree-based algorithms, in addition to their easiness of use, is the interpretability: in fact, it is trivial to directly extract the rules to partition the space (if the tree is not too deep). Features that are near the root of the tree are the ones that contribute most to the division of the dataset. On the other hand, an ensemble of trees sacrifices part of the interpretability for better accuracy. However, it is still possible to obtain a global estimate of the importance of the features for the regression (or classification) task computing for each predictor the corresponding decreasing of residual sum of squares (RSS) averaged over every tree in the forest according to the following formula:

$$RSS = \sum_{j=1}^{J} \sum_{i \in R_j} y_i - \hat{y}_j \tag{3.2}$$

where the R_i are the high-dimensional rectangles in the feature space and $\hat{y}_j$ is the mean response of the i-th rectangle.

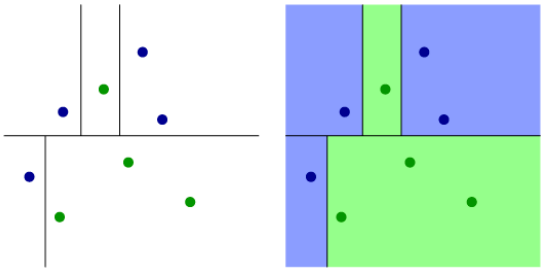

Figure 3.1: Decision boundaries of a decision tree

Despite its simplicity, the random forest, thanks to its ability to be robust with respect to outliers and the risk of overfitting, is widely used in practice. However, due to the slowness of the training phase and the difficulty of exploiting it with linear data, it is not suitable to be used in every case.

3.1.2 Boosting

A worthy of notice alternative to bagging is boosting. This second approach, however, is not based on the training of a certain number of independent decision trees with bootstrapped datasets. On the contrary, trees are fitted sequentially so that each tree learns from the mistakes of the previous ones, improving their performances where they do not achieve good results.

The trees resulting from this process typically have low complexity and a limited number of nodes. The training phase is therefore rather slow but the results obtained are often better than those of a random forest.
The training algorithm of a tree based model that exploits boosting is composed as follows:

1. Initialize $\hat{f}(x) = 0$ and $r_i = y_i \ \forall i \in Training\text{-}Set$
2. Set B as the number of the trees
3. For b in range (0,B):
   (a) train a tree f_b with d splits on the dataset using r as response
   (b) Compute: $\hat{f(x)} = \hat{f(x)} + \lambda f_b(x)$
   (c) Compute the new residuals: $r_i = r_i - \lambda f_b(x)$
4. The final model is given by $\sum_{b=1}^{B} \lambda f_b(x)$

Is easy to notice that residuals are used as response instead of the labels (or outcomes) y_i of the dataset. The result is again an ensemble model but the function

is built gradually with small updates.

Wrapping up, the main hyperparameters that must be tuned are:

- The number of trees B. Compared to random forests, boosted models are more prone to overfitting and therefore it is a crucial parameter
- The number of splits in a tree d, that controls their complexity shapes. It is not uncommon to set it to 1. In this way the f_b only takes into account a single variable (stump).
- The shrinkage parameter λ, that controls the learning rate of the model: if it is set too small the training would be slower and would require an higher B, if it is set too big the model would fit faster but the generalization error will be higher.

3.1.3 XGBoost

A boosting application that has met with considerable success and excellent results is eXtreme Gradient Boosting [27], known in literature as XGBoost. The authors proposed a solution that is able to scale to billions of samples thanks to several optimizations that make it extremely efficient and also suitable to be used in parallel on multiple machines to greatly speed up the training.
The algorithm solves an optimization problem minimizing:

$$\begin{gathered} \zeta(\phi) = \sum_i l(y_i, \hat{y}_i) + \sum_k \Omega(f_k) \\ \text{where } \Omega(f) = \gamma T + \frac{1}{2}\lambda ||\omega^2|| \end{gathered} \tag{3.3}$$

This object represents the objective function of the problem. The term l is the considered loss function that measures the 'distance' between the predicted value and the observed value. Ω is an additional regularization term that depends on T, the number of leaves in a tree, and the leaf weights (output) ω .
Reminding that a boosting model is trained incrementally, at each step the prediction is updated with the contribution of one additional tree. The equation becomes:

$$\zeta^{(t)} = \sum_{i=1}^{n} l(y_i, \hat{y}_i^{t-1} + f_t(x_i)) + \Omega(f_t) \tag{3.4}$$

In other words, at each step t, it is added the f_t that most improve the model according to a greedy approach. In order to exploit optimization techniques in

euclidean space, it is necessary to transform the objective function: with a Taylor expansion to the second order, f can be approximated to a simpler linear form.

$$\zeta^{(t)} \approx \sum_{i=1}^{n} [l(y_i, \hat{y}_i^{t-1}) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)] + \Omega(f_t) \tag{3.5}$$

Where g_i and h_i are the first and second derivative of the loss respectively. By removing the constant term we finally obtain the simplified objective function, that will be the new starting point:

$$\tilde{\zeta}^{(t)} = \sum_{i=1}^{n} [g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)] + \Omega(f_t) \tag{3.6}$$

The last object can be rewritten expanding Ω and computing the products. Defining $I_j = \{i : q(x_i) = j\}$, it becomes:

$$\tilde{\zeta}^{(t)} = \sum_{j=1}^{T} [(\sum_{i \in I_j} g_i) w_j + \frac{1}{2} (\sum_{i \in I_j} h_i + \lambda) w_j^2] + \gamma T \tag{3.7}$$

Finally, by computing the fist derivative, it is possible to obtaining the optimal weight for the leaf j with:

$$w_j^* = -\frac{\sum_{i \in I_j} g_i}{\sum_{i \in I_j} h_i + \lambda} \tag{3.8}$$

And by replacing it in (3.6) we can obtain the optimal value for tree q:

$$\tilde{\zeta}^{(t)}(q) = -\frac{1}{2} \sum_{j=1}^{T} \frac{(\sum_{i \in I_j} g_i)^2}{\sum_{i \in I_j} h_i + \lambda} + \gamma T \tag{3.9}$$

The formally derived formula cannot be used easily in practice since it is not possible to obtain all possible trees. Instead, the best split candidates are derived with:

$$\zeta_{split} = \frac{1}{2} \left[\frac{(\sum_{i \in I_L} g_i)^2}{\sum_{i \in I_L} h_i + \lambda} + \frac{(\sum_{i \in I_R} g_i)^2}{\sum_{i \in I_r} h_i + \lambda} - \frac{(\sum_{i \in I} g_i)^2}{\sum_{i \in I} h_i + \lambda} \right] - \gamma \tag{3.10}$$

Where I_l and I_r are the left and right nodes after the split and $I = I_l \bigcup I_r$. This means that the procedure is iterative and it only considers one node ad time, adding greedily a new branch.

As before, in addition to the split finding algorithm, XGBoost also takes advantage of several methods to reduce overfitting: Shrinkage and feature subsampling (similarly to random forest).

We can now consider a setting where data is too big to be fitted into memory, or it is distributed. In these cases, the proposed algorithm cannot be proposed as it is and an approximated version of the procedure must be exploited. In fact, instead of testing every single threshold, data can be divided in *quantiles* that may be used as candidates. In practise, in order to make it scalable and distributed, it is exploited an algorithm called *Quantile Sketch*, which is able to merge data coming from different sources to build an approximate histogram over which calculate the quantiles. More formally, we can define a rank function r that measure the proportion of samples whose feature k has value below a threshold.

$$r_k(z) = \frac{\sum_{(x,h)\in D_k, x<z} h}{\sum_{(x,h)\in D_k} h} \tag{3.11}$$

The goal is to find splits $\{s_{k1}, ..., s_{kl}\}$ that satisfy the following property:

$$|r_k(s_{kj}) - r_k(s_{k,j+1})| < \epsilon \text{ and } s_{k1} = min_i(x_k),\ s_{kl} = max_i(x_k) \tag{3.12}$$

Finally, the last optimizations implemented are a Sparsity-aware split finding, which makes the algorithm choose the best default path for missing data depending on the loss of the training data, and a compression-decompression method to reduce the usage of the hard drive and maximize the use of cache.

In conclusion, XGBoost has proven to be one of the best machine learning algorithms for structured data, both in terms of accuracy, training time and scalability. However, it lost to neural networks in tasks like time series forecasting, not excelling in managing long sequence dependencies or spotting trends.

3.2 Recurrent Neural Networks

Among the most interesting problems that can be solved through the use of machine learning are those using sequential data. Classical examples are speech recognition, time series analysis, natural language processing or video processing. Traditional neural networks architectures are not able to handle sequential data well, since they cannot model long sequences dependencies but instead treat each sample independently. Feedforward neural networks are then called *stateless*. At this point, it becomes necessary to use a new architecture that allows information to persist over time: Recurrent Neural Network (RNN).

The intuition is that the network has an internal state that is updated at each time step through a parameterized function of the old state and the current input: $h_t = f_W(h_{t-1}, x_t)$.

During the training process the weights W are learned through backpropagation. It is important to notice that the same matrix is used at every time step.

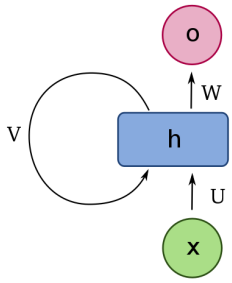

Figure 3.2: Basic structure of a recurrent neural network

A straightforward way to implement this behaviour is to use two separate weight matrices for the input x_t and for the state h_{t-1} and then apply a nonlinearity to the sum of the two contributions.

$$h_t = tanh(W_{hh}^T h_{t-1} + W_{xh}^T x_t) \tag{3.13}$$

The output vector instead, is again computed from the new resulted state:

$$\hat{o}_t = W_{ht}^T h_t \tag{3.14}$$

Internally an RNN can be represented as a sequence of cells, each producing an output and passing a message to the next cell based on the internal state. The final loss is then the sum of the loss of the single cells.

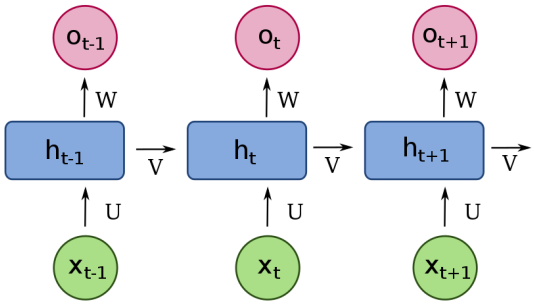

Figure 3.3: Enrolled graph of a RNN

However, in order to minimize the loss, errors are back propagated at each individual time step and across all the time steps to the beginning of the sequence.

Remembering that the cell update results from a non linearity and a matrix multiplication, the gradient computation requires many repeated derivatives [28]. For instance:

$$L = \sum_{t=1}^{T} l(y_t, o_t)$$

$$\frac{\partial L}{\partial W} = \sum_{t=1}^{T} \frac{\partial l(y_t, o_t)}{\partial W} = \sum_{t=1}^{T} \frac{\partial l(y_t, o_t)}{\partial o_t} \frac{\partial o_t}{\partial h_t} \frac{\partial h_t}{\partial W}$$

While the first two terms can be easy to compute, the third one can be trickier, since h_t depends on W and h_{t-1}.

$$\frac{\partial h_t}{\partial W} = \frac{f_W(h_{t-1}, x_t)}{\partial W} + \frac{f_W(h_{t-1}, x_t)}{\partial h_{t-1}} \frac{\partial h_{t-1}}{\partial W}$$

The final result comes from the aggregation of the gradients of the whole sequence with respect to the weight matrix.
At this point, two bad scenarios can happen:

- Exploding gradients: the involved values become too large and impossible to be optimized. Large errors lead to large updates and less stable, or not convergent, models
- Vanishing gradients: the values get smaller and smaller, losing the contribution of the first states and making the training extremely difficult.

These issues could be partially solved using different activation functions, for instance ReLU, and parameter initialization. Moreover, new architectures with more complex recurrent units have been proposed.

3.2.1 Long Short-Term Memory Networks

Proposed by Hochreiter and Schmidhuber in 1997, nowadays LSTM networks[29] are widely used in many applications. The key building block of LSTM is a structure called *gate*. Gates enable the network to better control information in within its cell, selectively adding or removing it when needed. Each gate is composed by a layer (e.g sigmoid or tanh) and a point-wise multiplication.
The computations of each unit are typically divided in four steps:

1. Forget: Decide what is the relevant information and remove what is not worth to be remembered through a sigmoid function, that puts to zero what is forgettable.

2. Input: Select the relevant part of new information that will be stored into the new cell state.

3. Update: Modify the internal state using both the relevant parts of current information and current input

4. Output: select which part of the information should be sent to the next cell and exposed to the outside.

The computations performed by the gates are reported below. At each time step there are two states, the hidden state h_t, similar to what the vanilla RNN does, and the cell state c_t. Only the first is exposed, while the other is kept hidden and only flows from one cell to the next one.

$$f = \sigma(W_f \cdot [x_t, h_{t-1}] + b_f) \tag{3.15}$$

$$i = \sigma(W_i \cdot [x_t, h_{t-1}] + b_i) \tag{3.16}$$

$$o = \sigma(W_o \cdot [x_t, h_{t-1}] + b_o) \tag{3.17}$$

$$g = tanh(W_i \cdot [x_t, h_{t-1}] + b_i) \tag{3.18}$$

$$c_t = f \odot c_{t-1} + i \odot g \tag{3.19}$$

$$h_t = o \odot tanh(c_t) \tag{3.20}$$

In simpler words, the gates are used to compute the new states h_t and c_t. Interestingly, c_t is the results of two contribution, the previous c_{t-1}, multiplied by the forget gate, and the new input, correctly filtered by the other two gates. It can be noticed that two different types of nonlinearity are exploited. The sigmoid shrinks the input between 0 and 1. This is useful since the information that is not useful can be simply put to zero. The tanh, instead, lies between -1 and 1 and is used to compute h_t. Therefore, c_{t-1} is multiplied by a number between 0 and 1 and is increased or decreased by one in the most extreme cases.

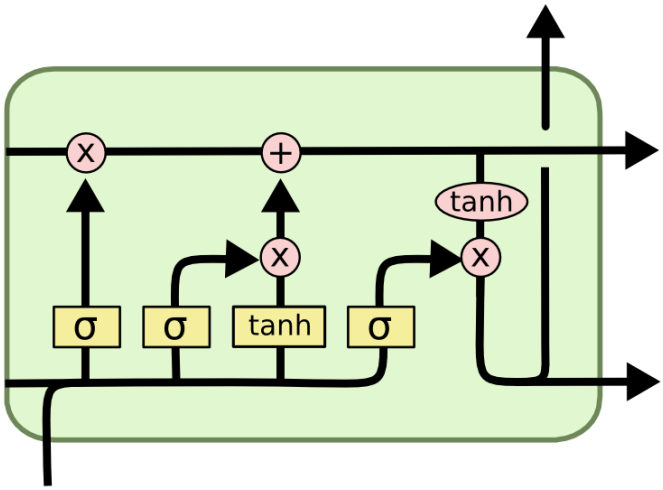

Figure 3.4: LSTM's cell structure [1]

The most important property of LSTM is that cell states can flow through the network without being interrupted: cell states and output are independent making the training with backpropagation through time more efficient with respect to vanilla RNN and solving the vanishing gradient issue.

The main reasons for these improvements are that, on one hand, there is an element-wise multiplication between c_{t-1} and f, that works way better than a full matrix multiplication. Also, the forget state can vary at each time-step instead of having multiplication with the same matrix over and over again, allowing for better numerical properties. The second reason is that the *tanh* used of computing the output, only multiplies h_t. During the backward pass the gradients do not flow through the non linearity at each time step but only once.

During the years, many variants of this type of network have been developed. For instance, GRU and LSTMs with Attention.

3.2.2 Gradient Recurrent Unit

With respect to LSTM cells, GRU [30] cells are simpler and do not have a separate memory cell. Therefore, they allow to generalize better with few examples and

[1]Source: colah.github.io/posts/2015-08-Understanding-LSTMs

are computationally more efficient, without the need of learning extra gates. Also, GRUs expose the full hidden state without an output gate and only have two gates:

- The reset gate, that selects the proportion of past information that will be forgotten.
- The update gate, which has a similar behaviour of LSTM's forget and input gate.

$$u = \sigma(W_u \cdot [x_t, h_{t-1}] + b_u) \tag{3.21}$$

$$r = \sigma(W_r \cdot [x_t, h_{t-1}] + b_r) \tag{3.22}$$

$$h'_t = tanh(W_h \cdot [x_t, h_{t-1} \odot r] + b_h) \tag{3.23}$$

$$h_t = h_{t-1} \odot u_t + (1 - u_t) \odot h'_t \tag{3.24}$$

In terms of formulas, first the results of update and reset gate are computed similarly as before. Then, the state is updated at the new time step partially taking information from the past state and the current one, through the use of the update gate.

3.3 Transformers

Introduced as an alternative to RNN, transformer is an architecture widely used in natural language processing. In *Attention is all you need*[31] the authors developed a novel method for the sequence to sequence task (a neural network that transforms a sequence into another one, for instance translation) purely based on the attention mechanism. Transformers have provided improvements in many different tasks and in managing long data sequences way better and more efficiently than LSTM.
The major limitations of recurrent neural networks which transformers try to solve are:

- No parallelization is possible and critical use of memory
- Difficult to learn very long dependencies
- Very hard to train, a large number of parameters to learn, and slow computations

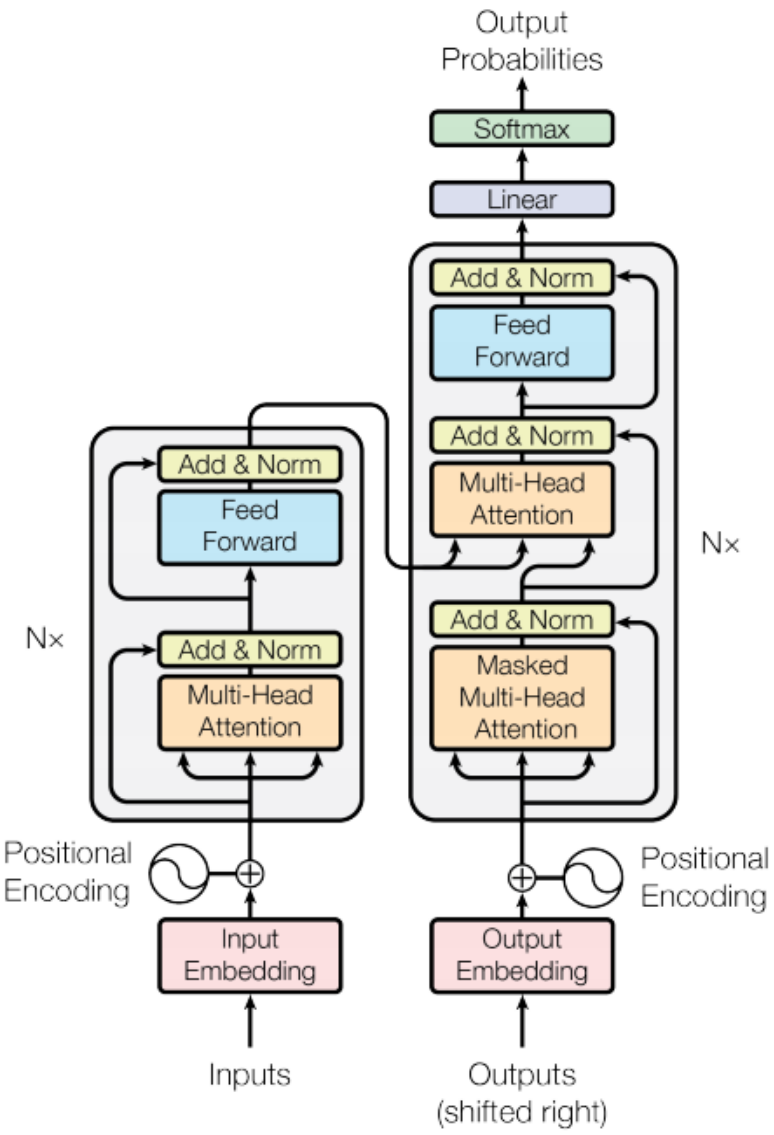

Figure 3.5: Transfomer architecture [31]

The self attention mechanism is a process that encodes a sequence $X = (x_1, ...x_n)$ into another sequence $Y = (y_1, ..., y_n)$. The elements of the resulting sequence y_i contains both the contributions of the original information x_i and the relationships that x_i has with the other elements. It is computed as a simple weighted sum.

$$y_i = \sum_j w_{ij} x_j$$

w_{ij} is derived by the input itself through a softmax operation and is not a parameter of the model, so it is not learned by backpropagation.

$$w_{ij} = \frac{exp(x_i^T x_j)}{\sum_j exp(x_i^T x_j)}$$

Hence, for a given input it is computed a weight for each other vector, including itself. In terms of matrices the operations can be rewritten a follows.

$$W = softmax(X^T X)$$
$$Y^T = W X^T$$

The second one is fundamentally a linear operation, which means non vanishing gradients with the last matrix multiplication. The first part instead, takes into account a non linearity and potentially vanishing gradients. The main advantage is that with self attention there is no problem at looking far into the sequence since all the inputs are used to compute the output at the same time and the sequential structure, that is present in recurrent neural networks is lost so far. Thinking the vectors as sequences of words, the intuition is that each word is dependent on each other and we are trying to understand how much attention a word x_i should pay with respect to itself and all the other ones in the set. This is a big advantage over bag of words, which considers the words independently.

To make the mechanism more flexible and powerful, the dot product can be replaced with a scaled version with the input dimension to keep the weights in a certain range, that otherwise will increase, and avoid vanishing gradients:

$$w'_{ij} = \frac{x_i^T x_j}{\sqrt{k}}$$

Another important point that should be noticed is that every vector appears in three different positions: First in the weighted sum for output computation, then in the weights computation matched with all the other vectors and finally as a match in the exponential. These are respectively called *Value*, *Query* and *Key*. It works like a (soft) dictionary where keys are mapped to values and can be retrieved through queries, but every key matches the query partially.
Therefore, three new matrices can be introduced for the linear transformations and the resulting values will replace the x_i in the previous passages.

$$k_i = K x_i + b_k$$
$$q_i = Q x_i + b_q$$
$$v_i = V x_i + b_v$$

In this way, with backpropagation, we can now also train these parameters in or The final step is the introduction of *multi-head attention*. The idea is that different words relate to each others with different relationships. For instance, in the phrase 'The food was not good', the words 'good' and 'not' have a relationship of inversion, while 'good' and 'food' have a relationship of property, in the sense that 'good' is an attribute of 'food'.
As a consequence the self attention layer can be separated in several parallel layers, that take as input a different projection in a lower dimensionality of the original sequence, and concatenate the output in a second moment.

Self-attention has proved to be extremely efficient, due to the simple structure and simple operations that are carried out. In addition, it allows for parallel computation, since the lost of the sequential structure and statelessness, and have a major advantage in an almost perfect long-term memory.

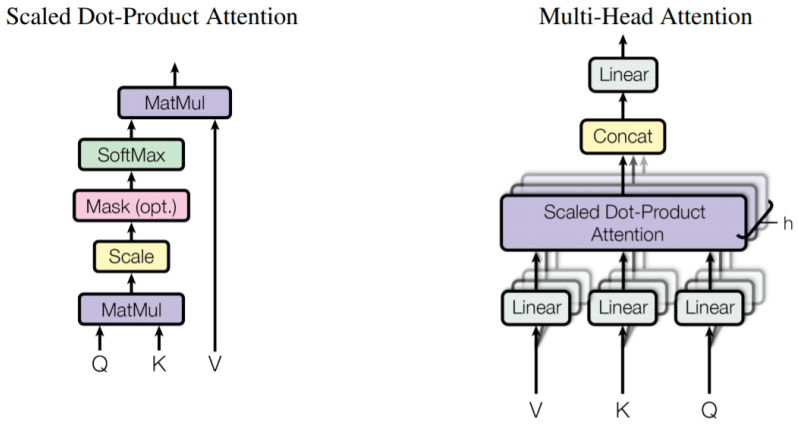

Figure 3.6: Self-Attention and Multi-Head Attention [31]

A transformer is defined as a model that used self-attention to propagate information along the time dimension. The original structure is mainly composed by two parts. The encoder takes an input sequence and maps it into a continuous representation, next the decoder takes the extracted information and generates a single output step by step.
The first step is passing the input to an embedding layer, that learns a vector representation for each word. Since these models have no recurrence, with the positional encoding the embeddings are expanded with also the information about the position of each word in the input. This is done by exploiting the sine and cosine functions for odd and even time steps respectively. According to the authors

[**30**], this representation allows the network to easily learn relative positions due to their linearity properties. Being also periodic functions, they benefit from not having to know in advance the length of the sequence. The positions are then added to the output of the embedding layer.

$$PE(pos, 2i) = sin(\frac{pos}{10000^{\frac{2i}{d_{model}}}})$$
$$PE(pos, 2i+1) = cos(\frac{pos}{10000^{\frac{2i}{d_{model}}}})$$

The encoder is made of two sub modules, a multi-headed attention and a fully connected network, which consists of two linear layers separated by a ReLU activation. There are also residual connections to help the gradients to flow directly through the network. The decoder, instead, is auto-regressive and takes as input not only the current words (the output of the encode) but also the previous output. In order to prevent the network from learning from future words, in the first multi head attention module of the decoder it is also used a mask to avoid computing attention scores for subsequent tokens. This is done by replacing the true value with negative infinity, which the softmax activation will automatically put to zero.

$$W'_{ij} \leftarrow -\infty \ if \ j > i$$

In the second multi head attention, the encoders output are used as queries and keys, while the values are the output of the previous layer. Finally, a linear layer works as a classifier and its size is the same of the dictionary. The predictions goes on until an end token is generated.

As far as the activation functions are concerned, LSTM networks suffered from an overuse of tanh and sigmoid functions. The main disadvantage of these non-linearities is that they have an upper bound. Here instead is used a ReLU activation, that allows the values to scale off to infinity since it does not saturate and is less sensitive to random initialization. Wrapping up, Transformers are very efficient and can be trained quickly even with a huge amount of data, thanks to the boost given by parallelization. Moreover, they can be finetuned and reused for new tasks, while for recurrent neural networks transfer learning it is not easily doable.

Chapter 4

Dataset

The study has been carried out in collaboration with the company, which manages the database of an important firm in the automotive sector. For the purpose of this thesis, event log data related to processes scheduled on a machine were used. These processes are formally called chains. A chain is a sequence of sub-processes that define a single flow of an ETL process. Each sub-process can be a bash script, a python module, or any PL/SQL procedure. Sub-processes are build to run in parallel or as a sequence. The log table stores the information of all the scheduled jobs which can be exploited to make predictions and analyzes. Its structure is shown in the table below.

Name	Description
catena_id	Unique identifier of the process
cd_job_catena_id	Corresponding name of the chain
cd_parent_job_id	Name of the sub-job within the chain
cd_lob_name_id	Activity done by the sub-job
source	Source Table
destination	Destination Table
nm_rows	Amount of data that is being processed
ds_status	Current status of the job
dt_mod	Timestamp associated to the entry

Table 4.1: Features of the process log table

A second dataset, concerning the scheduling of the processes, was provided. Each scheduling associates a calendar and a starting time to a chain.

Name	Description
scheduling_code	Unique identifier of the scheduling
calendar	Schedule days
early time	Schedule start time
description	Description of the scheduling in italian
command	Command launched on terminal

Table 4.2: Features of the scheduling table

As we will see later, the early time is not necessarily corresponding to the actual start of the chain. Instead, it is the minimum time from which the chain can start. The effective starting hour relies on file availability and on the end time of the chains on which it depends.
The chains can be divided into six categories according to their calendar (figure 4.8):

- Hourly: Very frequent chains that are scheduled more than once a day.
- Daily: Chains that are scheduled exactly once a day.
- Weekly: Chains scheduled only in one day of the week.
- Monthly: Chains that run once a month, typically the first or the fifth day of the month or the first working day of the month.
- Occasional: Chains that run on some specific days, as instance 2020-08-02.
- Manual: Chains that are not scheduled and are manually launched by the engineers when needed.

The two tables are not easily associable. The join key can be extracted from the command column of the scheduling table, in cases where it is set with 'launch main <chain-name>'

Finally, each chain is associated to a set of dependencies such as the presence of files in the corresponding folder or the termination of one or more previous

chains. Hence, while some chains have no predecessors, others cannot start if their requirements are not met. In that cases, the true starting time of a chain will suffer a delay due to the slowdowns of the processes that occur during the day.

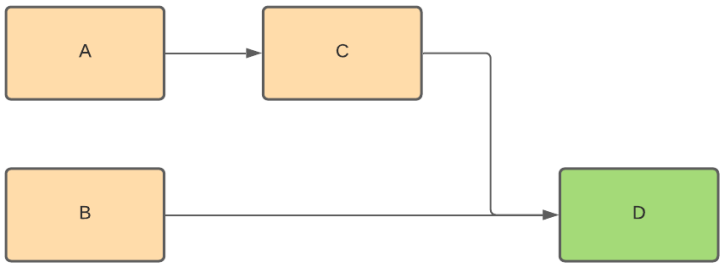

Figure 4.1: Example of a chain with three dependencies

The analyzed dataset is initially composed of 4 million rows with historical logs from January to April 2021.
For the purposes of this thesis and as a reference to what is illustrated in Chapter 2, the individual chains will be considered as the cases of the event log, while the jobs will be the events of the trace.

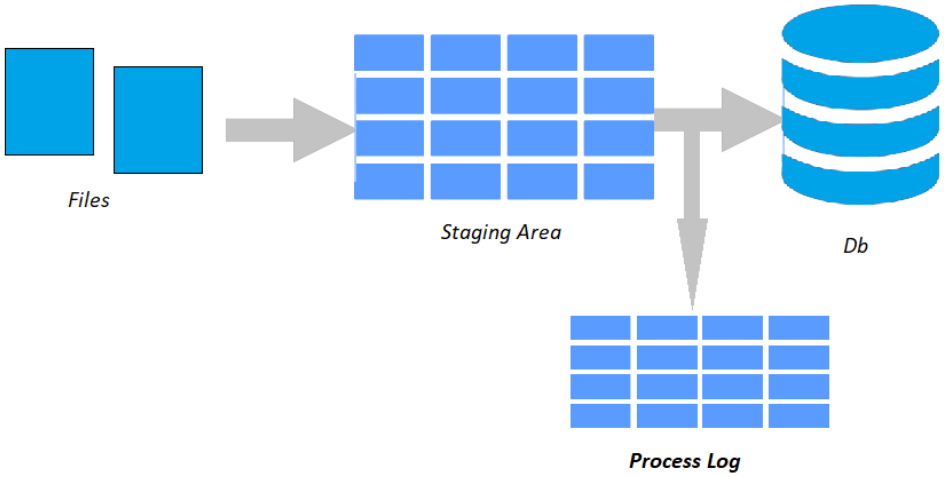

Figure 4.2: Data loading procedure into the corresponding tables

4.1 Data Cleaning

The quality of the data is crucial for any machine learning algorithm and it is important to detect issues and correct them in order to have a cleaner dataset, which means to respect the criteria of Validity, Completeness, Consistency and Uniformity. The dialogue and confrontation with the domain experts was fundamental to understand how the data was collected and what problems occurred during the acquisition.

The first problem is related to duplicate values. These can occur due to human errors or bugs in loading data. According to the assumption made before, the attribute *catena_id* is a unique identifier of a chain. In other words, there cannot be two chains having the same id. The presence of duplicates can have an impact on the performances of the models, due to the bias towards specific observations and a lower generalization capacity and, as a consequence, they have been removed.

Another key point for data quality is the presence of missing values. Null values in the database are often related to manual operations, tests, or routine processes that are not part of this analysis. In particular, for the process logs the following operations have been applied:

- The entries with negative or missing *catena_id* have been filtered out.
- The attribute *cd_parent_job_id* has null values only in the first and last event of the chain, or when the chain ends due to an error. Since this information is relevant, the missing values have been replaced with simple placeholders
- The column *nm_rows* has been set to 0 when null, this is reasonable since some processes could run even if there is no data to be processed.
- Finally, *ds_status* is often null, except in some cases where it is set as 'ERR' or 'WAR'. Consequently, this column will be replaced by a counter of warnings and errors within a case.

For last, there are cases of chains that have unusual behaviors: on one hand, there are two chains whose execution time (the difference between the last timestamp and the first one) is always zero. These chains have been removed since their resolution is immediate and there is no need of forecasting them.
An additional constraint that must be satisfied is that chains cannot last more than 24 hours. In the dataset, there are few entries whose execution time is incredibly large and which are worth investigating. After a confrontation with the domain experts, it has been discovered that once a chain ends with an error, it is restarted manually with the same id. Hence, in the calculation of the total running time the

restart time must be subtracted, since it cannot be predicted in any way. This does not solve all the problems, as there are still few cases that run for more than a day without errors but, instead, they remain on hold without finishing. The latter were considered outliers and had been removed.

Outliers removal is a crucial step in data processing. Outliers are extreme data that deviate from the other observations. They can be generated by novelties in the data but also by experimental errors or human mistakes. There exist many methods to detect outliers, besides the more standard as Z-score and Inter-quantile range, a more advanced method is isolation forest[32], an unsupervised tree-based algorithm. The main idea behind is that outliers should be easier to separate from the rest of the samples. As a consequence, if data is split according to some randomly chosen decision rules through a set of trees, in this case called isolation trees, they will be closer to the root with respect to the other points that will have a longer path. Formally, at each point is assigned an anomaly score, and it is marked as outlier if the score is larger than a threshold, typically close to one:

$$s(x, m) = 2^{-\frac{E(h(x))}{c(m)}} \tag{4.1}$$

Where $h(x)$ is the path length and $c(m)$ is a normalization factor corresponding to an estimated mean of h given m samples, computed as a failed search in a binary research tree.

4.2 Exploratory Data Analysis

A fascinating insight about the behaviour of the chains may come from the correlation analysis. In statistics, the measure of interest can be computed through the *Pearson correlation coefficient*, defined as the ratio between the covariance of the two variables and the product of the standard deviations. The result is a number between negative one, which means negative correlation, and one, that corresponds to a positive correlation.

$$Cov(x, y) = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y}) \qquad \rho = \frac{Cov(x, y)}{S_x S_y} \tag{4.2}$$

It is important to remark that correlation does not imply causation and, as a consequence, the variables must not have a spurious relationship.
Nevertheless, in the case of the chains, it is reasonable to think that a pattern

between the execution times in the same day may exist. As instance, we may think that if a chain that runs in the morning takes longer to run with respect to the usual, also its dependencies could be slower.

In order to compare two time series, the previous formula can be used to find a linear relationship. However, instead of comparing the values directly, the returns were used, i.e. the percentage changes with respect to the previous stage of the chain. This intuition comes from the financial field, typically exploited to see how the variation of a stock can influence the variation of a different one.

$$\rho = \frac{Cov(r_x, r_y)}{S_{r_x} S_{r_y}} \tag{4.3}$$

A different way to compute a correlation index is through the Spearman, that aims at describing the relationship between the variables using a monotonic function. The main advantage is that non-linear relationships can be discovered as well. The computation is similar but instead of using the raw values of the variables, they are converted to ranks.

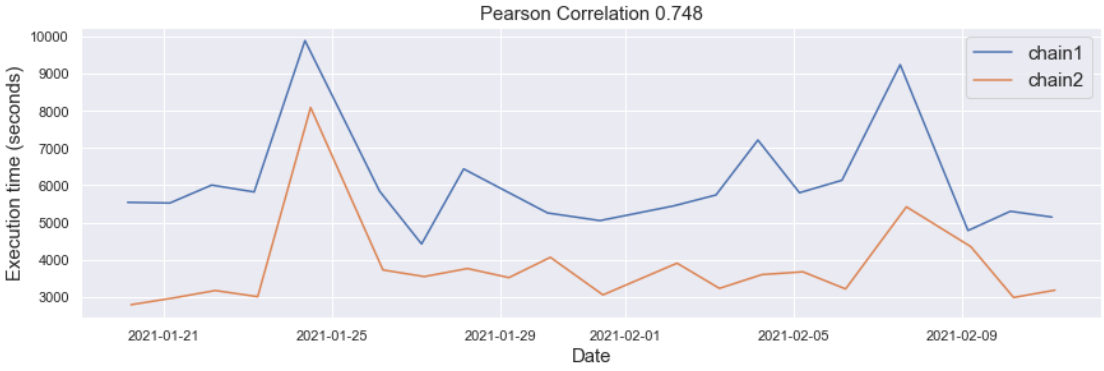

Figure 4.3: Trend comparison of two chains

The figure above shows the correlation of two chains in a twenty-days period. It is easy to spot that the two trends have a similar behavior and reach their peaks in the same day. Although they are not strictly linked by a relationship of causality, it can be thought that in the days corresponding to the peaks other chains also had a strong slowdown. Thus, there are also external factors that determine the execution time and the hypothesis is that if the first processes of the day are slower, the whole system will suffer.

It is therefore possible to seek to identify some reference chains whose trend

can then be reproduced by the following chains and insert this information in the forecasting model. In order to do this, we can compute the Pearson correlation between all chains, as stated before, and look for the most significant ones.

Figure 4.4 reports the correlation matrix for about ninety selected daily chains. Despite the premises, the results of the analysis show the absence of strong positive and negative correlations between the chains. These are generally uncorrelated with a few weak exceptions related to processes that often run concurrently without adding information for the prediction. In the end, it was decided to consider the chains independent of each other.

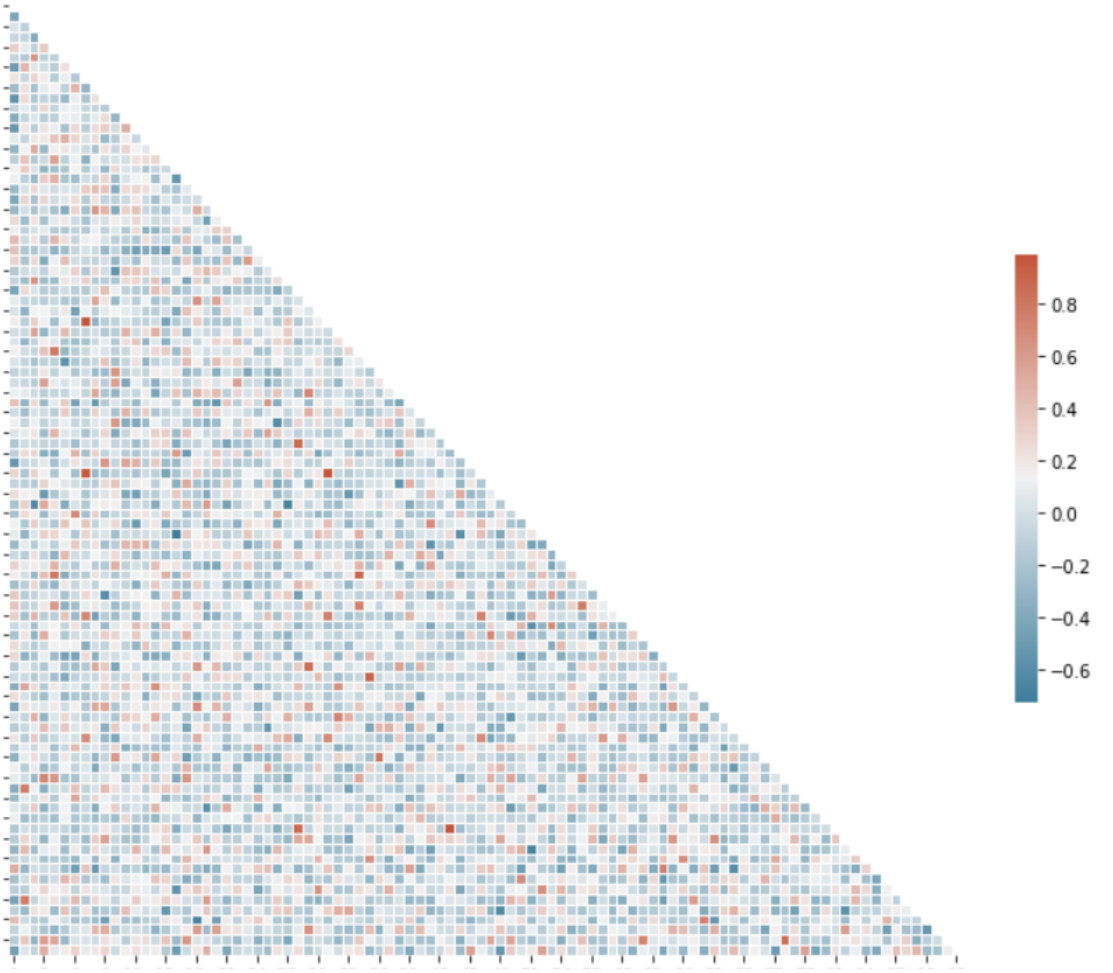

Figure 4.4: Correlation matrix

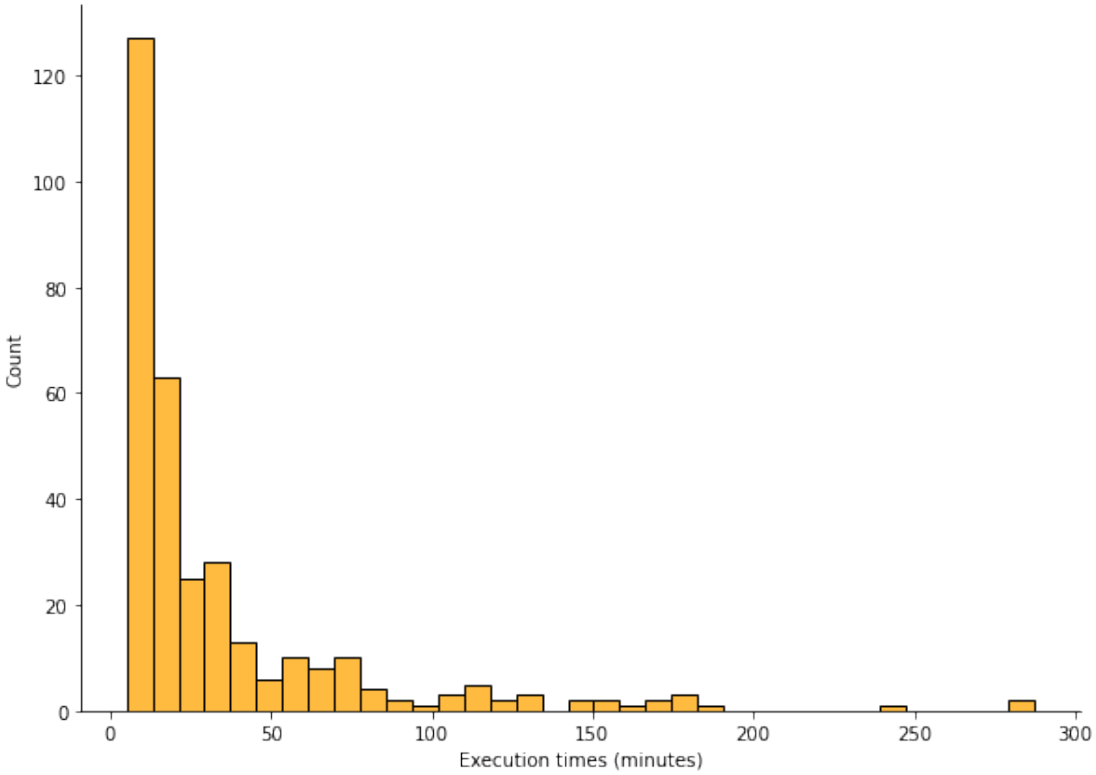

Figure 4.5: Distribution of execution times

Figure 4.5 shows how the executions times are distributed among the chains. The majority of the chains typically last less than an hour, so it becomes necessary to have a good precision in the forecasting of the times. However, some important chains have a lot of steps and can run even for longer times. The slower chains could also have an influence on the whole performance of the system, since they keep busy the machine for a long time. As stated before instead, very short chains are not worth studying.

Figure 4.6 represents instead the weekly trend of the times. It is easy to notice that while the durations are stationary during the working days, on the weekends there is great growth with high variability. This is due to the fact that on Saturdays and Sundays the processes, in addition to being less monitored, tend to run later, often overloading the machine. On Mondays, on the other hand, a much smaller number of chains are scheduled. Having the machine fewer processes to carry out, it will be faster and more efficient.

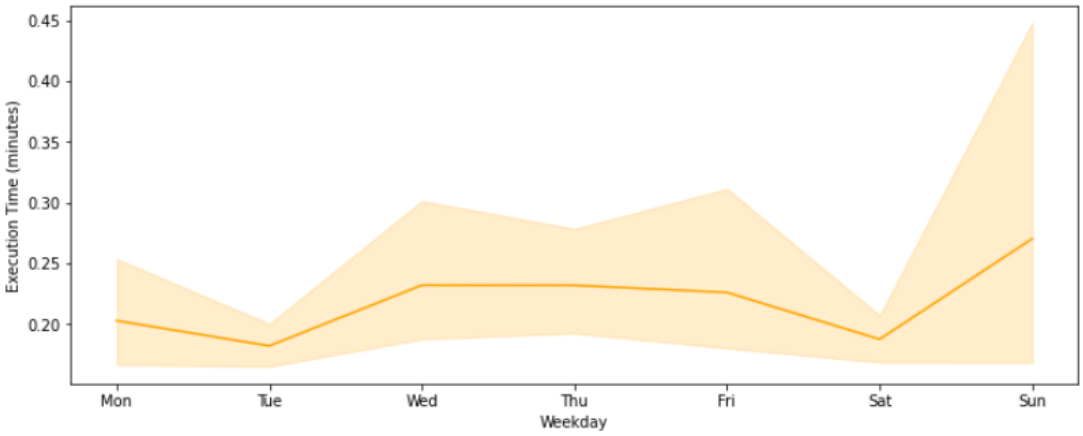

Figure 4.6: Mean and standard deviation of execution times during the week

Another interesting factor regarding the performances of the processes can be observed in the figure below. In the period following the end of March, there was in fact a dramatic improvement in execution times. The whole system has been optimized and migrated to a new version and, as a consequence, all chains have become more stable and shorter. As instance, it can be seen the changement from February to April of the three chains in the graph, named *chain1*, *chain2*, *chain3*, which are particularly relevant and are usually the ones that are most closely monitored.

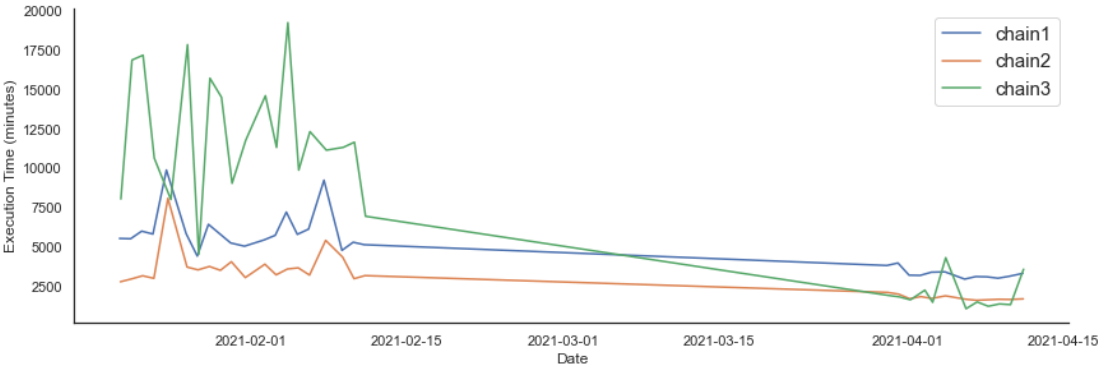

Figure 4.7: Performance before and after migration

However, given the great difference between the two versions, it is not possible to compare the relative logs. It was therefore decided to use only the data from April onwards for the training of the models, since otherwise the analysis would lose its meaning. Moreover, even the prediction could benefit from it, having more stable data and presumably a smaller amount of process failures.

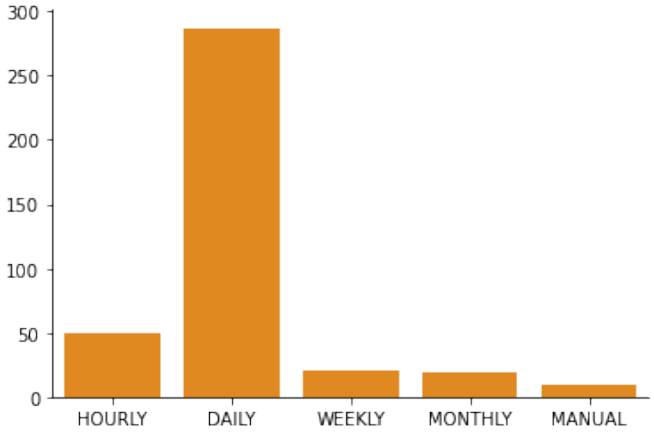

Figure 4.8: Number of chains for each category

Regarding the previously mentioned categorization of chains, it can be observed that the majority of them are daily and are scheduled each day at the same hour. Hourly chains instead are more sensitive to the time they are executed, since depending on the moment they will concurrently run with a different number of other chains. Many of the chains of this type are also dependent on themselves, that is, they cannot start unless the previous instance has finished. The weekly and monthly chains are more complicated to manage because of their rarity and it is not possible to establish their behavior from the evidence of the data. Finally, manual chains, as mentioned before, have been eliminated from the analysis as they are too subject to human behavior.

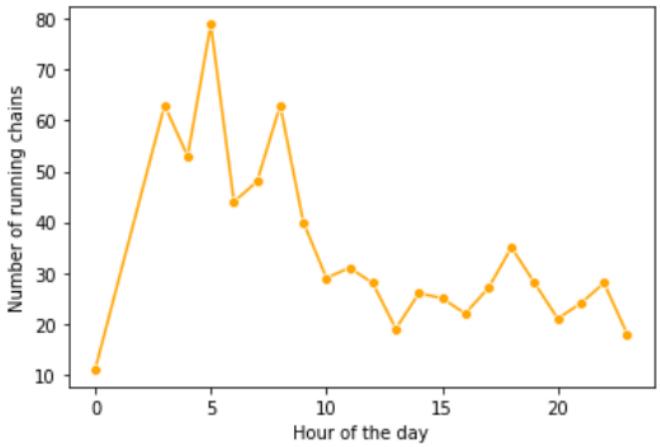

Figure 4.9: Number of scheduled chains for each hour

The last analyzed aspect of the processes is the timetable. As shown by figure 4.9, most of them typically run early in the morning, before or at the beginning of the working day and with peaks at 03:00 and 05:00. The latters are the most important since their delay would not allow the results to be available in a timely manner. Likewise, since the machine experiences a higher workload at those times, the chains are typically slower than they would be at other times of the day.

Once the dataset has been cleaned and analyzed, it is finally ready to be processed in order to be ingested by machine learning models in the next phase of the pipeline.

Chapter 5

Proposed Approach

As stated in the previous chapters, the goal of the project is to build a model able to predict the remaining time of running traces. To do this, two different approaches will be compared: On the one hand, the recurrent neural networks, in particular LSTM and GRU, will be tested, on the other hand, a model based on the attention mechanism, a transformer, will be developed. The models will be subsequently trained and tested using the previously described dataset, whose pre-processing will be described shortly.

5.1 Performance Evaluation

A commonly used evaluation metric for continuous variables is the mean absolute error (MAE). This metric measures the distance between each pair of observed and predicted values, respectively x_i and y_i, and it is computed as an average of the absolute errors:

$$MAE = \frac{\sum_{i=1}^{n} |x_i - y_i|}{n} \tag{5.1}$$

Since it can be interpreted as an error measure, the lower its value the better it is. Also, due to the use of the absolute value on the same scale of the data, it punishes with the same scale big and small errors but is a simple and intuitive choice when extreme prediction are not taken into account.

$$MSE = \frac{\sum_{i=1}^{n} (x_i - y_i)^2}{n} \tag{5.2}$$

A possible alternative is the mean squared error (MSE), which weights heavier very predictions far from the mean but bring less importance to small mistakes.

5.2 Preprocessing

In order to reproduce the real behaviour of the logs, the events and the cases have been ordered according to their timestamps. At the test time, when a new log arrives, it follows the preprocessing pipeline and is appended to the others. The dataset have been split in different sets: 70% of data have been used for training, about 20% for hyperparameter tuning and validation while the remaining part was used for testing.

First of all, since each even may output more than one log, for each case-id they have been grouped by the parent-job-id. Moreover, for each created row of the dataset, thanks to the intuitions coming from the exploratory analysis, have been extracted the following features :

- Day of the week: Categorical variable corresponding to the values from Monday to Sunday
- Hour: Hour of the day corresponding at the ending of the subjob. While for the daily chains may be more or less fixed, it is important to distinguish the hourly ones.
- Execution Time: Computed as the difference max(date) - min(date), it is the duration of the subjob within a case and gives an indication of the current speed of the procedure.
- Errors: Counter of the errors occurred during the execution of the job and extracted from the column *status*

The rows of the newly created dataset have been associated with the remaining time value of the case, that worked as the target variable of the supervised learning problem. In fact, for each extracted event, the difference between the end of the case and the end of the job has been computed.
The main challenge related to the target variable concerned the chains which, due to an occurred error, have been stuck for a certain period of time. The causes of a possible error can be multiple, for example the loss of connection during the process or a syntax error within the code. However, this type of interruption cannot be predicted in advance. Furthermore, when a chain breaks it is usually fixed on the spot and restarted. Given the impossibility of predicting how long a chain will be blocked, the values of the remaining time have been purified during the training phase, subtracting the difference between the error timestamp and the one of the next log. This solution was accepted by the team, agreeing to predict the actual remaining time without taking into account the time lost in restarting the processes.

Given the natural sequential structure of the traces, it appears natural to model the problem in a way that is manageable by a recurrent neural networks. In addition to the attributes mentioned before, the other mandatory features are the name of the chain and the name of the job associated to a specific event. In other words, in order to obtain a fixed size vector these categorical variables must be encoded. Although a solution such as a one hot encoding can come in handy, this leads to major problems in terms of performance and memory space: in fact, having to manage about a hundred chains and thousands of different jobs, would result in having as many columns in the dataset almost all mapped to zero and an extremely large and intractable sparse matrix would be obtained. It was therefore decided to opt for a simpler integer encoding for the job name feature, while maintaining the order relationship within a case. It is reasonable to think that large integers values are correlated with a best-case remaining time, as they correspond to a more advanced state of the chain.

Id	Chain	Job	Weekday	Hour	ExecTime	Err	Remaining
21456	Orders	Initialize	Mon	16	0.20	0	60
21456	Orders	Loading	Mon	16	0.20	2	59.8
21456	Orders	Transform	Mon	16	10	0	49.8
21456	Orders	Mapping	Mon	16	20	0	39
21456	Orders	Terminated	Mon	17	00	0	0
..	..	..	..	..	..	..	..
21458	Cars	Initialize	Mon	18	15	0	80
21458	Cars	Error	Mon	18	0	0	65

Table 5.1: Data after preprocessing

Table 5.1 shows the structure of the dataset used for forecasting. In the real time context, the logs will arrive from time to time according to the order of conclusion of the sub-process. It is important to remember that each id is the unique identifier of the instance of the Chain. In this sense, the id column is not used for prediction, since otherwise it would lead to extreme overfitting, but only to separate the vectors for a correct input. Another factor that must be noted is that the lines corresponding to the termination of the processes have been removed,

i.e. those with target label 0, since it is out of the scope of the analysis to identify the processes that have already terminated.

Since we expect the model to predict partial traces, the former must be extracted from the dataset. Given a sequence of length n, from the corresponding trace $\sigma =< e_1, ..., e_n >$, all the prefixes have been taken into account: for each prefix length $1 \leq k \leq n$ the subsequence $hd^k(\sigma) =< e_1, ..., e_k >$ can be derived and, as a consequence, the whole set of partial traces is obtained. Let's consider as instance the trace $< A, B, C >$, where each element is a vector of the form presented in table 5.1. Then, the extracted traces are $< A >$, $< A, B >$, $< A, B, C >$. Despite some approaches only make use of some of the prefixes, like taking only the last t elements, it has been chosen to include all of them, considered the complexity of the problem and in addition to the ability to obtain a prediction even with sequences of length one.

After the extraction, the sequences must be encoded into a fixed length vector. The chosen approach is the one known in literature as *prefixes padded*, that consists in applying zero padding to the partial traces whose length is shorter that a certain vector length. In this case, the final length of the vectors would be the one of the longest trace in the training event log. Considering the beforementioned example, with a maximum length equals to four, at this point we would have the sequences $<$ 0,0,0, $A >$, $<$ 0,0, $A, B >$, $< 0, A, B, C >$. Once that all the cases have been processed, the network can be finally trained using the remaining time as target variable. However, one of the limitations of this approach that can hardly be solved, lies in the possibility of having longer than expected sequences at test times. Although this is a remote problem, it was decided, for these cases, to consider only the last k events, obtaining a worsening of the performances that could only be solved through a new training phase.

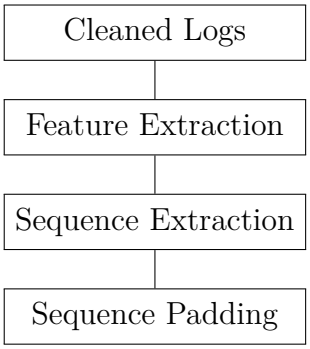

Figure 5.1: Overview of the procedure

In order to make distribution of the attributes homogeneous and comparable between each others, the numerical features have finally been standardized:

$$x'_{ij} = \frac{x_{ij} - \mu_j}{\sigma_j} \tag{5.3}$$

where μ_j and σ_j are the mean and the standard deviation of the jth attribute respectively.

5.3 LSTM and GRU implementation

The proposed architecture ha been tested on the validation set with different configuration. Among the considered hyperparameters, the number of layers, the presence of batch normalization layer and the number of neurons for each layer have been taken into account. Moreover, different experiments have been with both Adam and Sgd optimizers with different learning rates. In particular, the number of LSTM layers have been chosen between [1,2,3,4], learning rate = [0.1,0.01,0.001,0.0001], neurons = [50,100,150,200]. The number of epochs for each run has been fixed to 150 and mean absolute error as loss. At the end of the procedure, the network performing the lowest MAE on the validation set has been chosen. A similar process has been applied to a GRU network. At the end of both model, a single output dense layer has been applied in order to retrieve the prediction. The results are shown in the tables below.

Hyperparameter	Value
Learning Rate	0.01
LSTM layers	3
Batch Normalization	True
Neurons	200

Table 5.2: LSTM best configuration

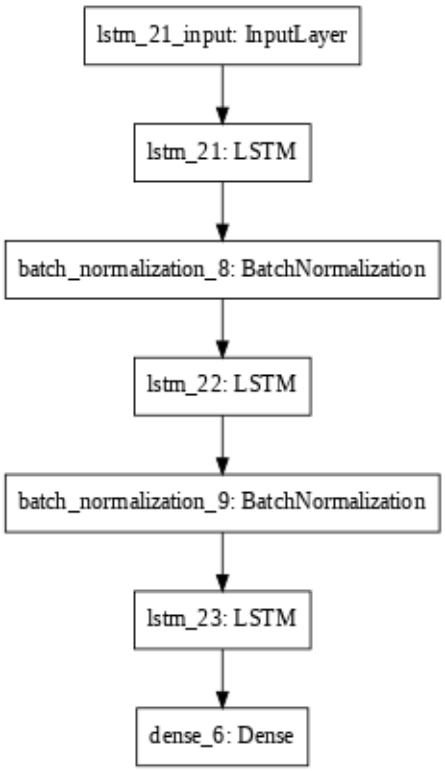

Figure 5.2: LSTM model

Hyperparameter	Value
Learning Rate	0.001
Gru layers	2
Batch Normalization	False
Neurons	100

Table 5.3: GRU best configuration

It can be noticed that the GRU network is able to reach the best result with a simpler architecture with respect to LSTM. Having a lower number of parameters due to the simplified gates structure, it is also stabler and faster to train. This is not surprising since GRU have proved to outperform LSTM in a variety of sequence modeling tasks in terms of generalization [33]

5.4 Transfomer implementation

Recurrent neural networks are not always able to handle long and complex dependencies such as the ones in the event log. In order to solve this issue, a transformer architecture is proposed. Being this a regression problem and not a sequence-to-sequence task, with respect to the original architecture, here there is no need to have both the encoder and the decoder part. The input of the network is again a partial trace following the same preprocessing as before, with the only difference being the vector shape, due to the transformer statelessness.

The first step of the pipeline consists in embedding learning. Instead of having one-hot encoding or integer encoding for the event names, they are projected instead in a lower dimensional space. This tries to overcome two problems, on the one hand it limits the use of memory compared to one-hot, on the other it allows to map events that have a similar behavior in the vector space, improving learning. The embeddings are subsequently added to the information derived from the positions of the individual events within the vector, avoiding losing the sequential structure of the logs.

At this point, the attention mechanism comes into play: Not all events are in fact related to each other in the same way, it is therefore crucial to select only the most important connections and to understand which events are to be attributed the higher score to get a good prediction and manage even very long sequences. Moreover, with multi-head attention to capture even multiple types of representations at the same time. With the same notation of chapter 4 the obtained results are computed as

$$MultiHead(Q, K, V) = concatenate(Head_i)W_o \tag{5.4}$$

$$Head_i = Attention_i(K, Q, V)$$

Where the matrix W_o is learned by backpropagation and the $Head_i$ are the output of the single attention layers. The output is finally passed through dropout and feed forward layers, with residual connections to have a better gradient flow. The final part of the network is mainly composed again by linear layers with relu activation, separated by dropouts layers to prevent overfitting, and preceded by a max pooling to aggregate the extracted features. Figure 5.3 shows an high level structure of the chosen architecture.

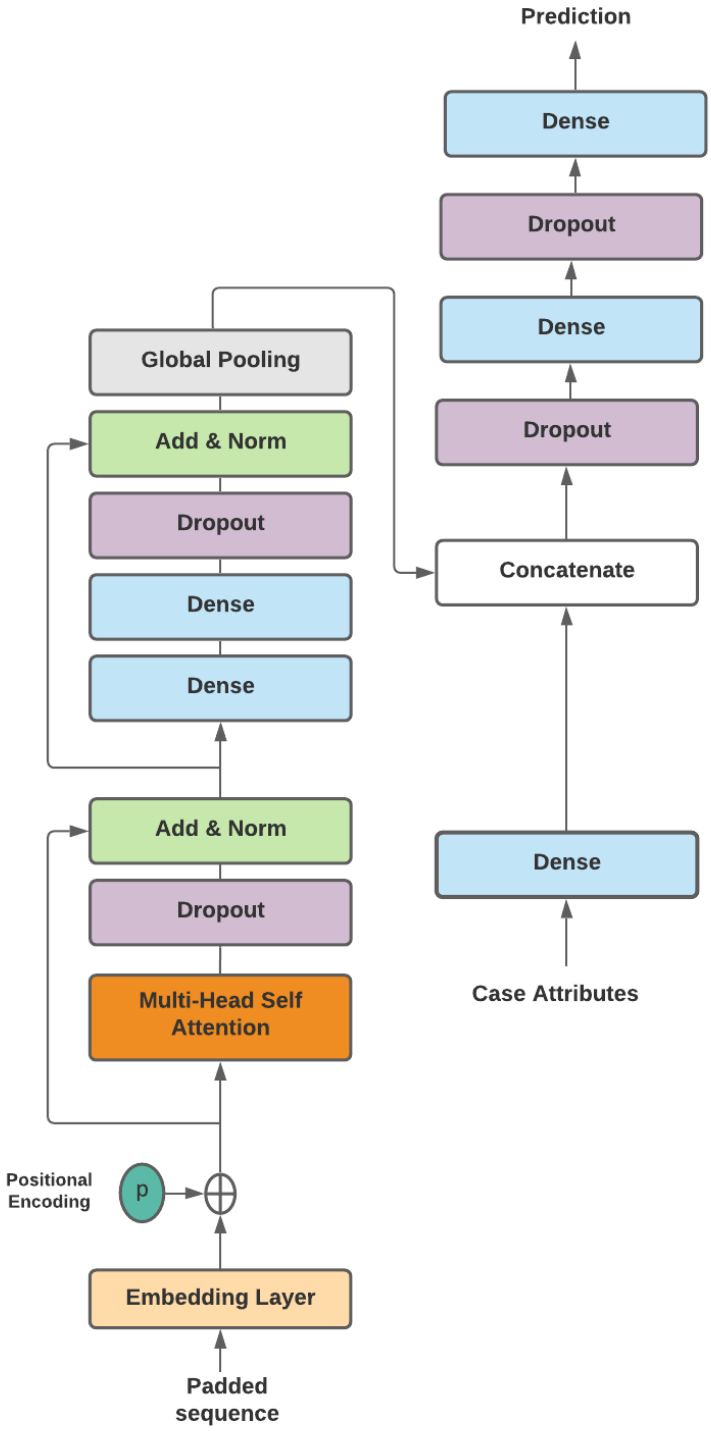

Figure 5.3: Transformer architecture used for process monitoring

The loss function used to train the network is the log-cosh. The main advantages of this loss is given by the fact that for large values of x, $log(cosh(x))$ approaches $|x| - log(2)$, while for small values of of it, its can be approximated by exploiting the Taylor series by $\frac{x^2}{2}$. The consequence is that it solves the problems of both MAE and MSE loss problems: On one hand residuals that are big are minimized with the absolute value, that is less sensitive to outliers, on the other smaller residuals are minimized with the square function, making the learning more precise. Moreover, the log-cosh is also twice differentiable, which makes its use beneficial also for models like XGboost, due to the the use of second derivatives.

$$L(y, y^p) = \sum_i log(cosh(y_i^p - y_i)) \tag{5.5}$$

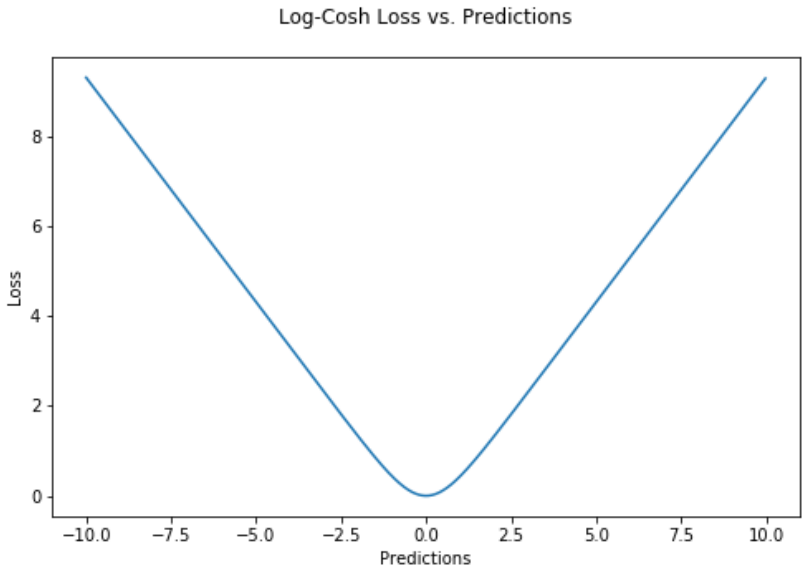

Figure 5.4: Log-cosh loss function

Regarding the network implementation, it has been chosen to use an embedding layer with 36 output units, as well as a positional encoding of the same size in order to allow the summation between the two. The number of heads has been fixed to 4, while the last linear layers have 32 and 128 units respectively. Inside the transformer block, the feed forward is made of two consecutive linear layers, the first with a Relu activation and the second with a linear activation. The dropout rate is set to 0.1. Finally, a dense layer with single output unit and linear activation has been applied to obtain the final prediction.

5.5 Framework overview

The trained machine learning model is then inserted into a broader context that takes into consideration the behavior of the other chains scheduled on a certain day to have both information on the remaining time of a process and a statistical comparison on the trend of the previous days.
The goal is to report the presence of delays or advances with respect to the standard case and predict the completion time of a chain that has not yet started.

In order to obtain a confidence interval for the population mean of each chain, the execution times were considered to be distributed normally on the same day of the week. Although it is not easy to test the normality of the data having few samples, it can be stated that this property holds asymptotically for the central limit theorem. To estimate the mean of a normal population with unknown variance is often used the t student distribution. A t random variable is defined as the ratio of two random variables, the first from a standard and the second chi square distributed

$$t_{N-1} = \frac{Z}{\sqrt{\frac{k}{N-1}}} = \frac{\bar{X} - \mu}{\sqrt{\frac{S^2}{N}}} \tag{5.6}$$

where $\bar{X}$ and S^2 are the mean and variance unbiased estimators of the population respectively, while N the sample size. It can be proven that $k = \frac{(N-1)S^2}{\sigma^2}$ follows a χ^2 distribution with $N-1$ degrees of freedom.

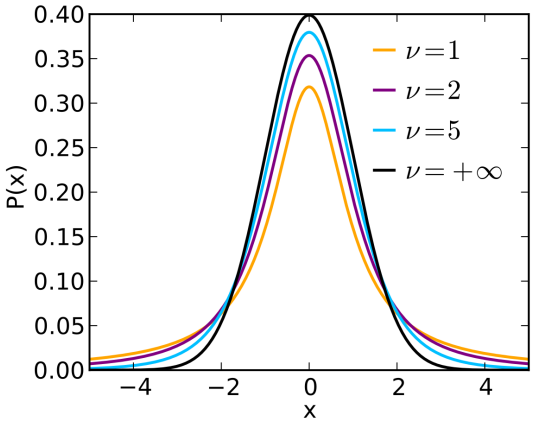

Figure 5.5: Probability density function of Student's t-distribution [1]

It can be easily observed that the distribution is symmetric and has a shape similar to the one of the gaussian. However, with respect to the former, the t distributions has heavier tails. In fact, it allocates more probability to samples that can far from the mean. The distribution is usually used to obtain confidence intervals for the mean i.e a statistical approach that in contrast to the point estimate (that consists in a single value) provides the range in which the parameter should be contained with a certain level of confidence.

In this case we have chosen to provide a 90% confidence interval for each chain and day of the week. The intuition is that for each chain, the distribution of execution times is fixed for a certain day of the week. On the contrary, considering different days together would result in less stable data and a poor accuracy, as showed in the data exploration. Formally, the confidence interval is computed starting from a two sided hypotesis testing:

$$P(-z_\alpha \leq T \leq z_\alpha) = 2F(z_\alpha) - 1 = \alpha \tag{5.7}$$

Where z_α is the quantile $q_{1-\frac{\alpha}{2}}$ of the t student distribution. At this point, by solving for the parameter of interest μ the range is obtained:

$$[\bar{X} - z_\alpha\sqrt{\frac{S^2}{N}}, \bar{X} + z_\alpha\sqrt{\frac{S^2}{N}}] \tag{5.8}$$

This information, in addition to being easily interpretable even by non-technical users, can help to report anomalous cases that occur during the day. In particular, at the end of the process, it can be shown through an alerting mechanism if this has had a very different timing than the average and that therefore interventions must be carried out to optimize the procedures. The main reason is that the data on the remaining time alone is not enough and adds little value to the analysis if it is not compared to the other instances of the same chain. Conversely, combining the prediction result with a well-constructed confidence interval can help monitor the performance of the entire system. As instance, process that took two hours and usually would have taken an hour and a half, must be notified quickly as it will delay all other chains dependent on him.
When it comes to finished in advance chains, this is often a positive news since data will be available on time. However, there may be cases of interrupted chains that have ended up in error, manually interrupted or chains that have turned but have skipped steps due to the presence of empty files on which to do etl and therefore did not actually do any operation. The result is, as a consequence, a series of

[1]Source: https://upload.wikimedia.org/wikipedia/commons/4/41/Student_t_pdf.svg

processes with a very short duration which however do not bring any advantage and can be misleading.

In order to provide a complete the prediction, the company needs also to have an estimation of the remaining time until the completion of a chain starting from its dependencies. In other words, the goal is to predict a process before its starting. This task has a series of difficulties behind it, since it is not possible to make precise predictions without available logs. Therefore, given the impossibility of predicting malfunctions or slowdowns hours before their occurrence, it was decided to use the machine learning model only for the processes in progress at a certain time.
At this point, once the prediction for a chain has been obtained, it is possible to add the value obtained to the confidence interval of the subsequent processes, computed as showed before, to obtain a final estimate (without counting the errors and interruptions) according to the dependencies tree through a recursive procedure.
Using a second parallel model, for example a random forest, was a possibility taken into consideration, but was subsequently excluded for two reasons: the lack of correlated variables obtainable only from the logs (in addition to the day of the week and an approximate starting time) and avoiding a more complex deployment and further training which in any case would not bring great benefits.
In any case, the process tree can have a complex structure (figure 5.6) and does not take into consideration human intervention, which can have different timing depending on the situation, and file dependencies that cannot be modeled.

Another analysis that was discarded was to represent the course of the execution times of the individual chains as a time series. Although it may be interesting, it was not possible to identify positive or negative trends during the analyzed period. A common way to measure the predictability of a time series, proposed by Diebold and Kilian [34], consists in the computation of the ratio between two forecasting errors, one long term and one short term:

$$P = 1 - \frac{E(L(e_{t,t+k})}{E(L(e_{t,t+j}))} \tag{5.9}$$

When P is large (approximately one) the series is predictable, while for values of P close to zero, like in our case, according to the authors the relative predictability is low.

The following algorithm shows the steps used to get the final value. The assumption is that all the chains at the root of the tree (i.e. those without predecessors) have started. Otherwise, it will not be possible to obtain the prediction and it will be necessary to wait for the subsequent logs.

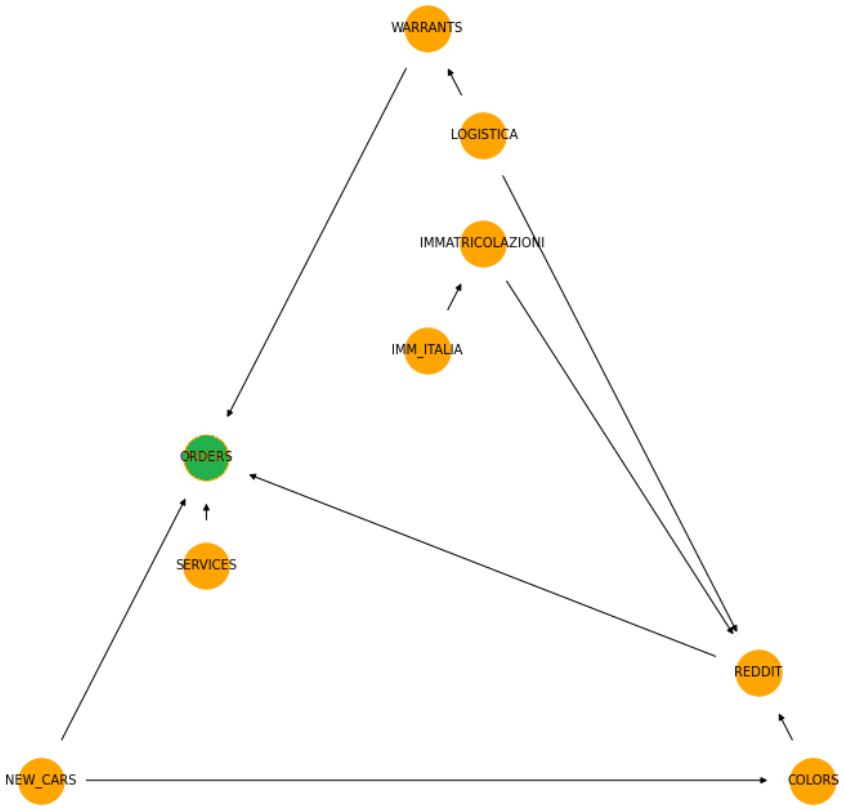

Figure 5.6: Dependency graph

Algorithm 1 Remaining-Time

Input: Logs with partial traces and chain name

if *Chain is finished* **then**
  return 0
else
  if *Chain is started* **then**
    return $model(preprocess(logs))$
  else
    predictions $\longleftarrow \emptyset$
    foreach *dependence d in dependencies(chain)* **do**
    $predictions[d] = Remaining - Time(d)$;
    return $max(predictions) + Confidence - Interval(chain)$
  end
end

Finally, we have chosen to alert for the presence of single events (subjobs) during the execution of the application that are recognized as anomalous. Not having the possibility to put labels manually and train a classification model, we have chosen to exploit an unsupervised algorithm for outliers detection to spot noisy data that live far from the other points. Among the possible alternatives, the choice fell on the use of a density based approach.

Local Outlier Factor is a density based algorithm [35] able to identify local outliers. In other words, a point will be considered anomalous based on the density of its neighborhood.
Let p be a point, we can define the k-distance(p) as the distance between p to its k-th nearest neighbor and denote by $N_k(p)$ the set of neighbors whose distance is lower or equal than $k\text{-}distance(p)$.
Next, we can introduce the reachability distance, a metric that is equal to the true distance of two points if it is greater than the k-distance, otherwise is set to the the k-distance value:

$$reach\text{-}dist_k(a,b) = max(d(a,b), k\text{-}distance(a)) \tag{5.10}$$

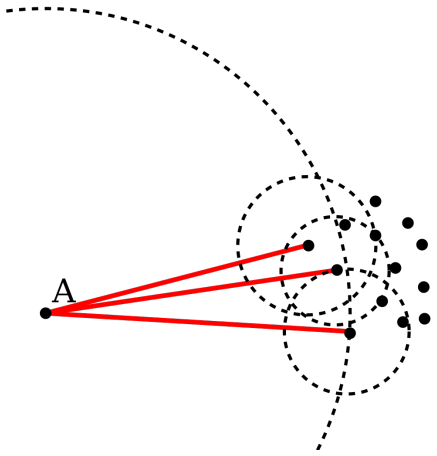

Figure 5.7: Local Outlier Factor intuition

It can be easily noticed that the reachability distance will depend on whether the point is crowded or isolated. Moreover, the k-local reachability density can be defined as the inverse of the average of the reachability between the point A and its neighbors:

$$lrd_k(a) = \frac{|N_k(A)|}{\sum_{B \in N_k(A)} reach\text{-}dist_k(a,b)} \tag{5.11}$$

Finally, the k-Local Density Factor is computed by dividing the average lrd of the points in $|N_k(A)|$ with $lrd(A)$:

$$LOF_k(a) = \frac{\sum_{B \in N_k(A)} lrd_k(B)}{|N_k(A)| lrd_k(A)} \tag{5.12}$$

The point is considered an outlier (anomaly) if the score is bigger than 1 (with a certain threshold) since it would have a lower density of its k neighbors while for lower values of LOF the point would likely be an inlier. The main advantage with respect to methods such as Isolation Forest or One class Svm is that the local approach allows to isolate and identify points that from a global point of view would not be labeled as anomalies.

Regarding the implementation for our case study, a LOF algorithm has been fitted on the whole set of events, taking into account, in addition to the name of the event, the duration, the timestamp corresponding to its ending and the day of the week, with the aim of identifying anomalous jobs both for their duration and for their end time. The hyperparameter k has been set to 20, chosen after a series of tests to avoid reporting too many abnormal values. However, given the difficulty of evaluating this type of algorithms, the final judgment on whether or not to intervene is still left to the human user.

5.6 Deployment

One of the last stages of the development consists of bringing a machine learning model into production, and this is not a trivial task. To make the prediction available in a practical way and allow the model to be reached in the production environment, it has been chosen to expose a set of REST APIs so that a client application can easily use a POST request method to retrieve the needed information in the form of the standard file format json. To do this, a combination of cherrypy, a python microframework for developing web applications, and Docker for managing the environment was used.
In particular, the following functionalities have been made available in the form of apis:

- Visualization of the chains currently running and still not ended

- Reporting of chains ended up in error and currently blocked, signaling of chains that depend on the latter
- Remaining Time prediction of running and non running chains according to the modalities showed before
- Percentage of completion, departure and end times and accumulated delay or advance compared to the values of the previous days.
- Graphical representation of the dependency graph extracted from the scheduling table
- Check for the presence of files that prevent a chain from starting

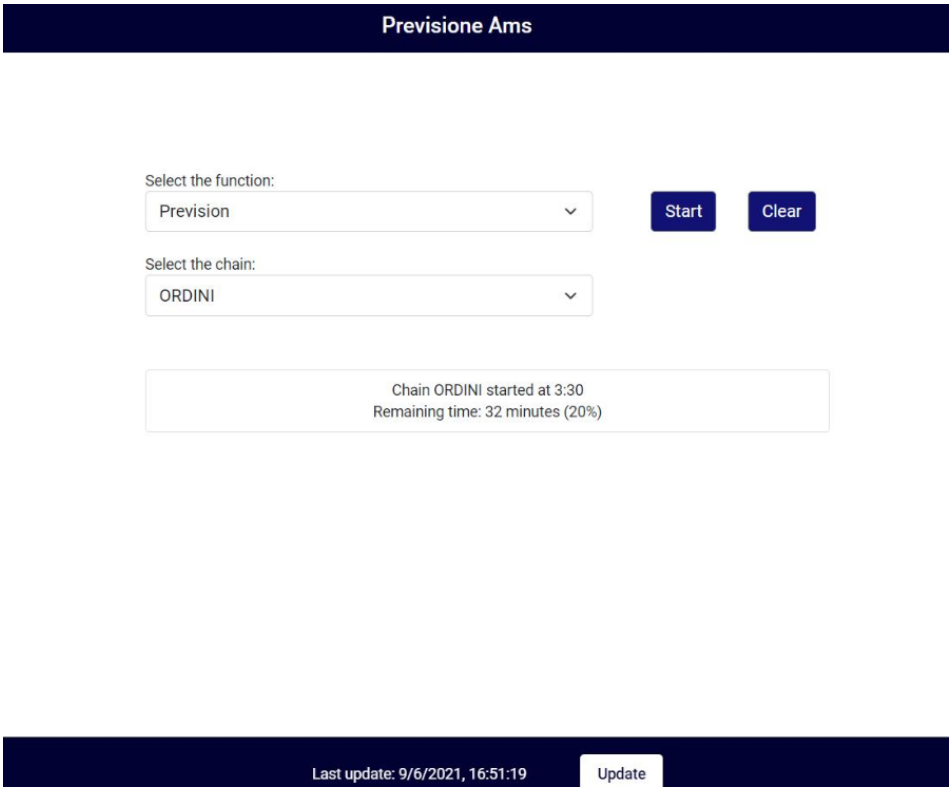

Figure 5.8: Developed front-end main page

At a later stage, a front end application was also developed to make it easy to access and visualize, in addition to the prediction of the remaining time, also statistical analysis of system performance, extract dependencies and possible delays, in order to efficiently report any deviations from the norm. The application is able to obtain data directly from the database, in order to be updated at any moment.

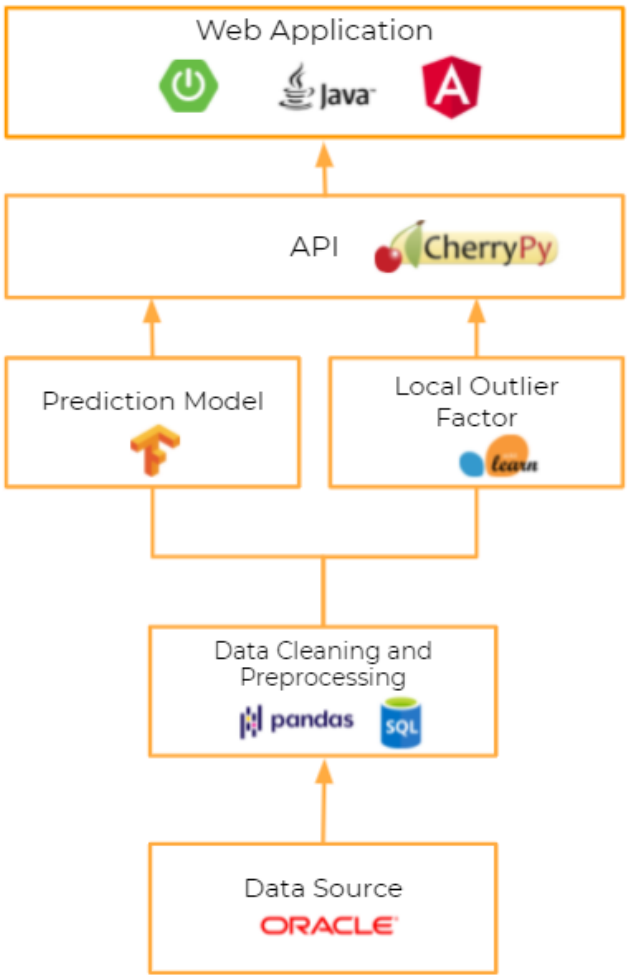

Figure 5.9: Block diagram of the developed framework

Summing up, the above diagram shows the various blocks and technologies used for the development of the framework. Starting from the bottom, the data initially collected in an oracle relational database are cleaned (as showed in chapter 4) and processed through SQL statements and python functions. Subsequently, part of the data is used for the training of the machine learning model and the unsupervised anomaly detection algorithm. At inference time, data related to ongoing processes is made available and is processed in the same way to obtain the remaining time prediction. The APIs are finally used in the back-end of the application, but can also be exploited from the outside.

Chapter 6

Experimental Results

We report here the experimental results for remaining time prediction on the test set. Due to the absence of previous experiments on this dataset, the two proposed alternatives have been compared with two baselines implemented with random forest and XGBoost and evaluated with the metrics illustrated before.

Figure 6.1 illustrates the trend of the true remaining time for some chains and the corresponding predictions on one of the days of the test set. The y-axis reports the remaining time in minutes, while on the x-axis there is the number of steps within the chain itself. It can be noticed that all models behave similarly in the case of easily predictable chains as in graph (a), related to one of the longest and most regular chains. Regarding graphs (b) and (c), we note how all the models underestimate the total duration of the process. Although the difference in terms of minutes is minimal, there may therefore be cases in which the chains have random variations that are difficult to predict, especially if the chain takes longer than necessary but the variation within individual cases is hardly noticeable and the training set it is not large enough to contain all possible sources of slowdown. Baselines based on tree models do not differ so much from those based on neural networks, however, in some situations, they struggle to recognize the downward trend of the process and have strong upward peaks in some points, as can be seen in (e) and (d). Moreover, case (d) is a particular chain that puts all models in difficulty, especially in the initial phase, where it is overestimated by LSTM and GRU and underestimated by the others. This chain (as well as others in the dataset) follows an irregular trend and since usually it would take just over half an hour, it can be said that an anomaly probably occurred that day. According to the designers, the chain has been implemented so as not to report the presence of errors in the log like the others, thus making the forecast more complex. Overall, it was preferred to opt for a model that overestimates the remaining time, rather than providing a wrong prediction that would not give an alarm.

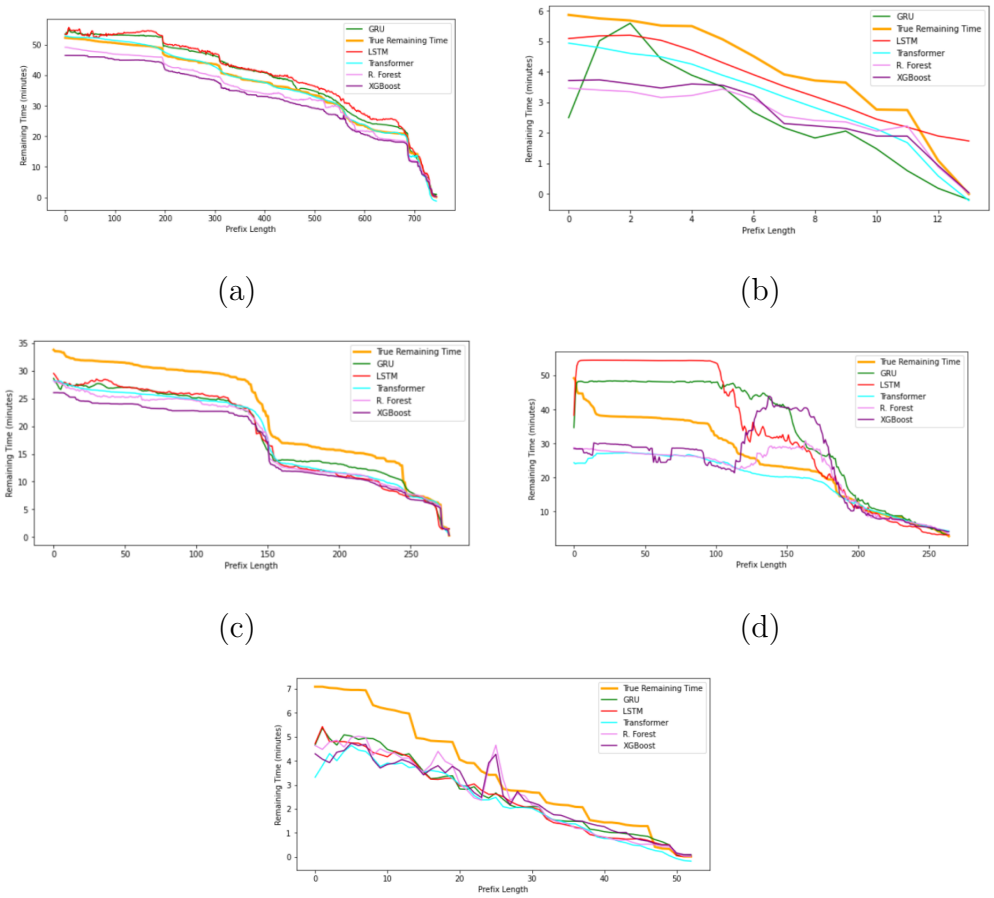

Figure 6.1: Remaining time prediction trend

The table below summarizes the experimental results. It can be seen that the architecture based of the attention mechanism outperforms the other methods for this dataset, providing better and stable performances. Moreover, this approach is the best when dealing with extensive chains and complex dependencies, proving of being capable of handling even very long sequences. More basic solutions like Random Forest and Extreme Gradient Boosting generally provide good results, although inferior to deep models. On the other hand, the possibility of obtaining an indication of the importance of the features brings advantage to the interpretability of the result: as expected, the features that take the highest Mean Decrease Impurity, that is the metric used to determine the importance, are the chain's name, the last log in the prefix and the time (computed as minutes from midnight).

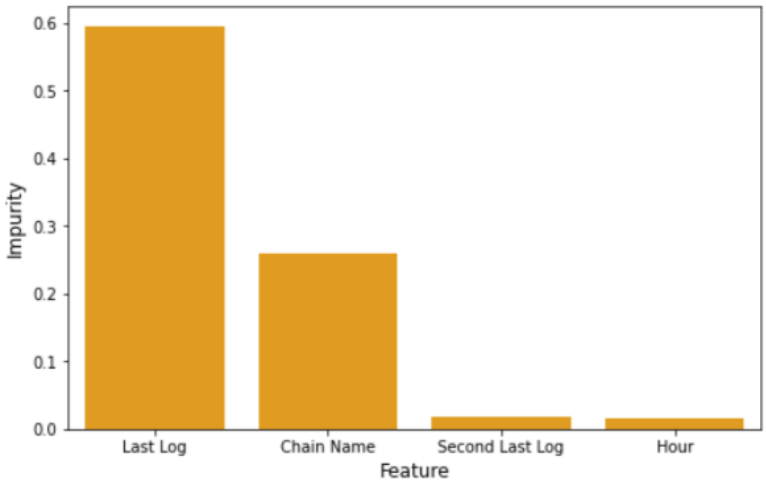

Figure 6.2: Feature Importance according to Random Forest

	MAE	**MSE**
Random Forest	3.47	21.87
XGBoost	3.20	19.9
LSTM	2.95	17.52
GRU	3.06	19.82
Transformer	**2.05**	**7.62**

Table 6.1: Obtained results (in minutes) on the chain dataset

Finally, figure 6.3 shows the MAE obtained with the two best performing models using different prefix lengths. Specifically, we can notice that chains with a shorter prefix are generally more difficult to predict, since by having fewer logs their behavior is poorly defined they are generally more unstable and subject to sudden changes. On the contrary, both models seem to give better results in predicting longer chains, i.e is chains that are in an advanced state, confirming that having a greater number of logs allows for a better generalization. In any case, as expected, the transformer model outperforms LSTM in three groups out of four. However, it is important to note that the number of steps is not strictly linked to the duration of the chain, which depends on the execution time of the individual jobs, which

can also be very short. Similarly, it is not unusual to observe cases with a limited number of logs but whose duration can exceed 40 minutes in certain situations.

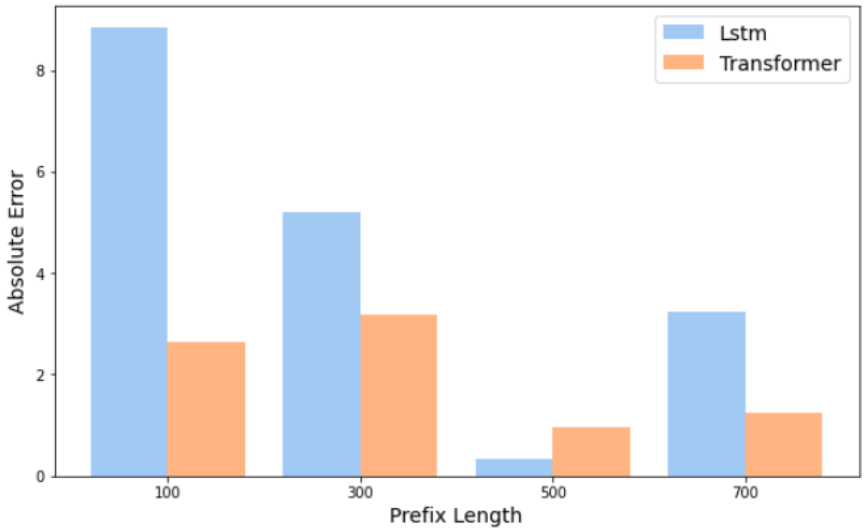

Figure 6.3: MAE values with different prefix lengths

Another aspect that is worth investigating is to test the different models trained only on the individual chains, in order to verify the local results. The main reason for this experiment is that, while a single model is useful, it is extremely costly to maintain. A more flexible and lighter model was therefore considered as an alternative solution.

	Avg. Ex Time	**# Events**	**Transformer**		**LSTM**	
			MAE	**MSE**	**MAE**	**MSE**
Chain-A	60	750	1.46	3.14	2.6	8.7
Chain-B	7	60	1.02	1.46	1.22	2.07
Chain-C	15	55	0.34	0.15	0.56	0.35
Chain-D	30	120	1.8	5.03	2.02	5.31
Chain-E	15	100	0.79	2.02	0.34	0.31
Chain-F	30	330	0.96	1.44	1.18	2.05
Chain-G	8	80	1.20	1.49	0.82	1.06
Chain-H	25	60	0.86	4.56	0.87	4.90
Chain-I	31	455	0.80	0.81	0.83	0.86

Table 6.2: Local results

Table 6.2 reports the results obtained on 9 chains selected from the hundreds of chains that are scheduled and executed every day. Also in this case, the goodness of the attention mechanism is confirmed, even if the gap between transformer and LSTM for these particular chains is very small and for two of these the second model obtains a lower error. In spite of everything, it must be remembered that not all processes are managed in the same way, some chains are monitored more often than others, while some are more sensitive and have errors more easily. Consequently, the best results are obtained with the most optimized chains.

Thanks to the insight provided by the features importance, it is possible to carry out a final experiment using only a subset of the attributes. In particular, we have chosen to use only the name of the chain, the time and the last event available in the log sequence. Consequently, part of the information due to the lack of previous logs is lost. On the other hand, a strong advantage is gained in terms of computational times for both training and inference. For this last case, recurrent neural networks were not used as there is no longer a need to remember the entire sequences. Instead, tree-based models take advantage of this simpler structure: a reduced number of features allows a more efficient subdivision of the space and a lower number of splits.

	MAE	**MSE**
Random Forest	3.08	16.02
XGBoost	3.44	20.10
Transformer	**3.03**	**10.1**

Table 6.3: Obtained results (in minutes) with selected features

In terms of performance, there is a small deterioration for Gradient Boosting and transformer models, while the random forest gets an improvement. As we expected, the results are worse than those obtained with the previous experiment because the global view of the process trend is lost. The transformer, while not exploiting its peculiarities (and becoming a normal feed-forward network with residual connections) still gets the upper hand, mainly due to the embedding layer and therefore to a better management of categorical variables. In fact, one hot encoding remains a limitation in all approaches due to the large increase in dimensionality and sparsity.

Wrapping up, the performances of three categories of models were analyzed. On the one hand the models based on decision trees, which however obtained slightly

worse results, on the other hand, the models based on deep learning, of which two belonging to recurrent neural networks class and the last one focused on the attention mechanism. The latter proved to be better according to the proposed methods, with the disadvantage of being more complex to train and less interpretable.

Chapter 7

Conclusions and future works

In this work, an end to end project was presented. Starting with data acquisition and cleaning, a model for predicting the remaining time was then developed and tested. Finally, the model was deployed and made available to the final user through a simple web application. The activity was carried out in an enterprise context, which led to confront the typical challenges of a real-world dataset. The results obtained were considered satisfactory by the customer, who is now able to monitor his processes in real time and have support in the decision making phase. Although there are still some limitations, the released application allows to obtain good performances with a minimum error in the order of minutes, which is acceptable considering the difficulty of predicting the readiness of human intervention.

The chosen approach was to model the problem in the way in which business process monitoring tasks are treated in the literature. Furthermore, the datasets used in the experiments of the aforementioned papers are often simpler, with a much shorter length of cases and a fairly limited number of events. In our case, however, it was necessary to process thousands of different events and hundreds of dependencies, which led to avoiding the use of classic encoding methods, such as one hot, in favor of a more efficient embedding.

In recent years the number of publications concerning BPM and remaining time prediction, as well as other typical tasks about event logs, have grown rapidly. Among the possible alternatives, it was natural to choose, for this first work, architectures widely used in natural language processing, such as recurrent neural networks and transformers, which require minimal data preprocessing and are able to learn and generalize even long sequences with state of the art results. Among

the biggest limitations, however, is the lack of explainability: these models work as black boxes but a great effort is being made towards obtaining robust visualizations capable of explaining the reasons behind a prediction [36] through the Shapley Values theory. A second limitation lies in the absence of an incremental approach: Adding a new set of events or optimizing chain processes will require a new training phase: collecting new data could therefore be costly in terms of resources and time.

As a future works, there are some directions that are worth investigating to improve and complement this work. First of all, we will try to include information on the time elapsed between one event and the next: It is not unusual in fact that, especially in cases of long delays, a lot of time passes between the arrival of two different logs. This can lead to an underestimation of the remaining time since the prediction will always be the same for the same entries in the event log. The idea is to exploit and adapt time aware neural networks, in particular T-LSTM, to remaining time prediction. This type of network, initially proposed by Baytaset al. in the context of Patient Subtyping [37], does not assume a uniform distribution for times but, on the contrary, allows to take into account irregular intervals due to a modification of the cell. Alternatively, another interesting architecture is the Graph neural network (GNN), which is a relevant research topic and could improve the forecasting of remaining time for chains not yet started through its dependency tree.

As a second direction, we will try to reduce the number of logs needed for each case. In fact, as the number of chains increases, the number of events grows exponentially. However, not all events have the same information content and some are more relevant than others. Hence, it is possible to reduce training and inference time and optimize memory use by selecting only the necessary rows. These assumptions are strongly related to the implementation of the chains themselves, and therefore require a clear domain expertise to be applied.

Bibliography

[1] Di Francescomarino Chiara, Ghidini Chiara, Maggi Francesco Maria, and Milani Fredrik. *Predictive Process Monitoring Methods: Which One Suits Me Best?* 2018.

[2] Dominic A. Neu, Johannes Lahann, and Peter Fettke. *A systematic literature review onstate-of-the-art deep learning methods forprocess prediction*. 2021.

[3] Efren Rama-Maneiro, Juan C. Vidal, and Manuel Lama. *Deep Learning for Predictive Business Process Monitoring: Review and Benchmark*. 2020.

[4] Mirko Polato, Alessandro Sperduti, Andrea Burattin, and Massimiliano de Leoni. *Data-aware remaining time prediction of business process instances*. 2014.

[5] B.F. van Dongen, R.A. Crooy, and W.M.P. van der Aalst. *Cycle Time Prediction: When Will This Case Finally Be Finished?* 2008.

[6] Bevacqua Antonio, Carnuccio Marco, Folino Francesco, Guarascio Massimo, and Pontieri Luigi. *A Data-Adaptive Trace Abstraction Approach to the Prediction of Business Process Performances*. 2013.

[7] Eugenio Cesario, Francesco Folino, Massimo Guarascio, and Luigi Pontieri. *A Cloud-Based Prediction Framework for Analyzing Business Process Performances*. 2016.

[8] Jian Guo, Akihiro Nomura, Ryan Barton Haoyu Zhang, and Satoshi Matsuoka. *Machine Learning Predictions for Underestimation of Job Runtime on HPC System*. 2018.

[9] Arik Senderovich, Chiara Di Francescomarino, Chiara Ghidini, and Fabrizio Maggi. *Intra and Inter-Case Features in PredictiveProcess Monitoring: A Tale of Two Dimensions*. 2017.

[10] Niek Tax, Ilya Verenich, Marcello La Rosa, and Marlon Dumas. *Predictive Business Process Monitoring with LSTM Neural Networks*. 2016.

[11] Manuel Camargo, Marlon Dumas, and Oscar González-Rojas. *Learning Accurate LSTM Models of Business Processes*. 2019.

[12] Nicolò Navarin, Beatrice Vincenzi, Alessandro Sperduti, and Mirko Polato. *LSTM Networks for Data-Aware Remaining Time Prediction of Business Process Instances*. 2017.

[13] A. Khan, H. Le, K. Do, T. Tran, A. Ghose, H. Dam, and R. Sindhgatta. *Memory-augmented neural networks for predictive process analytics*. 2018.

[14] Vincenzo Pasquadibisceglie, Annalisa Appice, Giovanna Castellano, and Donato Malerba. *Using convolutional neural networks for predictive process analytics*. 2019.

[15] Farbod Taymouri and Marcello La Rosa. *Encoder-Decoder Generative Adversarial Nets for Suffix Generation and Remaining Time Prediction of Business Process Models*. 2020.

[16] Aaqib Saeed Zaharah A. Bukhsh and Remco M. Dijkman. *ProcessTransformer: Predictive Business Process Monitoring with Transformer Network*. 2021.

[17] Cheng Guo and Felix Berkhahn. *Entity Embeddings of Categorical Variables*. 2016.

[18] Wahid, Adi, H. Bae, and Y. Choi. *Predictive business process monitoring – remaining time prediction using deep neural network with entity embedding*. 2019.

[19] Joerg Evermann, Jana-Rebecca Rehse, and Peter Fettke. *A Deep Learning Approach for Predicting Process Behaviour at Runtime*. 2017.

[20] J. Theis and H. Darabi. *Decay replay mining to predict next process events*. 2019.

[21] J. Evermann, J.-R. Rehse, and P. Fettke. *Predicting process behaviour using deep learning*. 2017.

[22] Ilya Verenich, Marlon Dumas, Marcello La Rosa, Fabrizio Maggi, and Irene Teinemaa. *Survey and cross-benchmark comparison of remaining time prediction methods in business process monitoring*. 2018.

[23] Wil Van Der Aalst. *Process Mining - Data Science in Action*. Springer, 2016.

[24] G. James, D. Witten, T. Hastie, and R. Tibshirani. *An Introduction to Statistical Learning: with Applications in R*. Springer, 2014.

[25] Tin Kam Ho. *The Random Subspace Method for Constructing Decision Forests*. 1998.

[26] Leo Breiman. *Random Forests*. 2001.

[27] Tianqi Chen and Carlos Guestrin. *XGBoost: A Scalable Tree Boosting System*. 2016.

[28] Pascanu, Tomas Mikolov, and Yoshua Bengio. *On the difficulty of training recurrent neural networks*. 2013.

[29] Hochreiter and Schmidhuber. *Long Short Term Memory*. 1997.

[30] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. *Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation*. 2014.

[31] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. *Attention Is All You Need*. 2017.

[32] Liu Fei Tony, Ting Kai Ming, and Zhou Zhi-Hua. *Isolation Forest*. 2008.

[33] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. *Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling*. 2014.

[34] Diebold, Francis, and Lutz Kilian. *Measuring predictability: theory and macroeconomic applications*. 2001.

[35] Breunig Markus, Hans-Peter Kriegel, Raymond Ng, and Joerg Sander. *LOF: Identifying Density-Based Local Outliers*. 2000.

[36] Riccardo Galanti, Bernat Coma-Puig, Massimiliano de Leoni, Josep Carmona, and Nicolo Navarin. *Explainable Predictive Process Monitoring*. 2020.

[37] Inci Baytas, Cao Xiao, Fei Wang, Anil Jain, and Jiayu Zhou. *Patient Subtyping via Time-Aware LSTM Networks*. 2017.