# Explainable Remaining Time Prediction for Business Processes

Klijn, E.L.

*Award date:*
2020

EINDHOVEN UNIVERSITY OF TECHNOLOGY
Department of Mathematics and Computer Science
Process Analytics Research Group

# Explainable Remaining Time Prediction for Business Processes

*Master Thesis*

Eva Klijn

Supervisors:
dr. ir. D. Fahland
prof. dr. ir. B.F. van Dongen
dr. ir. A.M. Wilbik

final version

Eindhoven, March 2020

# Abstract

Predictive process monitoring is a central practice in business process management that allows for the timely identification of errors, bottlenecks or deviations in process behavior, i.e. actionable information. Established predictive monitoring methods use event data extracted from Workflow Management Systems in which the case is perceived as the primary element of the process and as a result are approached under the wrong assumption that cases behave in isolation. Because of this intra-case perspective, current predictive monitoring techniques are not able to detect inter-case dynamics that do emerge in organizational processes such as batching. In this thesis we address the need for the awareness of inter-case dynamics in the development life cycle of predictive monitoring approaches specifically aimed at remaining time prediction. We propose a first set of techniques that together introduce inter-case dynamics in the remaining time prediction life cycle. We present techniques for identifying sub-sets of cases with high prediction errors that are subject to these inter-case dynamics and we present techniques that use these insights to create inter-case features and derive an inter-case evaluation specifically for inter-case dynamics caused by batching. In an experimental setup we show that by applying this set of techniques, we are able to improve prediction performance and a better evaluation thereof, leading to more adequate and better explainable models and results.

# Preface

This master thesis is the result of my graduation project for the Business Information Systems master at Eindhoven University of Technology, conducted within the Process Analytics group at the department of Mathematics & Computer Science.

First of all, I would like to thank Dirk Fahland for providing me with this interesting opportunity and his guidance throughout this entire project. It is because of his challenging questions, inspiring ideas and incredible feedback that I was able to deliver the work that lays before you today. Thank you for taking the time to listen to all my thoughts and answer all my questions. I would also like to thank everyone from the Process Analytics group for welcoming me on their floor and providing me with an environment that motivated me to push through. I would like to thank my office buddy Rashid in particular, for always being cheerful and putting things in perspective. Also thanks to Vadim for all the help with the PSM implementation and thanks to Anna Wilbik and Boudewijn van Dongen for being part of the assessment committee.

I want to thank my boyfriend Bas for always supporting me and believing in me, and putting up with me when I was stressed like hell. Having to stay indoors because of the Corona virus is not a thrill in itself, but even less so when you have a person sitting next to you in the days towards the deadline of her master thesis. I also want to thank my friends, especially Verena, for the support and the fun distractions that kept me sane. Also additional thanks to Dennis for helping me with all my coding and software problems.

Finally, I would like to thank my family for all their support and believing in me through the course of all my studies and this endeavour in particular.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

For an organization, the efficient and effective execution of its operational processes is essential. To accomplish this, organizations apply *business process management* (BPM) practices to ensure consistent performance and take advantage of improvement opportunities [3], such as cost or execution time reduction. One of the central elements in BPM is *process monitoring*; execution data is recorded and analyzed to assess the performance of the process with respect to a range of priorly chosen performance measures. When this is successfully put to practice, it allows for the identification of errors, bottlenecks or deviations in behavior or execution time, i.e. actionable information. This information is crucial for decision makers such that they can make the right choices in the management of these processes, such as scheduling or resource reassignment.

Process monitoring techniques can be roughly classified into two categories: those applied on already completed process executions - offline process monitoring - or those applied at run time - online process monitoring. A collection of these online techniques are commonly referred to as *predictive process monitoring*[4].

## 1.1 Thesis Context

In a business context, predictive process monitoring (PPM) allows for the timely identification of performance problems such that decision makers can take corrective action as processes unfold. This is accomplished by creating a predictive model based on historic data - in [5] referred to as the offline phase - to make predictions about the future of running cases - the online phase. In this thesis we will specifically focus on the subcategory of PPM approaches that are aimed at predicting the remaining cycle time of a process instance.

From a technical standpoint, the development of PPM is an extensive process that requires well-defined steps. Therefore, most approaches have roughly adopted the Cross Industry Standard Process for Data Mining, more commonly known as CRISP-DM, which is an iterative, adaptive process that helps fit data mining or analytical practices into the general problem solving strategy of a business [6]. This *life cycle* consists of six phases: (1) business and/or research understanding, (2) data understanding, (3) data preparation, (4) modeling, (5) evaluation and (6) deployment.

In the CRISP-DM life cycle as it is applied in PPM research (PPM life cycle), the business understanding is commonly referred to as the *research* understanding and also often approached as such. As a result, almost all works aim at improving the prediction performance of the *state of the art* in some way, for instance by introducing a new method or improving upon an existing one. Often in concurrence, the data understanding takes place, which encompasses the collection and (exploratory) analysis of event data, either openly available or provided by an organization, which is exclusively extracted from Workflow Management Systems (WfMS). In a WfMS, the case - a sequence of events that describe the path of an entity in an organizational process (e.g. an application in a loan application process or a patient in a hospital) - is perceived as the primary element of the process. Next, in the data preparation phase, the data payload of a case in a certain point in the process is lifted to make it interpretable for the modeling technique in the next phase. After a model is built, it is assessed in the evaluation phase by measuring its performance and aligning this with the research objective. If these results are unsatisfactory, the life cycle is (partly) reiterated by applying a *feedback loop* until desirable results are obtained. Once satisfactory results have been obtained, the research objective is considered to be fulfilled and the contribution can be presented to the research community.

We have observed that in some situations, almost all approaches still show a high prediction error. We see this for example in [4], which is a cross-benchmark comparison of 16 primary remaining time prediction methods based on 16 real-life event logs. This benchmark comparison showed that the best results were obtained using a LSTM neural network as was introduced in [7]. However, the application of this approach still resulted in a prediction error of 178.74 days on average for the Road Traffic Fine Management (RF) log. After further inspection, we additionally discovered that the application of a feedback loop remains undiscussed in PPM contributions, even if results showed a high error such as this one.

To uncover the cause of this high prediction error, we have inspected the data of the RF log in the performance spectrum miner (PSM), which is a visual analytics technique that allows users to gain a fine-grained and unbiased insight into the performance progression of a process [8]. The result of this inspection is depicted in Figure 1.1, where we see the progression of cases from the activity *Add Penalty* to the activity *Send for Credit Collection*.

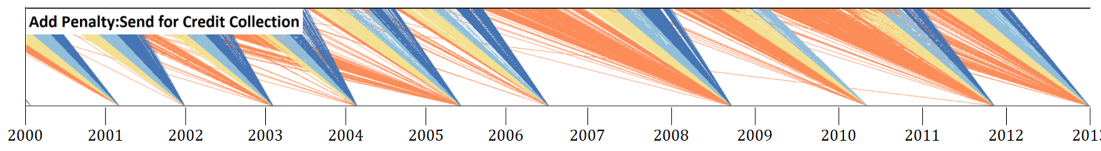

**Figure 1.1:** Performance spectrum visualization of segment *Add Penalty:Send for Credit Collection*

In Figure 1.1 we observe that cases are processed in batches of roughly one year apart, which explains why there is such a high prediction error for this data set. This also shows that cases do not progress in isolation, but they are part of a bigger whole and subject to *inter-case dynamics* such as batching. However, PPM currently applies an intra-case perspective, resulting in intra-case features and aggregates of intra-case error measurements.

## 1.2 Problem Statement & Research Questions

The uncovering of inter-case dynamics reveals the lack of dimensionality in contemporary remaining time prediction approaches, resulting in a life cycle that is adopted under the wrong assumption that cases behave in isolation, leading to inadequate models and inadequate results. Because of this intra-case perspective, the evaluation measures also do not allow for the capturing of these inter-case dynamics, resulting in no trigger of a (meaningful) feedback loop. Therefore, we address the following main research question:

*How can we increase awareness of inter-case dynamics in the life-cycle of remaining time prediction such that it constitutes more adequate and better explainable models and results?*

For this we take a model agnostic approach, such that we can specifically focus on the input features, the output measurements and the feedback loop that connects the two. Therefore we split up the main research question into the following sub-research questions:

1. *How can we identify problematic subsets (and their characteristics) caused by inter-case dynamics that are not captured by current intra-case evaluation methods such that we can trigger a possible inter-case feedback loop?*
2. *How can we use these subsets to derive insights for providing the model with inter-case information, i.e. for deriving inter-case features?*
3. *How can we use these insights to derive a more explainable evaluation of models from an inter-case perspective?*

## 1.3 Research Method, Outline & Results

To address these research questions, we will first make a comprehensive inventory of the phases of the life cycle for remaining time prediction and discuss related work along these phases. We will additionally illustrate the concrete steps that are taken in these phases by using a running example based on a primary method.

Based on the observations we make, we will make an inventory of the shortcomings that are currently present in the remaining time prediction life cycle. Based on these shortcomings we will make an inventory of what is exactly necessary to increase the awareness of inter-case dynamics in contemporary remaining time prediction. This inventory will consist of various methods with which we propose to enhance this life cycle:

1. We first address the feedback loop by providing methods for an inter-case error analysis. These methods will receive the inter-case perspective from the business and data understanding and apply this to the output measurements retrieved from the evaluation phase. By applying these methods, we wish to trigger a meaningful feedback loop that introduces the inter-case perspective to the main phases of the life cycle.
2. Once such an inter-case perspective is obtained, we want to transfer it to the modeling phase such that we can build a model that gives more adequate and better explainable performance results. For this we address the creation of inter-case features by providing a method that leverages this inter-case perspective into a concrete feature interpretable for a remaining time prediction method. We will do this specifically for the inter-case dynamics caused by batching.
3. We also want to introduce this inter-case perspective to the evaluation phase. For this we provide evaluation methods that better respect the inter-case perspective. We again specifically do this for the inter-case dynamics caused by batching.

In the end we apply these methods on a primary approach for remaining time prediction in an experimental setup. We show that we can identify problematic segments subject to inter-case dynamics and thereby trigger a feedback loop. By creating inter-case features based on these insights and using these in the prediction, we show that we can improve the prediction performance. We also show that we can use these inter-case insights to better evaluate prediction performance w.r.t. inter-case dynamics and are thereby able to increase the explainability of the model and the results. In the end we use the best inter-case feature to show that we are also able to improve the prediction performance of other remaining time prediction approaches.

Chapter 2 introduces the necessary background knowledge. Chapter 3 discussed the life cycle of remaining time prediction and Chapter 4 present the shortcomings we observe in this life cycle. Chapter 5 introduces an inventory of the steps we propose to address these shortcoming with, of which the corresponding methods are introduced in Chapters 6 to 8. We evaluate these methods in Chapter 9 and conclude in Chapter 10.

# Chapter 2

# Preliminaries

The remaining time prediction approach that we focus on as well as the reasoning and methodologies that we present draw concepts from both the machine learning and the process mining discipline. This chapter introduces the various concepts that are used in the remainder of this thesis.

## 2.1 Process Mining

Process mining [9] is a discipline that uses event data recorded in Workflow Management Systems (WfMS) to derive useful insights about business processes. In a WfMS, this event data consists of sequences of events that describe the executions of activities of an entity in an organizational process (e.g. an application in a loan application process or a patient in a hospital), also called a *case*.

Each recorded *event* has a number of attributes. Three of these are mandatory, namely the *event identifier* indicating which activity was observed, *case classifier* indicating in which case it was observed and the *timestamp* indicating when it was observed. An event can also carry additional data attributes in its payload, which can be either *static* or *dynamic*. Static attributes or *case attributes* are attributes of which the values are shared for all events belonging to the same case (e.g. the type of loan in a loan application process or the age of a patient in a hospital). Contrarily, dynamic attributes or *event attributes* are attributes which values change across the executions of events (e.g. the status of an application in a loan application process or the specialist that is treating the patient in a hospital).

An event can also carry additional data attributes in its payload, which can be either *case attributes* or *event attributes*. Case attributes are attributes of which the values are shared for all events belonging to the same case (e.g. the type of loan in a loan application process or the age of a patient in a hospital). Contrarily, event attributes are attributes which values change across the executions of events (e.g. the status of an application in a loan application process or the specialist that is treating the patient in a hospital). For analysis, case attributes are in some sense *static* (because they hold through the entire case) and event attributes are in some sense *dynamic*.

A non-empty sequence of events $\sigma = \langle e_1, ..., e_n \rangle$ that all refer to the same case make up a *trace* and a set of completed traces together make up an *event log* $L$. For process prediction, we want to make predictions for cases of which the outcome is yet unknown, meaning we need a set of uncompleted traces. An incomplete trace or *prefix* can be created by taking the first $k$ events from a complete trace using a prefix function $hd^k(\sigma) = \langle e_1, ..., e_k \rangle$, with $k \leq n$. If we take all possible prefixes of our input event log $L$, we end up with the *prefix log* $L^*$ of $L$.

In Table 2.1 we have included a simplified extract from the BPIC 2017 log that originates from a loan application process as an example. Each row corresponds to a single event execution, which, in this log, relates to either of two cases: A_17 and A_98. The other case attributes are Amount and Loan Goal and the event attributes are Activity, Time and Resource. Attributes are typically also distinguished based on data type such that they can be preprocessed accordingly for the prediction model. We distinguish between numerical and categorical data types. In Table 2.1, Amount and Time are numerical attributes and Case ID, Loan Goal, Activity and Resource are categorical attributes.

| Case ID | Amount | Loan Goal | Activity | Time | Resource |
|---|---|---|---|---|---|
| A_17 | 15.000 | Car | Create Application | 21-5-2016 13:10:11 | User_1 |
| A_17 | 15.000 | Car | Accepted | 21-5-2016 13:10:11 | User_1 |
| A_17 | 15.000 | Car | Complete | 21-5-2016 13:17:36 | User_1 |
| A_17 | 15.000 | Car | Validating | 24-5-2016 14:05:29 | User_19 |
| A_17 | 15.000 | Car | Pending | 30-5-2016 13:08:51 | User_118 |
| A_98 | 25.000 | Home Impr. | Create Application | 15-7-2016 12:36:20 | User_49 |
| A_98 | 25.000 | Home Impr. | Submitted | 15-7-2016 12:36:20 | User_49 |
| A_98 | 25.000 | Home Impr. | Accepted | 15-7-2016 12:40:18 | User_49 |
| A_98 | 25.000 | Home Impr. | Complete | 15-7-2016 12:45:01 | User_49 |
| A_98 | 25.000 | Home Impr. | Validating | 21-7-2016 16:13:09 | User_120 |
| A_98 | 25.000 | Home Impr. | Denied | 27-7-2016 13:58:56 | User_75 |

**Table 2.1:** Extract of BPIC 2017 log

The executions of the first case A_17 result in the (simplified and abbreviated) trace $\sigma = \langle Cre, Acc, Com, Val, Pen \rangle$. The prefixes of solely this trace would be $\langle Cre \rangle$, $\langle Cre, Acc \rangle$, $\langle Cre, Acc, Com \rangle$, $\langle Cre, Acc, Com, Val \rangle$ and $\langle Cre, Acc, Com, Val, Pen \rangle$.

## 2.2 The Performance Spectrum

The performance spectrum (PS) is a data structure and analytics technique introduced by Denisov et al. [1] that allows users to gain a fine-grained and unbiased insight into the performance progression of a process. This is achieved by the mapping of all observed flows between two processing steps over time without any prior aggregation, as illustrated in Figure 2.1.

Let us consider event log $L$ and two activities that are observed to occur in sequence, $a$ and $b$. The PS describes how cases transition from $a$ to $b$ over time $t$, also called the *segment* $(a, b)$. Whenever in a case $a$ is directly followed by $b$, i.e. if we observe $\langle ..., a, b, ... \rangle$ in the corresponding trace, we observe an occurrence of this segment taking place from time $t_a$ (moment of occurrence of $a$) to time $t_b$ (moment of occurrence of $b$). To visualize the occurrences of these segments, the life cycle events $a$ and $b$ are fixed as $y_a$ and $y_b$ on the $y$-axis and the time $t$ is projected on the $x$-axis, as also illustrated in Figure 2.1. Each occurrence of a segment is then plotted as a line from $(t_a, y_a)$ to $(t_b, y_b)$.

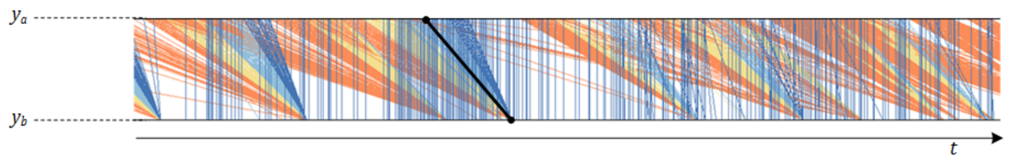

**Figure 2.1:** Detailed performance spectrum example of single process segment of Road Traffic Fine Management log

If we add a line from $(t_a, y_a)$ to $(t_b, y_b)$ for all observations $\langle ..., a, b, ... \rangle$ in all cases of $L$, then we get a visualization depicting all occurrences of segment $(a, b)$, of which an example for the Road Traffic Fine Management log is illustrated in Figure 2.1.

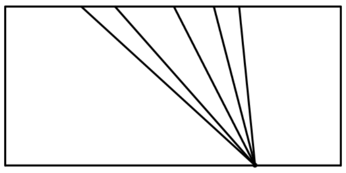

**Figure 2.2:** Schematic representation of batching pattern

The arrangement of the lines in Fig. 2.1 forms distinct patterns which can be classified by a taxonomy [1]. In this thesis we will revisit one of these patterns in extensive detail, which is the *batching* pattern. The taxonomy defines batching as FIFO behavior where batching occurs at either the preceding step $a$, the succeeding step $b$ or both. We will focus on the case where batching occurs at the succeeding step $b$, called `batch(e)` [1]. On the performance spectrum, this will show up as lines starting at various points on $y_a$ and converging to a more or less single point in time on $y_b$, as can also be identified in Fig. 2.2.

## 2.3 Machine Learning

Machine learning is a scientific discipline centered around algorithmic techniques that allow the computer to learn from data [10]. A machine learning algorithm requires a data set containing past observations, such that it can synthesize the underlying relationships to either predict a specific outcome or *label*, or to find patterns in the data, referred to as *supervised*- and *unsupervised learning*, respectively. For the purpose of predictive process monitoring, in which we aim to predict an outcome, we will solely focus on supervised learning.

A supervised learning algorithm requires a data set of past observations that can be used as training data. This is the data that the algorithm will learn from, such that it can later be applied to new observations. Each entry in such a data set consists of a set of independent variables or predictors $x = (x_1, ..., x_n)$ and a dependent variable or label $y$. The goal is to fit a prediction model such that it provides a mapping $x \to y$ with a minimal error, which can later be used to assign labels to new data. For a supervised learning task, two types of models are defined, namely a regression model and a classification model. The former is used for predicting continuous outcomes and the latter is used for predicting discrete outcomes. While this thesis is about remaining time prediction, which constitutes a continuous outcome and therefore requires regression models, we solely wish to evaluate and improve these the methods that use these models and do therefore not care for the type of regression model. However, we do touch upon some of them in the following chapters, such as random forest and extreme gradient boosting.

## 2.4 Predictive Process Monitoring

In the process mining discipline, there is a specific branch that focuses on operational support or the monitoring of a process. Here, the performance of a process is assessed with respect to certain performance measures by means of the recording and analyzing of execution data. When this is applied at run time to retrieve insights about the future of process executions, it is often referred to as online process monitoring or *predictive process monitoring*.

Predictive process monitoring (PPM) is centered around predicting an outcome or a certain performance measure of a case in the future, based on its history. The moment at which a prediction is made determines what is known about the history of the process execution, since the moment at which the prediction is made determines how much information is available for the prediction: the prefix already observed and the information carried by this prefix. Based on this knowledge, the prediction model aims to reason about a certain outcome at a moment in the future, which can be a continuous outcome, such as the time until completion, or a discrete outcome, such as the acceptance of a loan application. The development of the component $X$ that performs this predictive reasoning is at the core of PPM and also subject of this thesis.

Chapter 3

# The Life Cycle of Remaining Time Prediction

The aim of this chapter is to describe the current *state of the art* of the development of component *X* for *remaining time prediction* methods. For this, the emphasis will lie on the development *life cycle*: the steps followed to come from a problem statement and some input data to a proposed solution. As mentioned in the introduction, we take the CRISP-DM life cycle as a point of reference. CRISP-DM stands for *Cross Industry Standard Process for Data Mining* and is a referred to as a robust, well-proven industry standard process for data mining and analytics projects [11]. Its life cycle consists of six phases and is depicted in Figure 3.1.

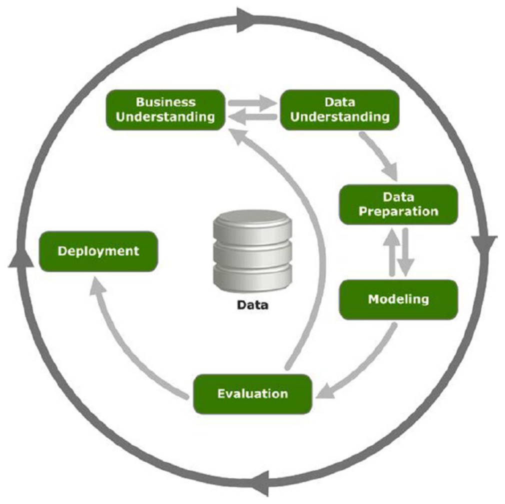

**Figure 3.1:** The CRISP-DM life cycle

To clearly illustrate this life cycle and also other concepts in the remainder of the thesis, a running example is introduced in Section 3.1. Sections 3.2 to 3.7 elaborate on the first five phases of the CRISP-DM life cycle and its application in remaining time prediction. Section 3.8 discusses the iterative aspect of this life cycle depicted by the anticlockwise arrow in the CRISP-DM model, which we will refer to as the *feedback loop*. The last phase - deployment - is left out intentionally, since this is not commonly covered in research but mostly left to industry.

## 3.1 Running Example

In the following sections we will discuss how existing literature on PPM addresses the different CRISP-DM life cycle phases. To make the discussion and limitation of current approaches to develop PPM models concrete, we illustrate for each phase which steps have been performed and documented by a concrete PPM learning method and data set as a running example. In this section, we explain how we selected the method and the data set on which we illustrate the approach.

In the remainder of this thesis, this running example will additionally serve to support our reasoning, illustrate concepts and methodologies and evaluate those.

#### Approach

For the selection of the approach, we will turn to a benchmark for remaining time prediction methods [4], of which the authors have made most of their implementations available at `https://github.com/verenich/time-prediction-benchmark`. The benchmark provides a clear overview of the approaches that have been introduced over the years and its focus is to present a comparison thereof. This comparison solely encompasses methods from primary studies and their techniques exclusively relate to prefix bucketing and prefix encoding, which are data preparation steps specific to process prediction problems which will be elaborated on in Section 3.5 and modeling.

The benchmark covers process-aware approaches, such as transition systems and stochastic Petri nets, and non process-aware approaches, such as deep learning and machine learning algorithms. Since the second category was more easily accessible and implementable, we selected a machine learning based running example. Additionally, the machine learning approaches follow a more "traditional" data mining workflow than the other approaches and will therefore serve well as an illustration of the CRISP-DM life cycle. The available machine learning implementations included configurations of different prefix bucketing techniques, prefix encoding techniques and algorithms.

Since the goal of this running example is to demonstrate the process of developing a remaining time prediction approach that improves upon the current state of the art in some way and the benchmark contains a collection of approaches that have all done just that, we randomly selected one configuration. In Sections 3.5 and 3.6 we elaborate on the bucketing-, encoding and modeling technique that we selected for our running example.

#### Data Set

In addition to the implementations, the authors of the benchmark have also made available the pre-processed data sets that they have used. We have chosen to use the log which had the highest prediction error in the benchmark results, which is the Road Traffic Fine Management Log [12].

## 3.2 Business Understanding

The initial phase covers the *business understanding* or, as it is in some cases called, *research understanding.* In this phase, the context of the problem is defined, as well as the requirements and constraints under which the problem is to be solved [11]. In research, this is usually an open problem that has not been investigated before or contributions to existing solutions.

In remaining time prediction, various types of works have been introduced and almost all aim at improving upon the prediction performance of the state of the art in some way. Most works focus on the prediction accuracy, but some also focus on other performance measures, such as the scalability of a solution [13]. The approaches that have been introduced over the years all roughly fall into one or more of the following three categories: they aim to improve prediction performance by (1) introducing (a combination of) modeling techniques novel in the area of remaining time prediction [14, 7, 15, 16, 17, 18], (2) extending existing techniques by incorporating a new combination of encoding or bucketing techniques [19, 20, 16, 17] or (3) enriching the data, such as the inclusion of context data in [19] or the extraction of inter-case data in [21]. Some approaches do not fall into exactly one of these categories but introduce a combination of these solutions.

As for the actual business understanding, nothing is mentioned in PPM literature. What we do see is that these prediction methods are almost all exclusively approached under the view of a single process in which each process execution (case) is handled in isolation from all other cases. These methods are therefore also directed towards the predictions of single case outcomes and their performance is also measured as such.

Running Example (from [4])

**Understanding:** Understanding of the research that has been done in the area of remaining time prediction and seeing an opportunity for improvement through the application of a novel combination of bucketing and encoding techniques.

**Objective:** Improving upon the remaining time prediction accuracy of the current state of the art.

## 3.3 Data Understanding

The research or business understanding is usually covered in concurrence with the second phase: *data understanding*. This phase encompasses the collection, description, qualitative- and exploratory analysis of data [11]. In a business context, the selection of data is fairly straight forward, since in for an organization aiming at remaining time prediction of their process, it is inherently known which data to consider and design the method for. However, the collection of this data is rather complex as most data is scattered across many systems. Contrarily, in research, there is no limitation to one specific process/data, but one can choose from whatever is available. In this case, the complexity does not lie in the collection of the data but in the selection: how to actually select an adequate set of event logs for a specific PPM approach. The next paragraph will elaborate more on this with respect to remaining time prediction. Finally, further qualitative- and exploratory analysis is done to ensure that the data quality and contents are sufficient to reach the business or research goal. While this data understanding step is usually never mentioned in research unless it serves the purpose of the study, it is of equal importance in academia as it is in industry.

For data selection in remaining time prediction, but also for PPM in general, currently no standard procedure exists. Some works include a motivation when introducing their data set selection, while others leave this out entirely. This variation across the different approaches seems to mainly depend on the research problem. Works that aim at improving the model itself usually do not motivate their selection choices, they mainly aim for a reliable and generalizable solution and therefore try to include multiple data sets, such as has been done in []. Approaches that focus more on the data itself typically select data of processes which show characteristics that indicate a possible benefit from the approach. Such as for example in [21], where a hospital event log is used for the inclusion of inter-case features (the amount of patient at an ICU at a specific moment in time as an additional predictor) or in [19], where an event log of a harbour is used for the inclusion of context features (the amount of containers in the harbour at a specific moment in time as a predictor). For this second category, the data understanding phase is most likely more extensive than for the first category.

Running Example (from [4])

**Collection & motivation:** As stated in Section 3.1, a data set of an Italian traffic fine management process was chosen. This data set is openly available at [link] and it was chosen to demonstrate an increase in prediction accuracy by applying a proposed new method.

**Description:** The Road Traffic Fine Management (RF) log consists of 150,370 cases, 561,470 events and 11 unique activities over a period of 12 years. Cases in the log have a mean length of 3.734 activities and a mean duration of 341.676 days. In the log, for each trace 4 case attributes and 10 event attributes are recorded.

**Preparatory analysis:** For the qualitative analysis, the data was checked for missing values and inconsistencies. For the exploratory analysis, we assume that some basic process discovery was applied to get an overview of the process, its attributes and its statistics.

## 3.4 Data Preparation Phase

The third phase - *data preparation* - consists of a set of tasks to prepare the data for building a model. The terms for the different tasks vary significantly across the literature, therefore here we will use multiple sources and extract a general description. Data preparation usually starts with *data cleaning*, which is followed by a collection of procedures which are commonly put under the denominator *feature engineering*. For standard data mining problems, this is concluded by *feature encoding*, where data is converted to make it interpretable for the algorithm used in the next phase. For problems that concern event data, additional steps are necessary to convert the sequences of events to features of constant size such that they can be used as input for a prediction model. Since these procedures are very specific to remaining time prediction and PPM in general, we discuss them in Section 3.5.

Additionally, it is important to state here that with respect to remaining time prediction, the implementation of the data preparation phase varies a lot across the different proposed methods. Most process-aware approaches do not use any data attributes apart from control-flow data, whereas non process-aware approaches almost always include (a selection of) non control-flow data attributes. Therefore, for the first category, this phase is often entirely skipped, while for the second category, a significant amount of time is spent here.

### 3.4.1 Data Cleaning

The data cleaning is usually a result of the qualitative analysis from the data understanding phase. There may be values missing, entries could be erroneous or inconsistent across the data set; data cleaning is a collection of techniques that deal with these deficiencies [22]. These are very custom procedures and not explicitly discussed in PPM research. While this is possibly a relevant topic for improving remaining time prediction approaches, it is outside the scope of this thesis and therefore we will not discuss it any further.

### 3.4.2 Feature Engineering

Where data cleaning aims to improve the usability of the data, feature engineering aims to increase its usefulness. In research, a lot of different terms and explanations are given for these data preparation procedures, therefore, for simplicity, we will put all these techniques under the denominator feature engineering: the transformation, selection, extraction and creation of features from raw data into a format that is suitable to deliver sufficient and useful results [23, 22].

#### Data Reduction

In order to improve the quality of the data, its size often needs to be reduced. This is firstly achieved by applying *feature selection* techniques, which evaluate the relevance of each feature with respect to the label, based on which irrelevant features can be discarded. Example procedures that have been observed in PPM research are the elimination of attributes that cross-correlate with other attributes or the elimination of attributes that have constant values across all entries, as has been done in [4].

Another way to reduce the amount of information in a data set, is by applying *feature transformation* techniques. These typically entail the replacement of original features with a function thereof, discarding unnecessary information in the process. An example of this application can be found in [4], where rarely appearing categorical values in a set of many are marked as 'other', reducing the amount of information for that data attribute.

While these data reduction techniques are valuable to mention in a benchmark, they remain undiscussed in other prior studies on PPM. While this is usually left out because it does not serve the prior study's purpose, it does not benefit the reproducibility.

#### Data Extension

A second way to improve the usefulness of the data is by enriching it with new features. *Feature extraction* is a technique that closely resembles feature transformation, but instead of replacing the original feature, an additional feature is created. Moreover, instead of only using a single data attribute to create new features, a combination of attributes can be used, making it a bit more sophisticated than simple transformation procedures. Examples are the extraction of the weekday or hour of the day from the timestamp [4, 7], or extracting the time since the last event by using the current and previous timestamp, as have been applied in [7, 14].

*Feature creation* is a more complex way to extend the data with additional features. Where feature extraction techniques can more or less be applied automatically, feature creation is applied manually. It demands extensive analysis of the data and usually encompasses multiple transformation steps, sometimes using additional tooling. In [21], a set of techniques is presented that together realize the derivation of a so-called inter-case feature. The resulting feature reflects for each case the amount of cases that are in temporal proximity at that moment in time.

In this thesis, we are going to add new features to our prediction model by applying a combination of these aforementioned types of feature creation steps.

### 3.4.3 Feature Encoding

In standard data mining problems, the last part of the data preparation phase encompasses *feature encoding*, which are a set of procedures to make the data set interpretable for the (machine learning) model. This is a fairly simple process of converting between various data types to improve their portability with respect to the algorithm that is to be used [22]. Operations are isolated to one specific data attribute and often include conversions from categorical or time series data to numerical data.

One of the most frequently used encoding techniques is *one-hot encoding*, which is also applied in [4]. When applying one-hot encoding, each categorical feature is converted into multiple columns, each being one possible value of that feature. The value of each feature is then expressed by a 1 in the corresponding value-column and zeros in all other columns of that feature.

Throughout this thesis, we will adhere to these standard approaches for feature encoding.

Running Example (from [4])

The approach chosen was built in a Python 3.7 environment and therefore the data set needed to be prepared accordingly. For this, the authors [4] first converted the XES formatted log into a CSV format to be able to import it in Python. Since the reasoning/analysis behind the data preparation process was not elaborated on, the following will also solely describe the processing itself.

**Data cleaning:** Missing values and inconsistencies had already been removed in the pre-processed log.

**Feature Engineering:**

- Feature selection: The attributes *matricola*, *paymentAmount* and *totalPaymentAmount* were eliminated based on cross-correlation with other attributes or constant values across all entries.
- Feature transformation: For the attributes *vehicleClass*, *lastSent*, *notificationType* and *dismissal*, values of entries that rarely appeared were marked as 'other'.
- Feature extraction: The attributes *month*, *weekday* and *hour* were extracted from the timestamp and the attribute *duration* was extracted by subtracting the timestamp of the previous activity from that of the current activity.
- Feature creation: No feature creation was applied.

**Feature encoding:** The numerical features were kept as is and categorical features were encoded using one-hot encoding.

## 3.5 Data Preparation for Remaining Time Prediction

Contrary to traditional data mining practices, process prediction problems demand additional steps to prepare the data for the modeling phase. These procedures are much more extensive and complex and very specific to the area of remaining time prediction, and PPM in general. While we still attribute this to data preparation, it substantively deviates from the traditional CRISP-DM life cycle and is therefore separately described in this section.

A first step is *prefix extraction*, where an event log is converted into a *prefix log*, containing all possible prefixes of each trace in the log or a subset thereof [24]. This is succeeded by prefix bucketing and prefix encoding, which are discussed in the next subsections.

### 3.5.1 Prefix Bucketing

In most existing remaining time prediction methods, prefixes are divided into several smaller training data sets, i.e. *buckets*, for which a separate prediction model is trained. The intention is that each of these smaller training sets contain cases with similar properties, which can have a positive effect on the prediction quality [25]. For remaining time prediction, mostly the following approaches are considered:

- *Single bucketing.* Considers all prefixes to be in the same bucket and thus trains a single prediction model on the entire prefix log, as has been applied in [20, 21, 7].
- *Prefix length bucketing.* Training data is divided into buckets such that each bucket contains prefixes of the same length, as has been applied in [17].
- *Cluster bucketing.* A clustering algorithm is applied to group the prefixes into clusters, which are represented as buckets. This approach has been used in [13, 19].
- *State bucketing.* This approach only applies when a process-aware method is used as a prediction model. Here, all prefixes are divided into buckets that correspond to the state that they are in and for each state, a separate prediction model is trained. This approach was used in [16].

In this thesis, we remain agnostic to the type of prefix bucketing. However, since we will not use implementations of process-aware approaches, we will not consider state bucketing.

### 3.5.2 Prefix Encoding

The last step of data preparation encompasses *prefix encoding*, where a function is applied to convert each prefix into an abstraction thereof. Various approaches have been introduced, of which a selection is usually made depending on the bucketing method in the preceding step and the modeling method that is chosen in the subsequent phase.

In process-aware approaches, prefixes are usually converted into an abstraction that can be mapped to a state in a model. This was first introduced in [15], where so-called log abstraction functions are used to convert prefixes into state representations, which can be sequences, sets or bags over one or more event properties. Similar abstraction functions are used in [13, 19, 16, 14], where the original approach was extended with additional modeling configurations. Alternatively in [17], a so-called backtracking algorithm [26] is used to determine the current state of each trace in the log.

In non process-aware approaches, prefixes need to be converted into feature vectors of constant size such that they can be used as input for a statistical model or machine learning model. The encoding of sequences into features for remaining time prediction was first introduced in [27], where control-flow information is encoded into feature-outcome pairs. [28] is a first work that incorporates the data payload by treating traces as complex symbolic sequences and introduces different types of prefix encoding techniques, which have been refined in [24]:

- *Last state encoding.* Only event attributes of the last $m$ events are considered, as has been applied in [20, 16].
- *Aggregation encoding.* All events since the beginning of the trace are considered, but various aggregation functions are applied to keep the feature vector size constant, as has been applied in [20].

- *Index-based encoding.* Event attribute values of all events are concatenated such that no information is lost, as has been applied in [17].

While we do consider prefix-length related features in this thesis, we remain agnostic to the type of prefix encoding.

Running Example (from [4])

Since the benchmark paper presented primary approaches that all have improved upon prediction accuracy at the time of their publication, we have chosen one of these approaches as a running example, with the following configurations:

**Prefix bucketing:** After converting the event log to a prefix log using all possible prefixes of all traces, prefix-length bucketing was applied.

**Prefix encoding:** Prefixes were encoded using aggregate encoding. For this, each numerical event attribute was aggregated by taking the mean, maximum, minimum, sum and standard deviation of all observed values. Each categorical event attribute was aggregated by counting the number of times each value has appeared for each attribute.

## 3.6 Modeling Phase

In the *modeling* phase, the prepared data is used to build a model in order to actually solve the business problem or fulfill the research objective. For this, a modeling technique needs to be selected first. In a business context, this must be a technique that is proven robust and satisfies the requirements and constraints of the business goal. While the same applies in PPM research, the contribution often concerns the modeling technique itself. In that case the technique is not actually chosen but developed.

Over the years, a lot of different techniques have been introduced in the area of remaining time prediction. These approaches are often categorized on their process-awareness. Process-aware approaches use an explicit representation of a process model in the prediction, such as transition systems [15], stochastic Petri nets [14] or queuing models [18]. Non process-aware approaches do not use an explicit representation of the process model, such as machine learning algorithms [21, 20], neural networks [7] or statistical models [29, 30]. A third category of approaches applies a combination of the two, i.e. hybrid approaches, where decision points in a process model representation are enriched with non process-aware models (such as statistical models or other) [19, 17, 16].

When a modeling technique is chosen, the model needs to be built, evaluated and tuned. For process-aware approaches this often encompasses the mining of transition systems (or other process model representations) and evaluating several metrics, such as fitness and precision [9]. As a result of this evaluation, the model can be tuned further to find the model that delivers the best results. For the non-process aware approaches that encompass machine learning, the building of a model is often known as training. Features from the data preparation phase are used as input to a machine learning algorithm that tries to learn a model that generalizes on the input features to fit a certain label [11]. Subsequently the model is evaluated using an accuracy metric such as the MAE or RMSE (because for remaining time we are only dealing with regression models) and can then be tuned using grid search or cross validation in order to find the model that delivers the best results.

This evaluation must not be mistaken with the evaluation in the next phase. In this phase, evaluation is exclusively performed to ensure good model performance, whereas in the evaluation phase, the approach itself is evaluated with respect to the business or research goal.

In this thesis we will use two machine learning algorithms, i.e. random forest and extreme gradient boosting. These are both ensembles of learning trees (in our case regression trees), which are meta-algorithms that combine several regression trees into one predictive model which is proven to yield better predictive performance [11]. We will also use grid search to tune our model parameters, for which a "grid" of values is specified and models are built for each value combination to find the optimal one [11]. However, we only want to focus on the input and the output side of these models. Therefore we take a model agnostic approach and treat these models as a black box in the remainder of this thesis.

Running Example (from [4])

In the benchmark experiment, the machine learning based approaches were all based on random forest or extreme gradient boosting. Since the latter was the best performing, we chose the extreme gradient boosting algorithm for our running example. For this algorithm, the Python library for XGBoost was used, which was already available in the benchmark implementation.

We split the RF log into a training and a test set using a temporal split. For this the cases are ordered according to their start time and the first 80% are used for training and the last 20% are used for testing, resulting in $L_{\text{train}}$ and $L_{\text{test}}$.

We used grid search (also available in the implementation) to tune the parameters of the gradient boosting algorithm using different chucks of $L_{\text{train}}$. When the optimal parameters were found, we used the entire training data $L_{\text{train}}$ for training the final model. This results in remaining time prediction model $RM_{p,a,x}$, based on prefix bucketing, aggregate encoding, and extreme gradient boosting, or method $(p, a, x)$ in short.

## 3.7 Evaluation Phase

When a model has been built that delivers the desired performance, it is assessed in detail in the *evaluation* phase, with the emphasis on the business or research goal. When the results of the evaluation are not desirable, (part of) the cycle is reiterated based on some form of error analysis, which we will call the *feedback loop*. Alternatively, in the *deployment* phase, the final model is put to production and is thereafter monitored and maintained in order to consistently guarantee desirable performance and validity.

In evaluation procedures, the goal is to assess whether the business or research goals are met. While in PPM practice, the goal is to select and tune the best performing method, in research the goal is usually to improve upon the performance of the current state of the art. In both cases, careful evaluation is crucial. To do this, the new method is compared to a *baseline* to verify an increase in performance. In research, the choice of this baseline method heavily depends on the proposed solution under which the research problem is to be solved. Based on the types of solutions that have been proposed in PPM research, as have been elaborated on in Section 3.2, different types of baseline selections have been been applied. When the goal is to introduce a new modeling method or improve upon an existing one, it is usually compared to similar methods or the method that it aims to improve upon, respectively. When the proposed solution is directed towards the configuration of input data, the method remains fixed, while the proposed data configuration is compared to a standard configuration.

When a baseline method has been established, the proposed solution can be tested. This is accomplished by running both the baseline- and proposed prediction model(s) on a set of unseen data and measuring their performance, which, in PPM, is usually assessed in terms of accuracy. In the case of the prediction of continuous variables, such as the remaining cycle time, accuracy is assessed using an error metric such as the mean absolute error (MAE) or the root mean squared error (RMSE). For both of these metrics, first the error is calculated for each prediction separately, after which all of error values are aggregated into a single value by calculating some form of the mean, depending on the metric in question.

In this thesis, we will later evaluate the performance of various features and various methods w.r.t. a baseline (this running example). We will do this both w.r.t. the MAE but also w.r.t. other performance measures we will introduce later on.

Running Example (from [4])

As stated in Section 3.1, the approach for this running example is a primary study and was an improvement upon the current state of the art at the moment of its publication. Therefore a baseline is omitted and solely the results of the running example will be presented.

The model was trained using the first 80% of the cases and tested using the last 20% of the cases. The model's accuracy was evaluated using the MAE, of which the results are depicted in Table 3.1, for each corresponding prefix-length.

| Prefix length | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| MAE | 325.10 | 176.38 | 227.38 | 202.92 | 47.18 |

**Table 3.1:** MAE in days per prefix-length for the running example

## 3.8 Feedback Loop

When the results of the evaluation are unsatisfactory, some form of error analysis is performed and (part of) the cycle is reiterated. This process is not formalized and usually driven by intuition. In most cases, the business- or data understanding is revisited, which commonly leads to additional operations or changes in the data preparation or modeling phase. In other cases, a different evaluation method might be necessary, such that it is better aligned with the research problem. In research, reiterations are usually not considered to serve the purpose of the study and are therefore never reported on. However, developing a prediction model intrinsically is an iterative process, so for most methods, reiteration of the cycle will almost definitely have been part of its development.

Running Example (from [4])

Table 3.1 presents the final results of the running example, which are also the results under which the authors of the corresponding work considered their research problem to be solved. This means that they did not deem another feedback loop necessary and all possible reiterations have evidently been executed prior to obtaining these results. We made use of the word "possible", because nothing is actually reported on reiterations.

Moreover, while the authors did solve their research problem, we cannot oversee the fact that these results still show a very high error. In Figure 3.2 we have depicted a histogram with the total duration of all cases in the test set. While we can observe that there are quite some cases that have a duration $> 400$ days, the majority of the cases have a duration $< 200$ days. From this it is clear that even an MAE of 100 days would already be unacceptable. However, Table 3.1 shows that most errors are far above 100 days. These two observations lay the groundwork for the next chapter.

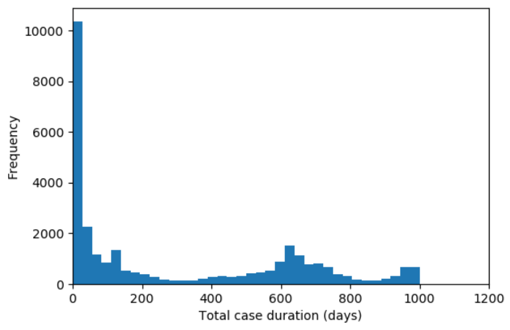

**Figure 3.2:** Total duration of cases in the test set of the RF log

## 3.9 Life Cycle for Remaining Time Prediction

In the preceding sections we have made an inventory of the different steps that are currently executed in the development process for remaining time prediction methods along the CRISP-DM life cycle. We have adopted this life cycle and made some modifications to embed established steps for the development of remaining time prediction methods. The resulting overview of the life cycle for remaining time prediction is depicted in Figure 3.3.

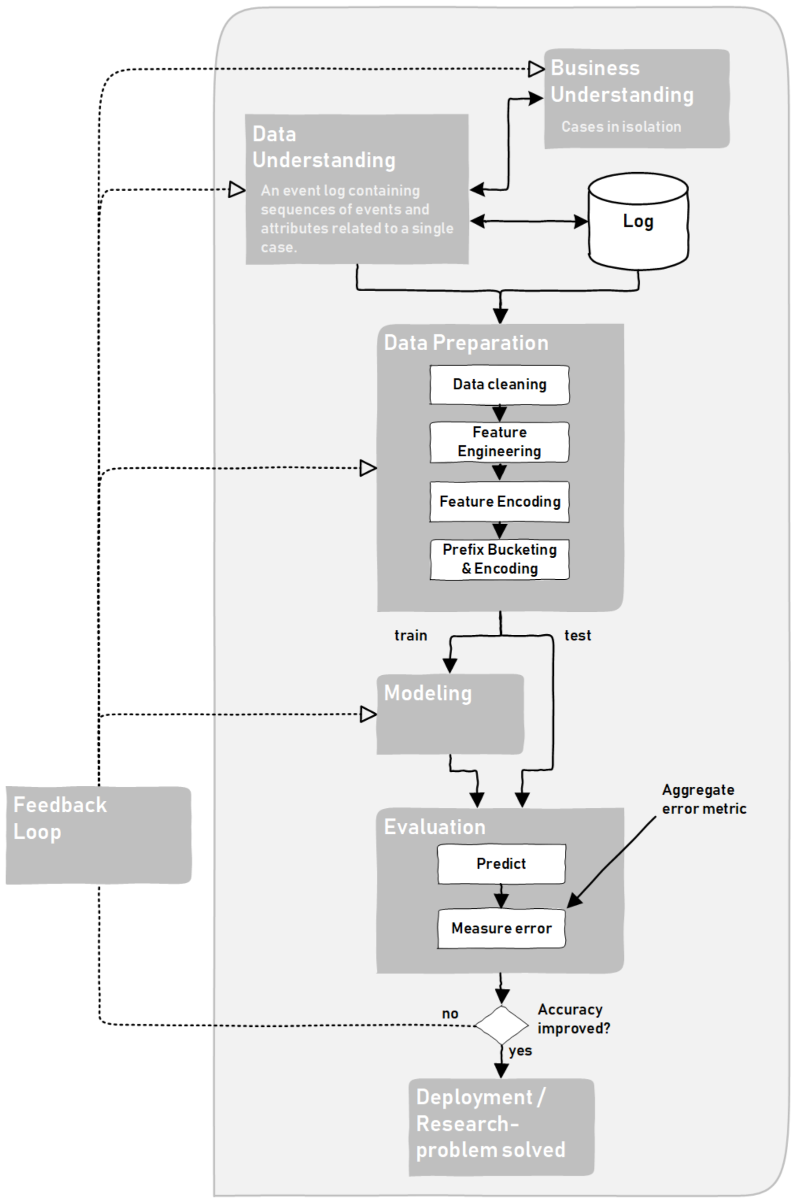

**Figure 3.3:** The current life cycle for remaining time prediction

Chapter 4

# Shortcomings of Contemporary Remaining Time Prediction

In the previous chapter, the development process of remaining time prediction methods was presented along the phases of the CRISP-DM life cycle. What we first have observed is a lack of (reporting on) the feedback loop. In remaining time prediction, and research in general, results are never obtained in a single go and feedback loops are part of a development process by default. However, as not only observed in the running example but also in the other primary studies, the application of a feedback loop is never elaborated on, let alone mentioned in PPM. While one could argue that it is a "hidden" step of which the reporting does not serve the purpose of the study, this iteration in the CRISP-DM cycle is still there for a reason. It serves as a link from the evaluation back to the business- and data understanding and documentation of its underlying reasoning can provide valuable insights into the decision making process.

This brings us to our second observation. We have roughly observed that all primary studies discussed were aimed at introducing novel prediction methods that return a higher accuracy than existing ones. For most works, this resulted in a small increase in accuracy compared to existing approaches. While there was achieved what was aimed for, the running example still shows a significant error, as was illustrated in the previous chapter. Specifically, when viewing the results for the RF log of all primary methods presented in the benchmark, the best performing method still had an average MAE of 178.74 days against a mean case duration of 341.68 days [4]. This indicates that some important information might be overlooked. This, combined with the fact that no further iteration was reported despite these high errors, could even indicate a lack of (a meaningful) feedback loop.

This chapter will re-evaluate the way in which this life cycle is currently applied, casting a critical eye on its application in current process prediction methods. For this, we will dive deeper in its application and again use the running example as illustration. Section 4.1 will revisit the business and data understanding phases to get a better understanding of the context of the problem. Section 4.2 will reiterate and link these findings to the current evaluation results and, to conclude, Section 4.3 will provide an overall assessment of what details are fundamentally overlooked in current remaining time prediction approaches.

## 4.1 Business & Data Understanding Revisited

What all approaches have in common, is that they do not devote much of their reporting to the business and data understanding phase. Therefore this section will re-evaluate this process and, based on this, extend it for the running example such that it better captures the context of the problem and the data itself.

### Business Understanding

When considering the business understanding, proposed solutions predominantly encompass new methodologies and their "business" understanding is actually more directed towards research (for example in [7], where there is a certain need for improvement of the current state of the art and the contribution is a method that delivers this), which also explains why it is often defined as such. While taking this research perspective is undoubtedly important, the actual business perspective is often overlooked. Existing remaining time prediction papers motivate their research with a practically relevant application, but do not discuss various important elements of the practical application: business processes. Therefore, the earlier defined research understanding will here be extended with an actual business understanding, which is in this case an understanding of the elements of the process itself. For this, since we cannot actually talk to employees of the organization, we will take the general concept of a *business* process as a point of reference.

In a business process, a case or process execution is usually perceived as the primary element of the process and it is therefore also approached as such. These process executions typically rely on multiple actors and machines in an organization which participate in multiple process executions but have a limited capacity. This means that which actors will be involved in the handling of the process is not dependent on the data of the process alone but is heavily influenced by the availability of actors and system resources, their workload, their choices of ordering all their work from multiple processes, and the collaboration among actors and machines for working on multiple processes.

These *inter-case dynamics* that do emerge in organizational processes are currently not recognized in the business understanding and therefore also not looked for during the data understanding, suggesting that the current results might be misleading.

### Data Understanding

As illustrated in Section 3.3, the emphasis in the data understanding phase for PPM mostly lies in motivating choices for certain data sets such that they permit for reliable and generalizable results of the new techniques proposed. While this is certainly justified, additional data understanding procedures usually remain untreated. Since each data set brings about a different set of characteristics and as these characteristics for the most part make up the context of the problem to be solved, they must be taken into account. The redefined business understanding suggests that additional exploratory data analysis is necessary to discover how these inter-case dynamics play out in the data, such that we can gain insights into how this might influence the overall process and affect the evaluation results.

For this we will use the performance spectrum miner (PSM) [8]. We have imported the RF log in the PSM and specifically searched for patterns that describe behavior of cases together that cannot be accounted for by cases in isolation. The two most significant observations of an inter-case dynamic are the segments shown in Figure 4.1. These observations both satisfy the conditions for the batching on end pattern defined by [1], which emerges when the succeeding processing step of that specific segment applies batch processing.

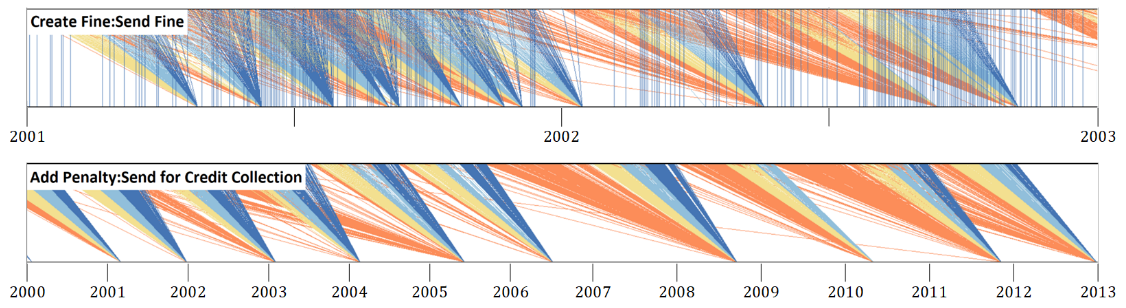

**Figure 4.1:** Exploratory data analysis results: PS visualization of segments *Create Fine:Send Fine* and *Add Penalty:Send for Credit Collection*

The first activity that applies batch processing is the activity *Send Fine*, which shows this batching on end pattern in the segment *Create Fine:Send Fine.* Here, where we see two distinct performance classes: cases that are non-grouped and fast going and cases that are processed in batches at arbitrary moments in time. The second is the activity *Send for Credit Collection*, which shows this batching on end pattern in the segments *Add Penalty:Send for Credit Collection*, *Appeal to Judge:Send for Credit Collection*, *Notify Result Appeal to Offender:Send for Credit Collection*, *Payment:Send for Credit Collection*, *Receive Result Appeal from Prefecture:Send for Credit Collection* and *Send Appeal to Prefecture:Send for Credit Collection.* All these segments show identical behavior and for this reason we have chosen to include only one. This is the segment *Add Penalty:Send for Credit Collection*, where we observe a single performance class: cases that are periodically processed in batches.

To get a more detailed understanding of the significance of these dynamics, we have made use of the recently introduced batch miner [2], which is a tool that for each segment partitions the traces into batches and outputs various statistics thereof. The output showed that from the 69% of the traces that include the activity *Send Fine*, 97% are processed in batches. For the activity *Send for Credit Collection*, which is present in 39% of the traces, we can clearly see that all are processed in batches.

## 4.2 Evaluation Revisited

In the previous we have seen that a significant amount of traces is processed in batches. The fact that so many cases are influenced by this inter-case dynamic might explain why we find such a high error for this event log. To analyze this in more detail, we use an extension of the performance spectrum to overlay the actual outcomes with their corresponding predictions for the segment *Add Penalty:Send for Credit Collection* for the predictions for prefixes of length 4, as shown in Figure 4.2. This visualization will be elaborated on in more detail in Chapter 6, but here we already show its results to support our reasoning.

The black lines in the top half of Figure 4.2 represent the actual progression of cases in the segment *Add Penalty:Send for Credit Collection* and the black lines in the bottom half of Figure 4.2 represent the actual progression of cases in the remainder of the process after *Send for Credit Collection.* For the latter case we only observe a black line going down at the end of a batch, this has to do with the zero time lapse in the remainder of the process since in this case, *Send for Credit Collection* actually is the last activity. The red and blue lines that overlay these black lines represent the predicted progression of cases from the activity *Add Penalty* - their point of prediction - until the end of the process. This visualization will be elaborated on in more detail in Chapter 6.

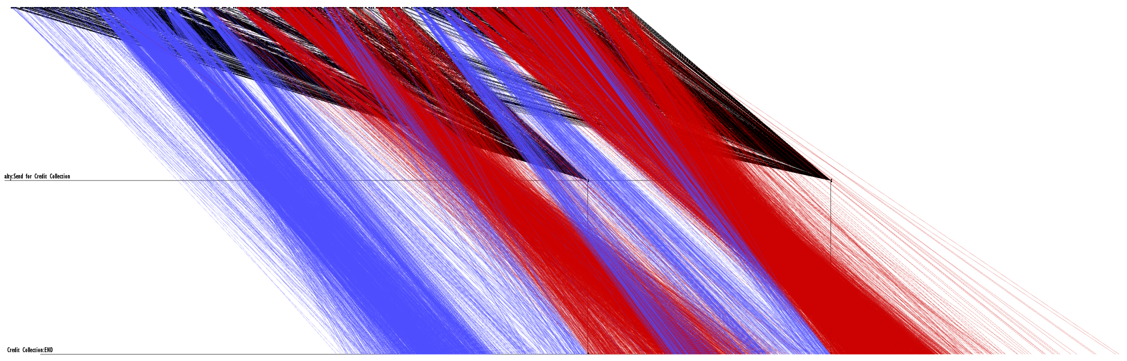

**Figure 4.2:** Exploratory data analysis results: extended PS visualization of actual outcomes (red) overlaid with predicted outcomes (black) for segment *Add Penalty:Send for Credit Collection* for prefix-length = 4

What we can see here is is that the predictions do not at all reflect the batching behavior that is present in the log. We see that some cases are predicted as faster than others (as is also the case with batch processing), but this does not coincide with the actual outcomes, nor do they converge in a similar way. From this we can conclude that the prediction model $RM_{p,a,x}$ has not picked up on the collective behavior that is present in the data set. Additionally, the batch miner also found that the batch processing of this segment occurred at regular intervals, roughly one year apart. This means that the error of the predictions we see in Figure 4.2 can easily be as high as half a year, which is very likely to explain the magnitude of the overall prediction error.

Finally, when we recall this aggregate prediction error that corresponds to the prefix length in Figure 4.2, an MAE of 202.92 days for $k = 4$, all it can tell us is that the prediction model is not very accurate (for this $k$), yet it does not at all reflect the fact that it is caused by these batching mechanisms. The reason is that these measures are an aggregate of intra-case outcomes (outcomes of cases in isolation) and because of that, these measures fall short when we are trying to approach a business process that encompasses inter-case dynamics. This results in rather superficial evaluation results and does therefore not constitute the insights necessary for a meaningful feedback loop.

## 4.3 Shortcomings

By revisiting the business and data understanding, we have uncovered the presence of inter-case dynamics in business processes and their event data. By using this knowledge in the evaluation process, we were able to explain the prediction error on a level other than inference from data preparation or modeling choices, but by *direct observation* of how predictions compare to the actual outcomes - the ground truth.

Figure 4.3 is an abstracted version of the current life cycle of remaining time prediction that was presented in Figure 3.3. We have additionally highlighted the shortcomings that we identified in the previous sections.

First is the lack of attention to the business understanding phase. By not respecting the organizational perspective from the start, fundamental process characteristics are not recognized and are subsequently overlooked in the data understanding phase. This results in an intra-case and therefore incomplete perspective for the remainder of the life cycle, (1) in Figure 4.3. This intra-case perspective leads to a prediction model that does not recognize inter-case dynamics and will as a result also return outcomes that do not reflect inter-case behavior.

Second is that the evaluation method is not tailored towards inter-case behavior, (2) in Figure 4.3. Because the prediction performance is measured for each case in isolation and this result is aggregated, we will never uncover the fact that we overlook the presence of inter-case behavior. Additionally, when we solely assess this aggregated result we will never initiate a feedback loop that addresses this problem.

Third, we have observed that authors do report their reasoning behind how they have come to their final solutions in terms of reiteration, i.e. feedback loop. What can be seen from the results presented in the previous sections is that a lack of attention to this particular part of the life cycle can lead to the overlooking of critical features, causing bad prediction performance. Based on this, one could argue that an actual phase or box is missing in the CRISP-DM model on the anticlockwise arrow depicting this feedback loop in Figure 3.1. The missing or superficiality of this feedback loop is depicted by (3) in Figure 4.3.

Based on this, the goal is to better outline the evaluation and feedback loop for remaining time prediction methods, such that these better capture the inter-case dynamics that are intrinsic to a business process. This will be the foundation for the remainder of this thesis.

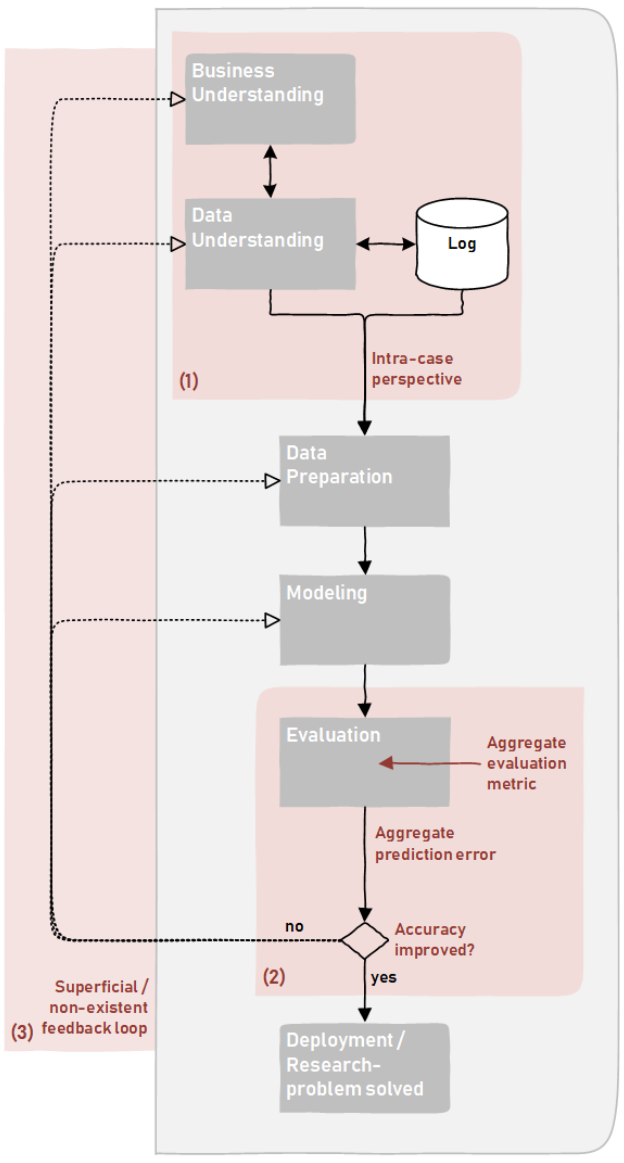

**Figure 4.3:** Shortcomings in the CRISP-DM life cycle for remaining time prediction

# Chapter 5

# Including Inter-Case Features in the Remaining Time Prediction Life Cycle

In Chapter 3, the general development process of contemporary remaining time prediction methods was outlined along the line of the CRISP-DM life cycle and Chapter 4 focused on the phases where these approaches have fallen short. The latter was specifically aimed at the absence of inter-case dynamics in the decision making process along the way.

This chapter will provide a general overview of the details we propose to extend various steps of this life cycle with. These additional details are all aimed at refining the feedback loop and thereby incorporating the notion of inter-case dynamics in the decision making process, such that shortcomings originating from these dynamics can be diminished in the future. Figure 5.1 illustrates a global overview of these refinements along the line of the life cycle for remaining time prediction as was presented in Chapter 3. In this figure, we distinguish the current - intra-case - flow (in grey) from the new - inter-case - flow (in blue).

The first step towards a remedy for this problem is the refining of the feedback loop. We want to take the inter-case perspective we have obtained by revisiting the business and data understanding phases and contrast this with the evaluation output. By linking the business/data understanding to the evaluation, we induce a meaningful feedback loop that is currently missing at (3) in Figure 4.3. By re-evaluating the outcomes from an inter-case perspective, we can diagnose what inter-case dynamic might be the cause for a certain prediction error. To achieve this, we want to compare the actual outcomes to the predicted outcomes for all cases together at an individual level without any form of aggregation, using techniques that can actually reveal these inter-case dynamics. During this analysis, we want to find a problematic subset of predictions that ignore some form of collective behavior that can be seen in the actual outcomes. Once such a subset is identified, we need to determine the actual mechanics or pattern that is the source of this behavior. As discussed, these steps require the knowledge obtained from revisiting the business and data understanding phase, as well as the output of the evaluation phase. Therefore, in Figure 5.1, we have merged these steps into the aggregate step **fine-grained error diagnosis** and positioned it accordingly. This will be the core of Chapter 6.

Once we have found what inter-case dynamic is the source of the problem, we not only want to incorporate it in the prediction model, but we also want to be able to better measure the performance with respect to this uncovered dynamic. Therefore we define two steps that can be followed in parallel. The first encompasses a methodology for inter-case feature creation. Incorporating such a feature in the prediction model will help in better predicting the behavior of a case and therefore also its outcome. While the overall method we present is general, we concretely show how to do this for the inter-case dynamics that are caused by batching. The output of this step will be used in the data preparation phase and is in Figure 5.1 positioned as such. This **derivation of inter-case features for batching** will be the core of Chapter 7.

The second parallel step encompasses the extending of the current evaluation process. When we have uncovered what kind of inter-case behavior is present, we can match the evaluation method and/or metrics such that we not only can asses the prediction performance in terms of intra-case outcomes, but also in terms of inter-case outcomes. Because we concretely show how to create features for batching, this evaluation method will also be aimed at inter-case dynamics caused by batching. The output of this step will be used in the evaluation phase and therefore, in Figure 5.1, this **derivation of an inter-case evaluation for batching** is also positioned accordingly. This step will be the core of Chapter 8.

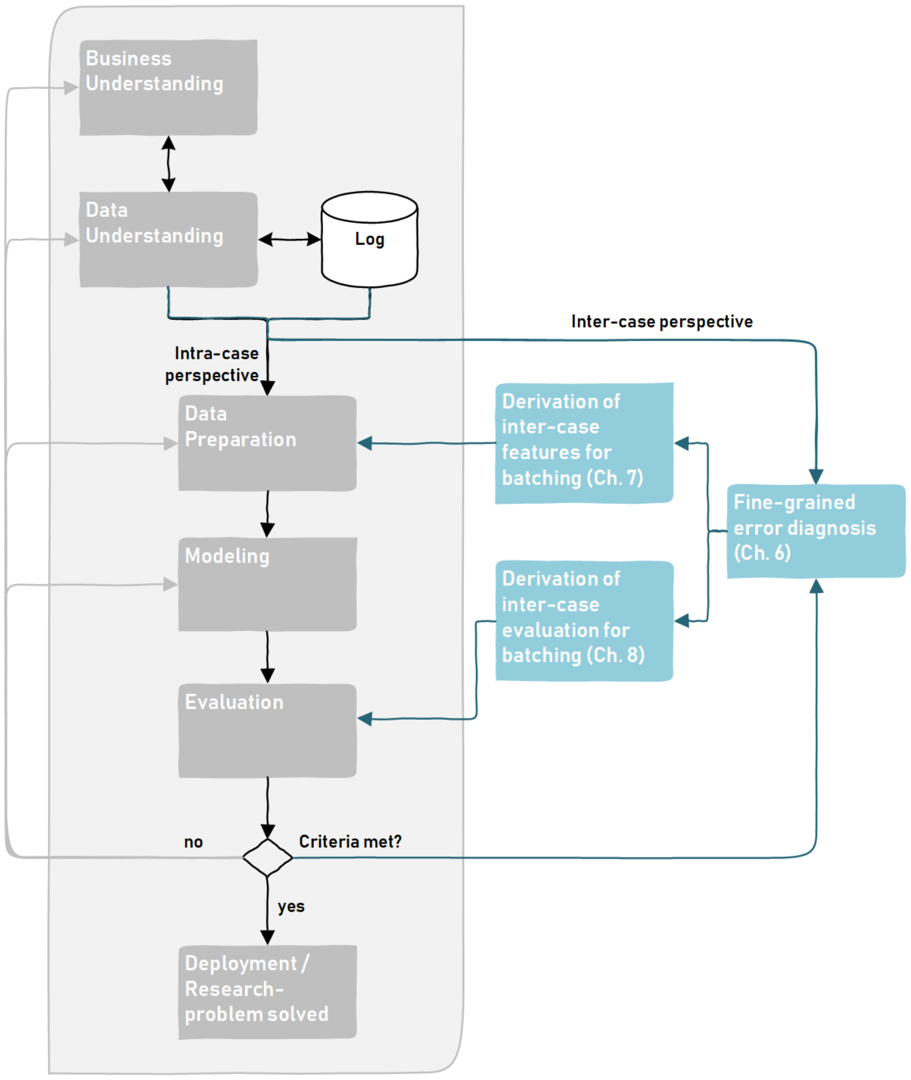
Business Understanding
Data Understanding
Log
Inter-case perspective
Intra-case perspective
Data Preparation
Derivation of inter-case features for batching (Ch. 7)
Fine-grained error diagnosis (Ch. 6)
Modeling
Derivation of inter-case evaluation for batching (Ch. 8)
Evaluation
no
Criteria met?
yes
Deployment / Research-problem solved

**Figure 5.1:** Extension of CRISP-DM life cycle for remaining time prediction

Running Example

In the following chapters, we will again use method $(p, a, x)$ and the RF log from our running example for illustration of the proposed steps in this chapter. For this we will give a brief recap of how the model $RM_{p,a,x}$ is built and what parts of our training and testing data of the RF log we need for the steps in the following chapters.

As explained earlier, for the training and testing of $RM_{p,a,x}$, the RF log was partitioned into a training and a test set using a temporal split of the log as was also done in [4]. For this the cases are ordered according to start time and the first 80% are used for training and the last 20% are used for testing, resulting in $L_{train}$ and $L_{test}$. We train model $RM_{p,a,x}$ according to method $(p, a, x)$ [4] using prefixes $hd_\sigma^k \in L_{train}$ to predict label $y$, i.e. the remaining time. Then, we use model $RM_{p,a,x}$ to predict for each prefix $hd_\sigma^k \in L_{test}$ the remaining time $\overline{y}$.

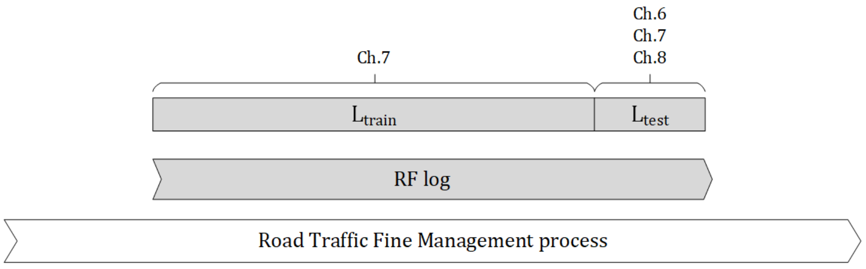

**Figure 5.2:** Train-test split of Road Traffic Fine Management log for the running example

In Figure 5.2 we have depicted the train/test split indicating which subsets are required for which steps/chapters. Chapter 6 encompasses fine-grained evaluation techniques for which we naturally solely use our testing data. Specifically, we will use the complete traces in $L_{test}$ and their corresponding outcomes $\overline{y}$. Chapter 7 encompasses deeper understanding of the process and therefore for this chapter we make use of both $L_{train}$ and $L_{test}$, but we do not care for outcomes $\overline{y}$. Finally, Chapter 8 again encompasses evaluation methods and we will there also solely use $L_{test}$ and their corresponding outcomes $\overline{y}$.

# Chapter 6

# Fine-Grained Error Diagnosis

This chapter presents the first steps towards the inclusion of inter-case dynamics in the remaining time prediction life cycle. The objective is to analyze the output of the evaluation phase from an inter-case perspective such that we can uncover possible inter-case dynamics that cause high prediction errors. For this, we will analyze the prediction output by contrasting the predicted outcomes with the actual outcomes on a fine-grained level. Since all cases are unique and all prefixes of these cases are unique, every prediction that we make, be it for training or testing, is also unique. To be able to analyze the errors for these predictions on an individual level, we first establish some terminology in Section 6.1. The actual methodologies that serve this fine-grained analysis are presented in Sections 6.2 and 6.3.

Section 6.4 elaborates on the actual process of identifying subsets of high error cases that do not conform to inter-case patterns we *also* can observe in the data. When we have identified such a pattern and thereby the context for a high prediction error, we can use this knowledge to improve the prediction model itself and extend its evaluation, which will be the topics of Chapters 7 and 8, respectively.

## 6.1 Describing Individual Predictions

In this section we establish some terminology to describe individual predictions. We define an individual prediction as the *reasoning step* that is performed about a specific prefix of a trace $\sigma = \langle e_1, ...e_n \rangle$ to predict an outcome as close as possible to the actual outcome. To clearly define all related concepts, we define the following terms:

- **Prefix** $hd^k(\sigma) = \langle e_1, ..., e_k \rangle$: the partial trace that carries the information used as input for the predictive reasoning.
- **Prediction** $P_{hd^k(\sigma)}$: the predictive reasoning step, defined by its input, i.e. prefix.
- **Predicted outcome** $\overline{y}_P$: the predicted outcome for the remaining time made by the predictive reasoning step $P_{hd^k(\sigma)}$.
- **Suffix** $tl^{n-k}(\sigma)$: the partial trace that carries the information yet unknown at the predictive reasoning step $P_{hd^k(\sigma)}$.
- **Actual outcome** $y_P$: the actual outcome for the remaining time, derived from the suffix

of $P_{hd^k(\sigma)}$, available only after the trace finished, i.e. for training or testing, but not at the moment of the predictive reasoning step itself.

- **Moment of prediction** $t_P$: the moment in the trace at which the predictive reasoning step took place, i.e. the time at which the last event of the prefix that defines $P_{hd^k(\sigma)}$ is observed.
- **Point of prediction** (intra-case) $e_k$: the point in the trace at which the predictive reasoning step took place. From an intra-case perspective, i.e. the perspective of a case, this is the last observed event in the prefix that defines $P_{hd^k(\sigma)}$.
- **Point of prediction** (inter-case) $a$: the point in the process at which the predictive reasoning step took place. From an inter-case perspective, i.e. the perspective of the process, this is a shared point of prediction across traces, i.e. a set of prefixes where their $k^{\text{th}}$ events have something in common, e.g. have the same activity, i.e. a common prediction point. We will use this later on to align multiple predictions over time based on their moment of prediction, such that we can lift intra-case perspectives to an inter-case perspective.

In the following, we just write $P$ for a predictive reasoning step $P_{hd^k(\sigma)}$ if the specific prefix is clear or not relevant, e.g. when discussing a set of predictions. Figure 6.1 illustrates these concepts for the case of remaining time prediction.

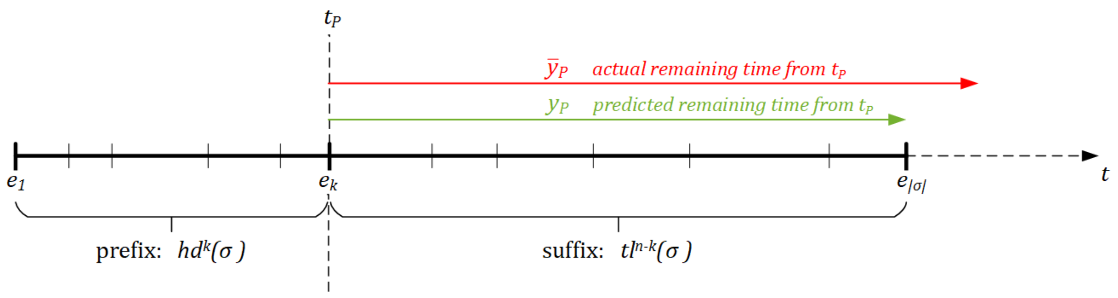
$t_P$
$\bar{y}_P$ actual remaining time from $t_P$
$y_P$ predicted remaining time from $t_P$
$e_1$
$e_k$
$e_{|\sigma|}$
$t$
prefix: $hd^k(\sigma)$
suffix: $tl^{n-k}(\sigma)$

**Figure 6.1:** Illustration of a prediction $P$ for the case of remaining time prediction

To illustrate the concepts and definitions of this chapter, we introduce a technical example (not related to the running example). This example comprises a small event log $L_1$ along with a corresponding set of predictions $\mathcal{P}$, depicted in Table 6.1, left and right, respectively, to show how the predictions relate to the events in the log. We firstly see that $t_P$ always corresponds to the timestamp of the last event in the corresponding prefix. We also see that $y_P$ is simply calculated by subtracting $t_P$ from the final timestamp of the corresponding trace. The last column, $\overline{y}_P$, is the result of a reasoning step, which in this case was chosen to illustrate typical errors in prediction models.

**Table 6.1:** Example event log $L_1$ (left) aligned with predictions $\mathcal{P}$ (right)

| Case ID | Activity | Timestamp | Prefix | $k$ | $e_k$ | $t_P$ | $y_P$ | $\overline{y}_P$ |
|---|---|---|---|---|---|---|---|---|
| 1 | A | 1 | $\langle A\rangle$ | 1 | A | 1 | 6 | 10 |
| 1 | B | 2 | $\langle A,B\rangle$ | 2 | B | 2 | 5 | 2 |
| 1 | C | 5 | $\langle A,B,C\rangle$ | 3 | C | 5 | 2 | 3 |
| 1 | D | 5 | $\langle A,B,C,D\rangle$ | 4 | D | 5 | 2 | 2 |
| 1 | E | 7 | $\langle A,B,C,D,E\rangle$ | 5 | E | 7 | 0 | 1 |
| 2 | A | 3 | $\langle A\rangle$ | 1 | A | 3 | 11 | 14 |
| 2 | B | 5 | $\langle A,B\rangle$ | 2 | B | 5 | 9 | 11 |
| 2 | C | 8 | $\langle A,B,C\rangle$ | 3 | C | 8 | 6 | 7 |
| 2 | E | 14 | $\langle A,B,C,E\rangle$ | 4 | E | 14 | 0 | 2 |
| 3 | A | 5 | $\langle A\rangle$ | 1 | A | 5 | 12 | 11 |
| 3 | E | 7 | $\langle A,E\rangle$ | 2 | E | 7 | 10 | 8 |
| 3 | B | 14 | $\langle A,E,B\rangle$ | 3 | B | 14 | 3 | 4 |
| 3 | C | 17 | $\langle A,E,B,C\rangle$ | 4 | C | 17 | 0 | 0 |

## 6.2 Performance Spectrum with Error Progression

The first method for a fine-grained error-analysis that we present is the *performance spectrum with error progression* (PSw/EP). The goal of this method is to understand where along an entire case prediction errors arise. We want to achieve this by analyzing the progression of the prediction error over the course of a trace. Specifically, we want to use the original structure of the PS (see Chapter 2) and add special color coding for each segment observation to indicate a certain type of error progression. This method does not require any input parameters and solely uses the set of predictions $\mathcal{P}$ as input data.

Let us again consider a prediction $P$. Each prediction $P$ was made in a particular trace $\sigma$, at a particular prefix $hd^k(\sigma)$. This means that for a trace $\sigma$ of length $|\sigma|$, we can extract $|\sigma|$ prefixes and as a result we also have $|\sigma|$ predictions $P_1...P_{|\sigma|}$. To analyze the progression of the prediction error over the course of a trace, we want to compare each pair of subsequent predictions, resulting in $|\sigma|-1$ comparisons. Since we compare predictions between subsequent processing steps and for each trace we have $|\sigma|-1$ segments, we can link each comparison to an *occurrence* of a process segment.

Each comparison consists of a pair of remaining time predictions $P_{hd^k(\sigma)}$ and $P_{hd^{k+1}(\sigma)}$ made at time $t_k$ and $t_{k+1}$, respectively, corresponding to the points of prediction $e_k$ and $e_{k+1}$, respectively. These subsequent points of prediction $e_k$ and $e_{k+1}$ can be translated into the process segment $(e_k, e_{k+1})$. For the occurrence of segment $(e_k, e_{k+1})$ for each $\sigma$, we want to evaluate whether the error increased or decreased (or remained equal) after its completion. To do this, we first compute the errors at points $e_k$ and $e_{k+1}$. This results in an error $\epsilon_k = \overline{y}_{P_{hd^k(\sigma)}} - y_{P_{hd^k(\sigma)}}$ at point $e_k$ and an error $\epsilon_{k+1} = \overline{y}_{P_{hd^{k+1}(\sigma)}} - y_{P_{hd^{k+1}(\sigma)}}$ at point $e_{k+1}$. Because the purpose of this method is to solely compare the magnitude of these errors, we subsequently take their absolute values $|\epsilon_k|$ and $|\epsilon_{k+1}|$. Finally, because it is very likely that the error will decrease as the prefix length increases, i.e. more is known about a trace, we will take the absolute errors relative to the actual outcomes: $rae_k = |\epsilon_k|/y_{P_{hd^k(\sigma)}}$ and $rae_{k+1} = |\epsilon_{k+1}|/y_{P_{hd^{k+1}(\sigma)}}$. As a result, for the occurrence of segment $(e_k, e_{k+1})$ we have $rae_k$ and $rae_{k+1}$, corresponding to step $e_k$ and $e_{k+1}$, respectively.

Now we can derive the error progression for this segment occurrence, which we will do by evaluating whether the error went up or down as a result of the segment. We classify each segment occurrence $(e_k, e_{k+1})$ with a decreasing error if $rae_k > rae_{k+1}$, with an increasing error if $rae_k < rae_{k+1}$ and, in the exceptional case, with an unchanged error if $rae_k = rae_{k+1}$.

We do this for all prediction pairs of all cases and as a results we end up with a classification that for all segment occurrences of all cases separately indicates whether the error increased or decreased as a result of it.

The next step is to use this classification to visualize the error progression over time. For this we want to make use of the PS [1], which allows us to classify each segment occurrence with a color, by coloring each segment occurrence based on our own error progression classification. A segment occurrence that results in an error decrease will be colored red and a segment occurrence that results in an error increase will be colored blue.

To illustrate these steps, we use the predictions from Table 6.1. For each prediction we first calculate the *rae*, depicted in Table 6.2 (left), then we check for each subsequent pair whether the *rae* increased or decreased (or remained equal), depicted in Table 6.2 (right). By checking the prediction points of the corresponding predictions, we can link this progression to its corresponding segment occurrence, also depicted in Table 6.2 (right).

**Table 6.2:** Example predictions $\mathcal{P}$ with *rae* calculation (left) and *rae* progression related to corresponding segments (right)

| Case ID | Prefix | $e_k$ | $y_P$ | $\overline{y}_P$ | $rae$ | Segment | Progression $rae$ |
|---|---|---|---|---|---|---|---|
| 1 | $\langle A\rangle$ | A | 6 | 10 | 0.667 | | |
| 1 | $\langle A,B\rangle$ | B | 5 | 2 | 0.600 | (A,B) | $\Downarrow$ |
| 1 | $\langle A,B,C\rangle$ | C | 2 | 3 | 0.500 | (B,C) | $\Downarrow$ |
| 1 | $\langle A,B,C,D\rangle$ | D | 2 | 2 | 0 | (C,D) | $\Downarrow$ |
| 1 | $\langle A,B,C,D,E\rangle$ | E | 0 | 1 | $\infty$ | (D,E) | $\Uparrow$ |
| 2 | $\langle A\rangle$ | E | 11 | 14 | 0.272 | | |
| 2 | $\langle A,B\rangle$ | B | 9 | 11 | 0.222 | (A,B) | $\Downarrow$ |
| 2 | $\langle A,B,C\rangle$ | C | 6 | 7 | 0.167 | (B,C) | $\Downarrow$ |
| 2 | $\langle A,B,C,E\rangle$ | E | 0 | 2 | $\infty$ | (C,E) | $\Uparrow$ |
| 3 | $\langle A\rangle$ | A | 12 | 11 | 0.083 | | |
| 3 | $\langle A,E\rangle$ | E | 10 | 8 | 0.200 | (A,E) | $\Uparrow$ |
| 3 | $\langle A,E,B\rangle$ | B | 3 | 4 | 0.333 | (E,B) | $\Uparrow$ |
| 3 | $\langle A,E,B,C\rangle$ | C | 0 | 0 | 0 | (B,C) | $\Downarrow$ |

Figure 6.2 illustrates the PS visualization of the error progression over time, corresponding to the event log from Table 6.1 and the classification from Table 6.2. Additionally, Figure 6.8, which will be discussed later on, shows an actual PS visualization with error progression. As with the standard PS, segments can be ordered along a trace variant and as such, the error progression can be followed along the trace variant.

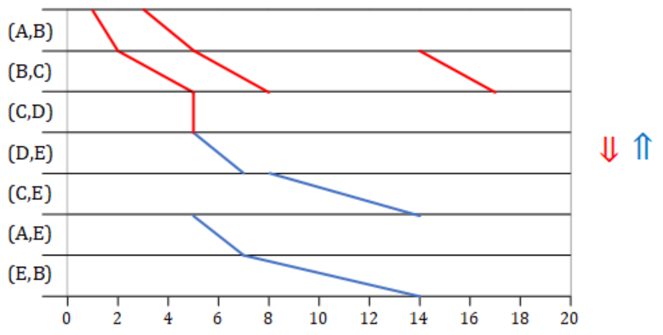

**Figure 6.2:** PSw/EP for $\mathcal{P}$

In a PSw/EP diagram, the lines in x,y show the actual progression of cases over time on the level of process segments. The color of each of these segment occurrences indicates the progression of the error of the prediction model along the line of a case. From this we can see whether there is a correlation between particular features in x,y (cases over time) and errors by seeing multiple occurrences that follow similar patterns in x,y showing similar error colors.

In terms of analysis, we encounter two types of error progression (we do not consider the exceptional case where the error remains equal). An error decrease after a certain segment occurrence $(a, b)$ means that the observation of $b$ by the prediction model decreased the error, which could suggest that activity $b$ carries some uncertainty which is apparently resolved after its completion. An error increase after a certain segment occurrence $(a, b)$ means that the observation of $b$ by the prediction model increased the error, which could suggest that the learning about activity $b$ increases uncertainty about the remainder of the trace. Of these two progression types, only an error decrease can tell us something about uncertainty within the segment $(a, b)$ itself. Therefore, to discover patterns that could be critical in explaining prediction errors, we will exclusively focus on segment occurrences classified with an error *decrease.*

Finally, a prediction model usually predicts up until a specific prefix length $k_{max}$. This also limits our analysis to prefixes up until $k_{max}$, resulting in segments that do not have corresponding predictions and therefore no error progression classification. These segments will have a grey color to indicate they must be excluded from the analysis.

## 6.3 Overlaid Performance Spectrum

Section 6.2 introduced how we can visualize the progression of the error along an entire case, but here the goal is to understand how the behavior of the predicted outcomes compares to that of the actual outcomes. For this we present the *overlaid performance spectrum* (OPS). The aim of this method is to enable an unbiased, fine-grained analysis of predictions by visualizing their predicted- and actual behavior in the remainder of the process, i.e. their suffix. As we have previously seen, the PS can show such behavior of all cases over time. Therefore, we want to visualize these predicted and actual outcomes in the context of the PS as well.

First, since we aim for a fine-grained analysis of the predictions, we want to visualize only a single process segment $(a, b)$ at a time. Specifically, because we want a shared point of reference among the predictions such that we can compare them, we want to only visualize those predictions that traverse segment $(a, b)$ in their next step. Second, since our objective is to also minimize our bias, we only want to visualize predictions of equal prefix-length $k$ at a time. This is because the more events are contained in a prefix, i.e. the larger $k$, the more information we have about the corresponding case and the more we can tell about its future, i.e. the more accurate predictions we can make, which would result in an unfair comparison if different prefix-lengths $k$ are involved.

We first bucket the predictions based on prefix-length, e.g. predictions $\mathcal{P}_k$ are predictions with prefixes $\langle e_1, ..., e_k \rangle$ of the same length $k$. As cases evolve in different ways, i.e. each bucket $\mathcal{P}_k$ contains predictions with different prediction points $e_k$, we "sub-bucket" the predictions based on the prediction point $e_k$. For the possible prediction points $a_1...a_m$, we consider the sub-buckets $\mathcal{P}_{k,a_1} \subseteq \mathcal{P}_k ... \mathcal{P}_{k,a_m} \subseteq \mathcal{P}_k$. Each $\mathcal{P}_{k,a_i}$ contains all the predictions of $\mathcal{P}_k$ that have $e_k = a_i$. At this instant, each $\mathcal{P}_{k,a_i}$ contains predictions with a different next activity $e_{k+1}$. Therefore we again sub-bucket the predictions, this time based on next activity $e_{k+1}$. For the possible next activities $b_1...b_m$, we consider the sub-buckets $\mathcal{P}_{k,a_i,b_1} \subseteq \mathcal{P}_{k,a_i} ... \mathcal{P}_{k,a_i,b_m} \subseteq \mathcal{P}_{k,a_i}$. Each $\mathcal{P}_{k,a_i,b_j}$ contains all the predictions of $\mathcal{P}_{k,a_i}$ where the next segment will be $(a_i, b_j)$ and that have prefix-length $k$.

For each prediction in $\mathcal{P}_{k,a_i,b_j}$ we now have a shared point of reference, namely the start time of the corresponding trace within segment $(a_i, b_j)$ will be equal to the moment of prediction:

$t_{a_i} = t_P = t_{e_k}$. We also have the actual timestamp of the next step $t_{b_j} = t_{e_{k+1}}$, actual moment the process ends $t_k + y_P = t_{e_{|\sigma|}}$ and the moment the process is predicted to end $\overline{t}_{e_{|\sigma|}} = t_P + \overline{y}_P$. We recall that in the PS each case through a segment $(a, b)$ is a line from $(t_a, a)$ to $(t_b, b)$. We want use this to map the timestamps relating to a prediction $P$ onto process segments. We will write $(b, e_{|\sigma|})$ for the segment from $b$ until the actual end of the case and $(b, \overline{e}_{|\sigma|})$ for the segment from $b$ until the predicted end of the case. Figure 6.3 illustrates such a mapping from a prediction $P$ to segments $(a, b)$, $(b, e_{|\sigma|})$ and $(b, \overline{e}_{|\sigma|})$. Using this mapping, we can plot the predicted outcomes $\overline{y}_P$ against the actual outcomes $y_P$ in x,y for each prediction in $\mathcal{P}_{k,a_i,b_j}$.

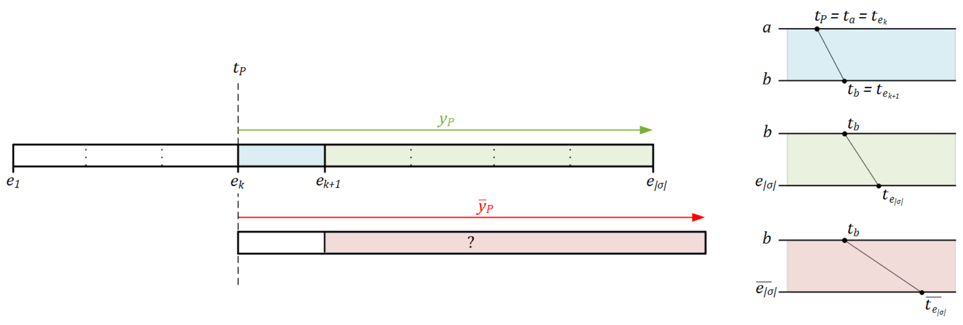

**Figure 6.3:** A prediction $P$ annotated with segments

Figure 6.3 shows how we want to visualize the values $t_{e_{k+1}}$, $y_P$ and $\overline{y}_P$ w.r.t. $t_P$ in the PS. For properly analyzing the behavior of $\overline{y}_P$ w.r.t. $y_P$, we want to overlay the segment that visualizes the actual value $y_P$ with the segment $(b, e_{|\sigma|})$ $(b, \overline{e}_{|\sigma|})$ that visualizes the predicted value $\overline{y}_P$, such that they can be compared to each other directly. For this we introduce *layers*.

### First Layer

The OPS takes the segment $(a, b)$ as a starting point, which is equal to $(e_k, e_{k+1})$ for all predictions $P \in \mathcal{P}_{k,a,b}$. This segment S1 describes the *actual* behavior of the predictions from step $e_k$ to step $e_{k+1}$, illustrated in blue in Figure 6.5. Because the chances are high that $e_{k+1}$ is not the last activity in the overall process and since we aim to predict when the process ends $t_{e_k} + y_P$ and not when the second activity of a process segment occurs $t_{e_{k+1}}$, we need to extend S1 so that we can describe the remainder of the behavior after $e_{k+1}$. We will do this by adding a segment S2 below S1. S2 is illustrated in green in Figure 6.5 and describes the *actual* behavior of the predictions from step $e_{k+1}$ to the final step $e_{|\sigma|}$. S1 and S2 together form layer L1 in the OPS.

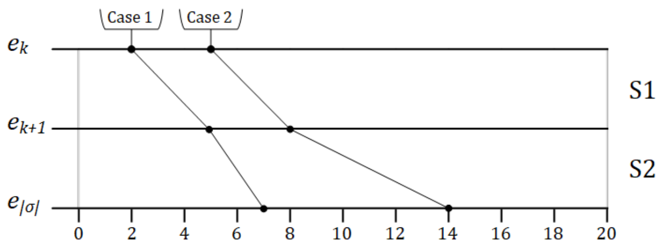

**Figure 6.4:** Example of the first layer L1 of overlaid performance spectrum

Layer 1, containing S1 and S2, visualizes the *actual* behavior from $t_P$ until the end of the process. Figure 6.4 shows an example of L1, in which the black lines show the actual progressions in x,y of two predictions.

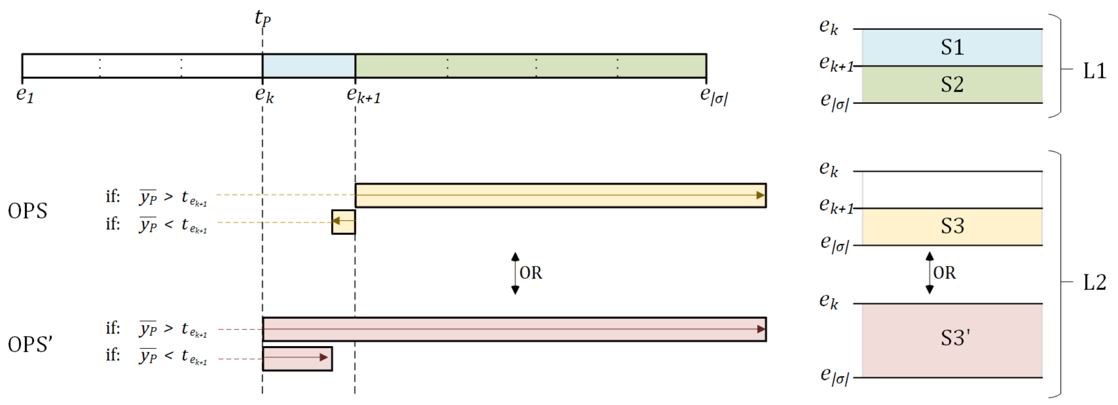
$t_P$
$e_1$
$e_k$
$e_{k+1}$
$e_{|\sigma|}$
OPS
if: $\overline{y}_P > t_{e_{k+1}}$
if: $\overline{y}_P < t_{e_{k+1}}$
OR
OPS'
if: $\overline{y}_P > t_{e_{k+1}}$
if: $\overline{y}_P < t_{e_{k+1}}$
S1
S2
L1
S3
S3'
L2

**Figure 6.5:** Translation from predictions to segments: composition of the OPS

**Second Layer**

The layer L2 of the OPS will visualize the *predicted* behavior of our predictions. We want to analyze the behavior by visualizing not only the predicted outcomes but also the errors following S1, i.e. the errors after the next step $(e_k, e_{k+1})$. We propose two variants for visualizing the predictions: OPS from $e_k + 1$ until $e_{|\sigma|}$ and OPS' from $e_k$ until $e_{|\sigma|}$. For each of these cases, we select one segment to put on top of L1. Figure 6.6 shows an overlay of L1 (from Figure 6.4) with these alternatives for L2, which we will explain in detail in the following.

**OPS.** For this variant, we visualize the predicted outcome $\overline{y}_P$ for the segment $(e_{k+1}, e_{|\sigma|})$ as a line from $t_{e_{k+1}}$ to $t_{e_{|\sigma|}}$, meaning we take the actual execution of $e_{k+1}$ into account. This segment S3 describes the *predicted* behavior of a prediction $P$ from step $e_{k+1}$ to step $e_{|\sigma|}$. This way we can analyze the deviations between the actual and predicted outcomes that result from the segment $(e_k, e_{k+1})$ under consideration. This segment shares both the first and last step of S2 and will therefore be laid over this segment, as illustrated in yellow in Figure 6.5.

**OPS'.** For this variant, we visualize the predicted outcome $\overline{y}_P$ for the segment $(e_k, e_{|\sigma|})$ as a line from $t_{e_k}$ to $t_{e_{|\sigma|}}$. This segment S3' describes the *predicted* behavior of a prediction $P$ from step $e_k$ to step $e_{|\sigma|}$. Because this segment shares the first step with S1 and the last step with S2, this segment will be laid over both of these segments of L1, as illustrated red in Figure 6.5.

To increase understandability, we also introduce a way to color code the predicted segment occurrences in S3 and S3'. When the outcome is overpredicted, i.e. it exceeds the actual outcome, the predicted segment occurrence will be coded in red. Alternatively, when the outcome is underpredicted, i.e. it predicts a value smaller than the actual outcome, the predicted segment occurrence will be coded in blue.

Figure 6.6, which we have used to illustrate the different layers in the previous, is actually the result of this method applied to the technical example from Table 6.1, using segment $(B, C)$ and prefix-length $k$. The set of predictions $\mathcal{P}_{2,B,C} \subseteq \mathcal{P}$ only include case 1 and 2, because for case 3 the segment $(B, C)$ does not occur at $k = 2$. Case 1 is underpredicted and shown in blue and case 2 is overpredicted and shown in red. Additionally, Figures 6.9 to 6.12, which will be discussed later on, show actual OPS and OPS' visualizations for two different process segments of the RF log.

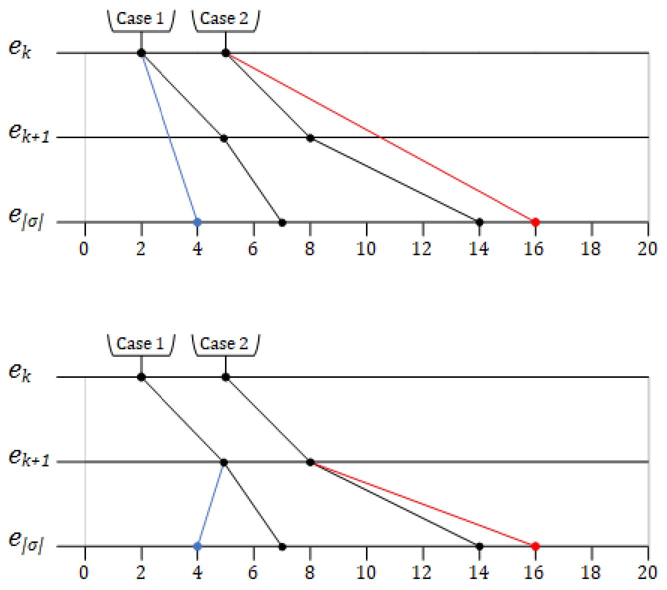

**Figure 6.6:** OPS (top) and OPS' (bottom) of $\mathcal{P}$ for segment $(B, C)$ and $k = 2$ (of our running example from Table 6.1)

## 6.4 Subset & Subset Pattern Identification

Once we have the methods in place, we can analyze the predictions to possibly uncover behavior that is not captured by the prediction method. We aim to do this by uncovering subsets of predictions of which the collective behavior of predicted segment occurrences does not match the collective behavior of the actual segment occurrences.

### 6.4.1 Select Segments

For this process we first need to identify which segments we want to analyze. When we consider the entire set of predictions that relate to an actual event log, we have a significant amount of segments to consider. For example, the RF log counts 70 unique process segments. To reduce this search space, we only consider segments that occur significantly often in the log. We believe that identifying errors in segments that occur often in the log will likely be more valuable in yielding improved prediction results than identifying errors in segments that occur less often.

### 6.4.2 Visualize & Inspect

The next step is to visualize the OPS and PSw/EP for these selected segments. Because we build upon the visual analytics technique from [1], which allows to visually detect patterns in the performance progression of a process, we can inspect the OPS and PSw/EP for patterns of interest.

### 6.4.3 Diagnose

In the end our goal is to analyze visualizations such as Figures 6.9 to 6.12 and diagnose the source of a prediction error by finding patterns in them. To help in systematically identifying these patterns, we refer to the taxonomy defined in [1]. This taxonomy defines various patterns that can visually be deducted from the PS and each of these patterns can be matched to an occurrence of process behavior, which in most cases is collective, i.e. a result of cases together. A selection of the patterns that relate to the order of the lines and their occurrences is depicted in Figure 6.7. We will use this taxonomy as a basis for identifying discrepancies in groups of predicted and actual segment occurrences.

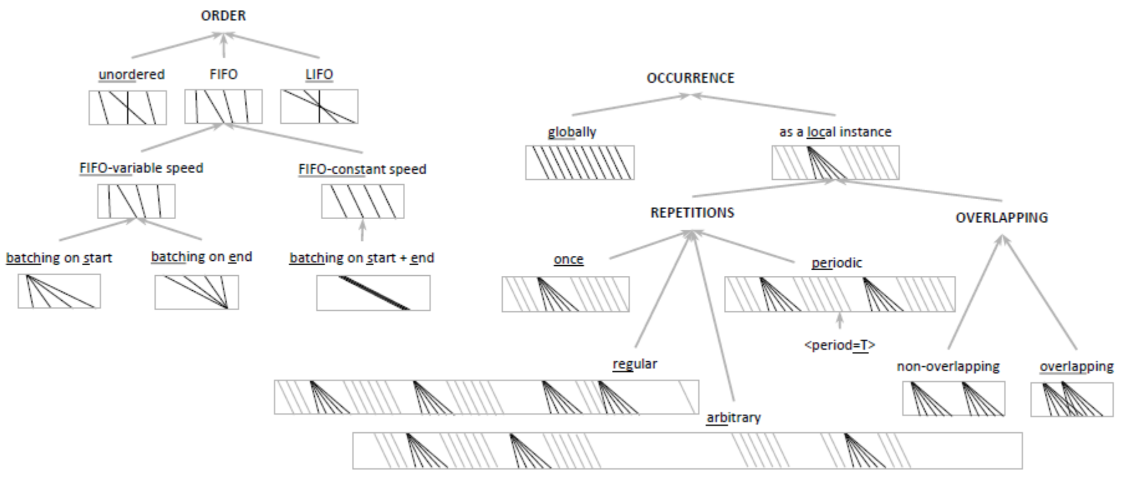

**Figure 6.7:** Taxonomy of order and occurrence patterns (from [1])

The following two paragraphs shortly describe how we generally wish to detect the uncaptured inter-case behavior depending on the error analysis methods we have presented in Sections 6.2 and 6.3. From this analysis, we retrieve a pattern $R$ for segment $S$ as a relevant pattern, i.e. $R$ is a result of inter-case behavior captured in the actual progressions and not captured in the predicted progressions of cases. This pattern $R$ for segment $S$ we will use as input for the derivation of inter-case features and inter-case evaluation in Chapters 7 and 8, respectively.

**PSw/EP** When considering the PSw/EP, we want to find critical segments, i.e. segments of which its completion results in an error decrease, because exactly for these segments we in the end also want to also decrease the error at the start. To identify an inter-case behavior worth investigating, we need to check for these critical segments whether they show an inter-case pattern. If this is the case, such an inter-case pattern is most likely not picked up and might be the cause of the high prediction error at the moment preceding such a segment.

**OPS** When considering the OPS, we can actually compare the predicted and actual segment occurrences since these are both visualized. For this we follow the following procedure:

1. We identify the most significant segments and for each available prefix-length $k$ we can create an OPS and an OPS'.
2. For each of these structures, we aim to find patterns in the actual segment occurrences that are a result of inter-case dynamics, for example described by one of the patterns in Figure 6.7.
3. If we observe such a pattern, we evaluate whether the predicted segment occurrences also follow this pattern. We do this by listing the characteristics of the observed pattern in the

group of actual segment occurrences and contrasting these with the characteristics of the group of predicted segment occurrences. If these do not match, the prediction method has apparently not picked up on the inter-case behavior that caused the pattern and incorporating it in the prediction might reduce the error of the prediction model.

While this process can be carried out quite systematically, it is important to keep in mind that we are still trying to deduct these subsets and its patterns visually and are thus susceptible to the user's interpretation.

Running Example

Here we will apply the two methods presented in this chapter to our running example and combine the outcome of these two analysis steps. If we then find the same pattern in each step, i.e. using the OPS/OPS' and the PSw/EP, the pattern is most likely relevant.

As discussed in Chapter 5, we have run model $RM_{p,a,x}$ from [4] for each prefix $hd^k(\sigma) \in L_{test}$ and extracted all information related to each individual prediction (see Section 6.1) such that we now have a set of predictions $\mathcal{P}_{test}$. This we can then use as input for the visualization methods presented in this chapter.

**Select segments**

Our first step is to select the segments we want to consider, which relates to the step discussed in Subsection 6.4.1. We have loaded the RF log in the PSM and selected the segments that had the most observations. For the PSw/EP, this is sufficient, but for the OPS, we also need to consider for each of these segments for which of the prefix lengths they apply. For example for segment *Create Fine:Send Fine* we only have to consider $k = 1$, since it does not occur later in the prefix for any of the traces. As was also done in the original experiment, we only consider prefixes up to length 5. The resulting segments and corresponding prefix lengths are depicted in Table 6.3. Some of these combinations only had a very small amount of predictions, these are depicted in brackets. The same goes for the segment *Insert Fine Notification:Add Penalty*, since this segment exclusively shows global FIFO behavior and will not add anything meaningful to this analysis.

**Table 6.3:** Selected segments and corresponding prefix-lengths of the RF log for fine-grained error analysis

| Segment | $k=1$ | $k=2$ | $k=3$ | $k=4$ | $k=5$ |
|---|---|---|---|---|---|
| Create Fine:Send Fine | × | | | | |
| Send Fine:Insert Fine Notif. | | × | (×) | | |
| Insert Fine Notif.:Add Penalty | | | (×) | (×) | |
| Insert Fine Notif.:Insert Date Appeal to Pref. | | | × | (×) | |
| Insert Date Appeal to Pref.:Add Penalty | | | | × | (×) |
| Add Penalty:Send for Credit Collection | | | | × | (×) |
| Payment:Send for Credit Collection | | | | | × |

**Visualize and inspect PSw/EP**

We visualize the PSw/EP for the all the segments from Table 6.3 using the method we introduced in Section 6.2. The result is depicted in Figure 6.8. For the observations depicted in grey no classification was available. This is because the classification was solely made for the testing data while the observations at the start of the third to fifth segment are of cases that started before the point of the temporal train/test split and are therefore still part of the training data. As for the last segment in Figure 6.8, the corresponding activities occur at $k = 5$ and $k = 6$. Since mode $RM_{p,a,x}$ only predicts for $k \leq 5$, we have no prediction output for $k = 6$ which means no error progression can be calculated and therefore we are also not able to classify the observations within this segment.

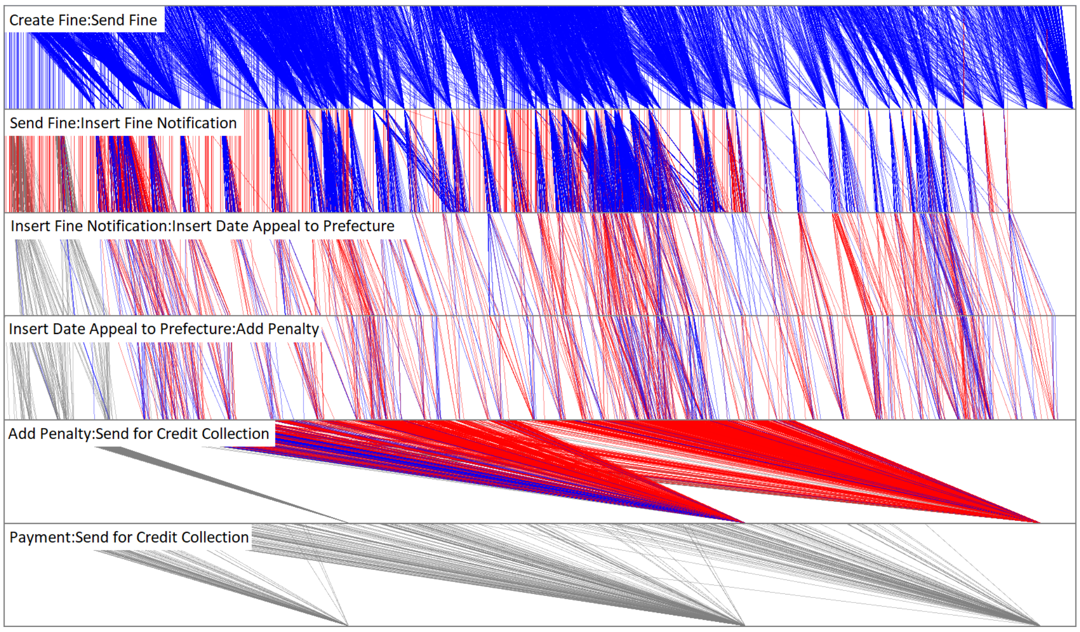

**Figure 6.8:** PSw/EP of segments *Create Fine:Send Fine*, *Send Fine:Insert Fine Notification*, *Insert Fine Notification:Insert Date Appeal to Prefecture*, *Insert Date Appeal to Prefecture:Add Penalty*, *Add Penalty:Send for Credit Collection* and *Payment:Send for Credit Collection*

We first observe that all segment occurrences in the segment *Create Fine:Send Fine* are colored blue, meaning that after the segment *Create Fine:Send Fine* the error exclusively goes up. This could be caused by the fact that the prediction error is just very low when only *Create Fine* is known. However, we also think that this could be caused by the fact that, for all traces, after the activity *Send Fine* occurred, the process either immediately ends (which is the case for 13.6% of the traces) or continues for three or more activities (84.4% of the traces), meaning that the difference in remaining time is very substantial between the two possible outcomes, which can result in a relatively higher error compared to when only *Create Fine* is known.

In the segment *Send Fine:Insert Fine Notification*, we observe two patterns: batching on start and no batching (cases going straight through). We see that the non-batched on start cases predominantly result in the error going down. The reason for this could be that these cases are processed very fast, something that the prediction model does not capture at the prediction point *Send Fine* (as can be seen from the projected predictions in the OPS and OPS' visualizations in the previous), resulting in a higher error. However, no hard conclusions can be drawn from this. The segments *Insert Fine Notification:Insert Date Appeal to Prefecture* and *Insert Date Appeal to Prefecture:Add Penalty* show batching on start and batching on end behavior, respectively. However, nothing conclusive can be said, since the classification w.r.t. the error progression does not reflect a specific pattern.

The final classified segment, *Add Penalty:Send for Credit Collection*, exclusively shows batching on end behavior and the predominant amount of observations are classified with a higher error before this segment than after. This implies that the insight into this segment results in an increase in prediction accuracy.

### Visualize and inspect OPS and OPS'

Next, we visualize both an OPS and an OPS', as introduced in Section 6.3, for each of the frequent segment and prefix-length combinations, i.e. those not in parentheses, from Table 6.3 using the predictions $\mathcal{P}_{test}$. The resulting visualizations can be found in Appendix A.1 and the four most significant observations, which we also identified as interesting based on the PSw/EP, are depicted here in Figures 6.9 to 6.12.

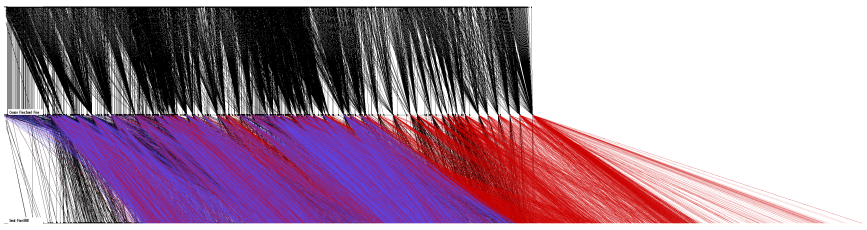

**Figure 6.9:** OPS of predictions of $RM_{p,a,x}$ for segment *Create Fine:Send Fine* for $k = 1$

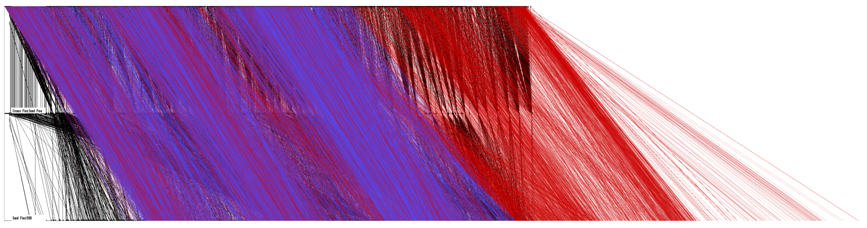

**Figure 6.10:** OPS' of predictions of $RM_{p,a,x}$ for segment *Create Fine:Send Fine* for $k = 1$

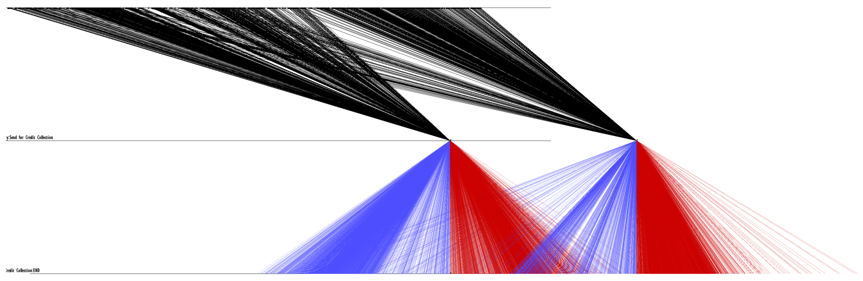

**Figure 6.11:** OPS of predictions of $RM_{p,a,x}$ for segment *Add Penalty:Send for Credit Collection* for $k = 4$

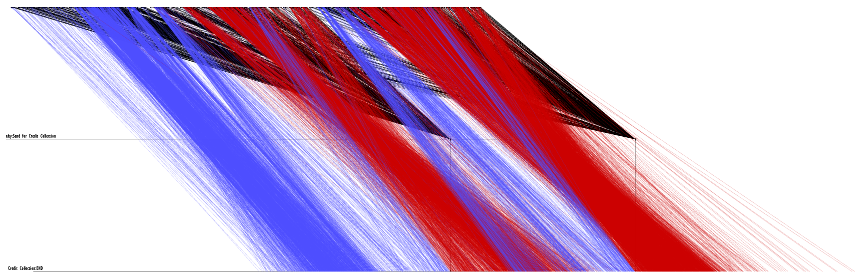

**Figure 6.12:** OPS' of predictions of $RM_{p,a,x}$ for segment *Add Penalty:Send for Credit Collection* for $k = 4$

We recall that the black lines show L1, i.e. the actual segment occurrences in the next segment and the segment of the next step until the end of the case. The red and blue lines show L2, i.e. the predicted segment occurrences from the next activity until the end of the case or from the prediction point until the end of the case, in the OPS and OPS', respectively.

We firstly observe a batching on end pattern in *Create Fine:Send Fine* that we can only detect in L1, i.e. the actual segment occurrences, and not in L2, i.e. the predicted segment occurrences, which could indicate that the model has not picked up on this. However, *Create Fine:Send Fine* solely occurs at $k = 1$, i.e. in the beginning of the process, meaning that a lot can still occur along the way and thus the batching might not even be reflected anymore by the actual outcomes at the end of the process. Therefore it is hard to determine whether this activity could be the cause for the high prediction error. For *Add Penalty:Send for Credit Collection* we also detect a batching on end pattern in L1 that is not detectable in L2, but in this case it occurs at $k = 4$, thus here the fact that the model did not detect batching may be the cause for the high error.

In the other OPS and OPS' visualizations for the RF log shown in Appendix A.1, we observed:

- Unordered behavior in L2 of both the OPS and OPS' for all segments (see Figures 6.9 to 6.12 and A.1 to A.8).
- Three to four points in time in L2 where the lines are more concentrated in the OPS (see Figures 6.9 to 6.12 and A.1 to A.6).).
- A seemingly equal amount of red and blue lines in L2, of which the red lines occur mostly after the batches are processed and the blue lines before (see Figures 6.9 to 6.12, A.1 and A.2).). This indicates overprediction of cases that arrive in the last half of a batch and underprediction of cases that arrive in the first half of a batch (as we can very clearly observe in Figure 6.12).
- Black lines that follow a batching pattern in S2 for OPS' of segments prior to the *Send for Credit Collection* activity (see Figures 6.9, 6.10 and A.1 to A.6). This batching pattern precisely coincides with the batching pattern in S1 of segment *Add Penalty:Send for Credit Collection* (see Figures 6.11 and 6.12).

**Diagnose: Identifying $S$ and $R$ with high impact on error**

Based on the analysis using both the introduced methods, we can determine what inter-case dynamic is the cause of this high prediction error. The two best candidates were the segment *Create Fine:Send Fine* and *Add Penalty:Send for Credit Collection*, because these both showed black lines forming a batching pattern that was not picked up by the red/blue lines. However, for *Create Fine:Send Fine* we cannot draw any hard conclusions because it occurs to early on in the process, therefore, in the remainder, we will solely focus on *Add Penalty:Send for Credit Collection.* As we have also seen, the batching pattern not only applies to the segment *Add Penalty:Send for Credit Collection*, but also to other segments that end with *Send for Credit Collection.* This means that the input for the next phase is the batching pattern and all segments that end with *Send for Credit Collection.* We will write (-,SC) as short hand for all these segments. This results in the input $R$ =batch(e) and $S$=(-,SC).

# Chapter 7

# Derivation of Inter-Case Features for Batching

In Chapter 6 we introduced a methodology to help identify inter-case patterns that cause high prediction errors. While this method can yield many different patterns as output, in this chapter we will only consider the batching pattern. In this chapter will use these insights from Chapter 6 to derive inter-case features to help reduce prediction errors caused by batching, which is the second step towards the inclusion of inter-case dynamics in the life cycle of remaining time prediction as is indicated in Figure 5.1. While the overall method we present is general, we concretely show how to do this for the inter-case dynamics caused by batching.

As discussed in Chapter 3, feature creation is a complex way to extend the input data with additional features and is usually performed manually. It demands extensive analysis of the data and in most cases encompasses multiple transformation steps and in some cases requires additional tooling. Since each machine learning problem is different and so many possible choices can be made regarding steps/techniques/tooling for feature creation, there is no standardized process to approach it. Even more, feature creation is often referred to as being more of an art than science, since it requires human intervention in creatively mixing the existing features [31].

Because there is no standardized process for creating features from input data, let alone for inter-case features for batching, Section 7.1 presents the choices about the steps that we make for the creation of inter-case features for batching and the justification thereof. Sections 7.2 to 7.4 will elaborate on these steps in detail, along with illustrations of these steps applied to the running example.

## 7.1 Inter-Case Feature Creation

The objective is to create inter-case features for batching such that prediction errors caused by batching dynamics can be reduced. In Chapter 3 we found a high prediction error for the RF log using the baseline model $RM_{p,a,x}$ and in Chapter 6 we found that cases that traverse a particular segment $S$ within specific batching patterns $R$ are subject to these high prediction errors. We want to reduce this error prior to the occurrence of such a segment $S = (a, b)$ containing pattern $R$. We aim to do this by leveraging these process and inter-case insights, i.e. $S$ and $R$ from Chapter 6, into an estimate that for each case that has reached activity $a$, i.e. is at the top part of segment $S$, tells us how long that case will spend in $S$, which we can then add as a feature for a remaining time prediction model such as $RM$.

As described, we specifically want to create features for prefixes that have segment $S$ in their next step, such that the arrival time of that case within that segment is known at the moment of prediction. This is especially important for a segment subject to inter-case dynamics, since the time a case will spend in such a segment is highly dependent on the arrival time of that case within that segment, e.g. when a case arrives late in a batch instance, its time in that segment will be much shorter than when it would arrive early.

Before we explain how we want to leverage $S$ and $R$ for engineering features that predict how much time a case spends in $S$ due to $R$, we first want to explain where an inter-case pattern such as batching (in a segment $S$) originates from. A pattern $R$ can actually be traced back to the WfMS that the event data comes from, as described in Chapter 2. Such a WfMS does not only take care of the routing of cases, i.e. the control-flow, but also takes care of resource assignment, scheduling, queuing disciplines and decision logic. The managing of these non control-flow aspects brings about a large set of rules that result in underlying process mechanisms not explicitly recorded in the event log. These mechanisms, such as for example FIFO or batching, turn up on the PS in the form of a pattern. Figure 7.1 illustrates how a WfMS relates to an event log that is used for prediction and in turn relates to a pattern. To create features that explain these mechanisms, we want to "reverse engineer" this uncovered pattern back to its origin: the rules that this pattern originates from. When we are leveraging process and inter-case insights, i.e $R$ and $S$ from Chapter 6, we are thus essentially uncovering these WfMS rules. If we have these rules, we can use them to create features which we can provide to the prediction model such that it can make more informed predictions for cases subject to these rules.

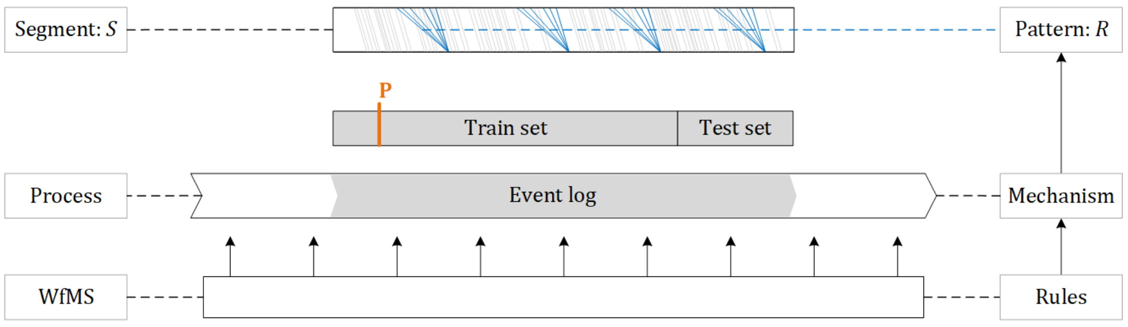

**Figure 7.1:** Relation between WfMS, process and performance pattern

**Table 7.1:** Example of existing features for model $RM$ (columns in black), features we want to add (columns in red, blue and green) and the output side for $RM$ (column in purple)

| Activity | Time | ... | ... | ... | $c_S$ | $\bar{c}_S$ | $c_R$ | $\bar{c}_R$ | $d$ | $\bar{d}$ | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $X_1$ | $X_2$ | $X_3$ | ... | $X_n$ | $X_{n+1}$ | $\overline{X}_{n+1}$ | $X_{n+2}$ | $\overline{X}_{n+2}$ | $X_{n+3}$ | $\overline{X}_{n+3}$ | $y$ | $\overline{y}$ |
| A | 21-5 13:17 | ... | ... | ... | 0 | 1 | 0 | 0 | - | - | 127 | 130 |
| C | 24-5 14:05 | ... | ... | ... | 1 | 1 | 1 | 1 | 254 | 210 | 270 | 256 |
| C | 30-5 13:08 | ... | ... | ... | 1 | 1 | 1 | 1 | 187 | 156 | 178 | 206 |
| C | 15-7 12:36 | ... | ... | ... | 1 | 1 | 0 | 1 | - | 142 | 287 | 301 |
| B | 21-7 16:13 | ... | ... | ... | 0 | 0 | 0 | 0 | - | - | 301 | 247 |
| C | 27-7 18:04 | ... | ... | ... | 1 | 1 | 1 | 0 | 203 | - | 183 | 191 |
| C | 05-8 12:20 | ... | ... | ... | 0 | 0 | 0 | 0 | - | - | 93 | 80 |
| A | 06-8 21:00 | ... | ... | ... | 0 | 0 | 0 | 0 | - | - | 45 | 44 |

Table 7.1 shows an example table of features used to train a remaining time prediction model $RM$. The columns in black show the existing intra-case features. We want to leverage $S$ and $R$

by consecutively adding columns to the existing features that indicate for each prefix (1) whether it will go through segment $S$, red column in Table 7.1, (2) whether it will also go through pattern $R$, blue column in Table 7.1, and (3) how long it will last in $S$ as a result of the pattern $R$, green column in Table 7.1. Conceptually, we can then use these three features as additional input features to learn a new, better prediction model for the remaining time $y$ (purple column in Table 7.1).

The creation of each of these new features $X_{n+1}$ (indicating that prediction $P$ will go through $S$, written as $c_S$ for short), $X_{n+2}$ (indicating that prediction $P$ will go through pattern $R$ in $S$, written as $c_R$), $X_{n+3}$ (for the time until the batch is processed, written as $d$) requires its own prediction models $CM_S$, $CM_{S,R}$ and $TM$, respectively. Figure 7.2 gives an overview of the different steps necessary to build these models (offline) and the steps that need to be executed at run time.

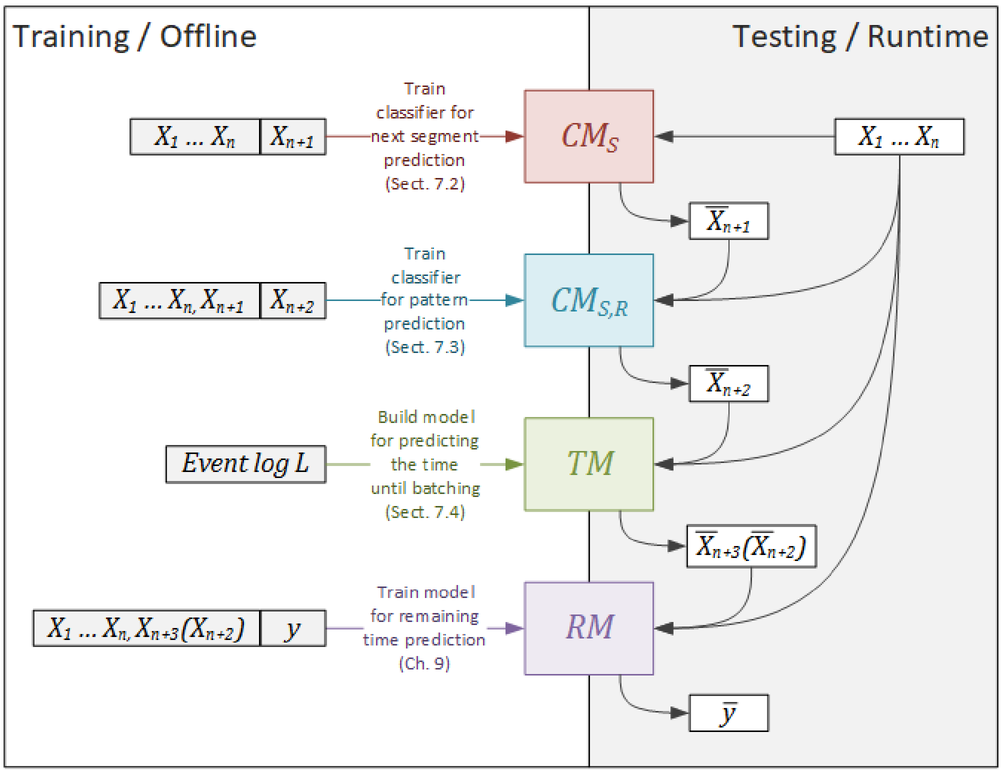
Training / Offline
Testing / Runtime
X1 ... Xn | Xn+1
Train classifier for next segment prediction (Sect. 7.2)
CMS
X1 ... Xn
X̄n+1
X1 ... Xn, Xn+1 | Xn+2
Train classifier for pattern prediction (Sect. 7.3)
CMS,R
X̄n+2
Event log L
Build model for predicting the time until batching (Sect. 7.4)
TM
X̄n+3(X̄n+2)
X1 ... Xn, Xn+3(Xn+2) | y
Train model for remaining time prediction (Ch. 9)
RM
ȳ

**Figure 7.2:** Overview of feature creation steps presented in this chapter

In the offline phase, we need to:

1. Create a bucket of prefixes that all have $e_k = a$, based on segment $S = (a, b)$ we retrieve through the error analysis from Chapter 6.
2. Learn a classification model $CM_S$ that predicts whether a prefix at prediction point $a$ will go through segment $S$ (Section 7.2).
3. Learn a classification model $CM_{S,R}$ that predicts whether a prefix that goes through segment $S$ will also go through pattern $R$ (Section 7.3).
4. Create a model for predicting (an abstraction of) the remaining time until the prefix is processed in a batch (Section 7.4).
5. Create a model for predicting the remaining time $RM$

In the online phase, we need to:

1. Take the features $X_1...X_n$ to predict $\overline{c}_S$ using model $CM_S$.
2. Take the features $X_1...X_n, \overline{c}_S$ to predict $\overline{c}_R$ using model $CM_{S,R}$.
3. Take the features $X_2, \overline{c}_R$ to predict $\overline{d}$ using model $TM$.
4. Take features $X_1...X_n, \overline{d}(\overline{c}_R)$ to predict the remaining time $\overline{y}$ using model $RM$. Here we specifically choose to only use the feature $\overline{d}$ in the prediction if $\overline{c}_R = 1$. Prefixes for which $\overline{c}_R = 0$ will not get a feature $\overline{d}$, therefore we will write $\overline{d}(\overline{c}_R)$.

Since we are trying to create features for event data, which is subject to time, we can for each prediction during the online phase only use the knowledge of the process up until that point. Take for example prediction $P$, highlighted in orange in Figure 7.1. If we want to create a feature for $P$, we can only use the data up until the moment of prediction $t_P$. But when we derive these rules from batching pattern $R$ found in Chapter 6, we use the entire event log and thus also knowledge that we do not have at $t_P$. However, we assume these rules in the WfMS apply to the entire process and were already in place preceding the recording of the event log, meaning that for our own evaluation, we can assume that even for predictions at the start of this event log, this knowledge was already available.

## 7.2 Next Segment Prediction

In this section, we address the learning of a classification model $CM_S$ to predict whether a case will go through a particular segment $S$. Through the error analysis in Chapter 6, we have received a segment $S$ and based on this we have selected prefixes $\mathcal{P}$ from a bucket of prefixes $B$ that all have *reached* segment $S = (a, b)$, i.e. of which their common prediction point is $a$. We need to select the subset $\mathcal{P}_S \subseteq \mathcal{P}$ (rows in the Table 7.1) that will all go through segment $S$, i.e. will have activity $b$ in their next step. To determine what prefixes need to be in $\mathcal{P}_S$, we use event log filtering to check for each prefix' suffix whether $e_{k+1} = b$.

Once we know what prefixes need to be in $\mathcal{P}_S$, we need to create a data set for training a classification model $CM_S$ that predicts whether a case will be in segment $S$ next. We take the prefixes $\mathcal{P}$ and add a positive label (expressed by a 1) for each prefix $\in \mathcal{P}_S$ and add a negative label (expressed by a 0) for each prefix $\notin \mathcal{P}_S$. When each prefix is labeled, we train a classification model $CM_S$ using features $X_1...X_n$ to predict label $c_S$ (short for $X_{n+1}$) as illustrated in Table 7.1. Essentially, training a model to predict whether a case will be in a certain next step is next activity prediction, for which various methods have already been introduced [16, 7]. Since the choice of such a method is not specific to the problem of creating inter-case features we address here, we will not discuss this any further.

Now that we have classification model $CM_S$, we need to determine whether the output is reliable enough such that we can use it as input for the next step. We evaluate reliability of the model not only based on accuracy but also based on precision. We do this because the time a case spends within a segment can vary significantly per segment. When we classify a prefix to be part of a "slow" segment while it is actually part of a fast going segment, we will in the end be giving our remaining time prediction model $RM$ the complete opposite information than that it should get. Such an example of a false positive can completely disturb a model $RM$ and therefore also its reliability. Therefore if the output of a classification model $CM_S$ turns out to be unreliable, the feature $\overline{c}_S$ should be discarded.

Once we have built a reliable classification model $CM_S$, we can use it at runtime to predict $\overline{c}_S$ from input features $X_1...X_n$. This outcome will then be passed to the next prediction model $CM_{S,R}$.

Running Example

The input from the fine-grained error diagnosis from Chapter 6 is $S$=(-,Send for Credit Collection (SC)) and $R$=batch(e). Our first step is to detect which prefixes in $L$ will go through the segment (-,SC). We used the Pandas library in Python to label each prefix of $k \in \{4,5\}$ with a label for whether *Send for Credit Collection* will occur in the next step. This results in two sets of labeled prefixes for training $L^*_{4,train}$ and $L^*_{5,train}$ and two sets of labeled prefixes for testing $L^*_{4,test}$ and $L^*_{5,test}$. We will only build models for $k \in \{4,5\}$, because only for these $k$ the next step of the prefix goes through (-,SC).

To build the classification model, we used the implementation of the benchmark for outcome-oriented predictive process monitoring from [24], which is available at `https://github.com/irhete/predictive-monitoring-benchmark`. Similar to $RM_{p,a,x}$, we have chosen a prefix bucketing method, aggregate encoding method and an extreme gradient boosting algorithm. We have applied hyper-parameter optimization (which was also available in the benchmark implementation) using our own labeled training data. After optimizing the parameters using grid search [11], we trained classification models $CM_{4,\text{(-,SC)}}$ and $CM_{5,\text{(-,SC)}}$ using $L^*_{4,train}$ and $L^*_{5,train}$. We tested the models using $L^*_{4,test}$ and $L^*_{5,test}$, of which the results are depicted in Table 7.2.

**Table 7.2:** $CM_S$ results for next segment classification on testing data of the RF log for $k \in \{4,5\}$

| Model | TP | TN | FP | FN | Accuracy | Precision |
|---|---|---|---|---|---|---|
| $CM_{4,\text{(-,SC)}}$ | 8791 | 1520 | 2739 | 1351 | 0.716 | 0.762 |
| $CM_{5,\text{(-,SC)}}$ | 46 | 4011 | 209 | 276 | 0.893 | 0.180 |

We can see that there is a large difference in the results for the different models. While $CM_{4,\text{(-,SC)}}$ has a higher precision, $CM_{5,\text{(-,SC)}}$ has a higher accuracy. We consider the accuracy of $CM_{4,\text{(-,SC)}}$ still acceptable, but the precision of $CM_{5,\text{(-,SC)}}$ is too low. Therefore we will only consider $CM_{4,\text{(-,SC)}}$ and at runtime only make a classification $\overline{c}_S \in \{0,1\}$ for prefixes of length $k = 4$ and not for prefixes of length $k = 5$.

## 7.3 Pattern Prediction

In this section, we address the learning of a classification model $CM_{S,R}$ to predict whether a case will go through a specific pattern $R$. In Section 7.2 have selected prefixes $\mathcal{P}_S \subseteq \mathcal{P}$ that will all go through segment $S$, i.e. of which their prediction point and next step are $a$ and $b$, respectively. We need to select the subset $\mathcal{P}_{S,R} \subseteq \mathcal{P}$ (rows in the Table 7.1) that will all be part of $R$, i.e. part of a batch. To determine which prefixes will be in $\mathcal{P}_{S,R}$, we use the batch miner [2] to detect for each prefix whether its next segment is part of a batch.

We add labels to each prefix $\mathcal{P}_{S,R}$ similar to as described in Section 7.2. We train a classification model $CM_{S,R}$ using features $X_1...X_n, c_S$ to predict label $\overline{c}_R$ (short for $\overline{X}_{n+2}$) as illustrated in Table 7.1 and evaluate it based on accuracy and precision.

Once we have built model $CM_{S,R}$, we can use it at runtime to predict $\overline{c}_R$ from input features $X_1...X_n, \overline{c}_S$, which will be passed to the next prediction model $TM$.

Running Example

In this step we would have to build a classification model that predicts for all prefixes that go through $S$=(-,SC), whether they will also be subject to $R$=batch(e). In our case, all cases that traverse the *Send for Credit Collection* activity are batched, meaning that we do not need to build a classification model in this step and we can directly use the classification $\overline{c}_S$, i.e. $\overline{c}_R = \overline{c}_S \in \{0, 1\}$. We can now use $\overline{c}_R$ as input to the third prediction model in the next section.

## 7.4 Time To Batch Prediction

In this section, we address building a model $TM$ for predicting a certain "distance" or time from the arrival of a case until it is batched. In Section 7.3 we have selected a set of prefixes $\mathcal{P}_{S,R} \subseteq \mathcal{P}$ that are part of the pattern $R$ in segment $S$ at their moment of prediction $t_P$. We want to predict for all these prefixes how long they will be in $S$ subject to $R$. Since we will only consider the batching pattern for $R$, we need to predict how long each case will wait in $S$ until it is processed in a batch, which we will define as the *predicted time to batching* $\overline{t}_R$.

To predict $\overline{t}_R$, we want to build a remaining time prediction model for batching $TM$. This model uses features $X_2$, i.e. the start time of a case in segment $S$, and $c_R$, i.e. the classification of whether the case will be in $R$, as input. This model needs to somehow leverage the starting location of cases in $R$, i.e. their moments of prediction $t_P$ (which is also $X_2$), into a prediction $\overline{d}$ (short for $\overline{X}_{n+3}$). Therefore we want to provide it with some context of the batching pattern $R$ that each case will be in (7.4.1 and 7.4.2), such that it can predict, given the starting time of that prefix in $S$, in which next batch that case will be processed and when it will be processed, from which we can then derive the time until batching $\overline{t}_R$ (7.4.3). As an alternative to the concrete value $\overline{t}_R$, we also propose an abstracted form of this estimate $\overline{p}$ (7.4.4).

### 7.4.1 Deriving Context for Batching

We first need to determine which parameters we need to describe to build a prediction model $TM$. These parameters need to describe the batching behavior such that we can provide the model with a specific batching context. Therefore, we will use a couple of the batching parameters defined by [2], illustrated in Figure 7.3. At the top of the segment we can see the time at which the first case arrives in a batch $t_{i,start}$ and the time at which the last case arrives in a batch $t_{i,end}$. At the bottom of the segment we can see the batch moment, i.e. the time $BM_i$ at which the batch is processed and the time $BM_{i-1}$ of the previous batch. From each of these points we can derive parameters that together describe the triangular shape of this batch and its position w.r.t. the preceding batch. The batch interval $BI_i = BM_i - BM_{i-1}$ is the time between two successive moments of batching and the minimum waiting time $W_{i,min}$ is the waiting time of the last arriving case in batch $b_i$ [2].

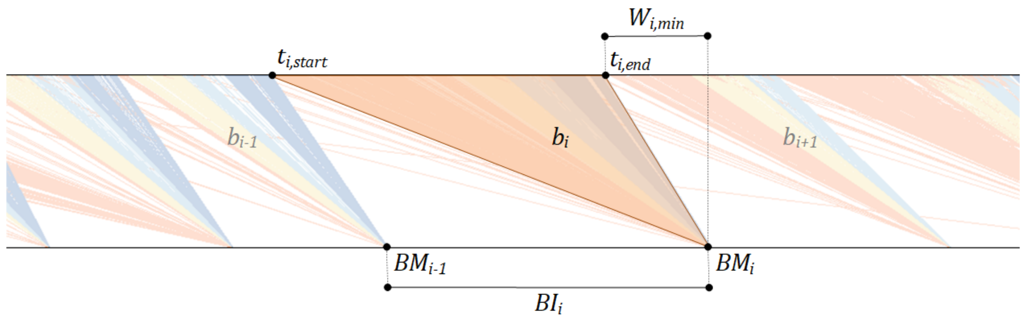

**Figure 7.3:** Batch in performance spectrum annotated with batching parameters

We can also see from Figure 7.3 that there are batches overlapping. When this happens, we cannot distinguish which cases will be part of the fist batch and which will be part of the next batch based on their moment of prediction alone and we will therefore not take this into consideration.

### 7.4.2 Deriving Batch Context Parameters

Next, we want to make this batching context concrete by deriving the values for each of these parameters such that we can build a training set. For this we will use the batch miner from [2]. This tool can take a segment from the PSM data [8] as input and returns among others $BM_i$, $BI_i$ and $W_{i,min}$ of each batch $b_i$ in that segment. As a result, we have perfect knowledge about which cases belong in which batch and when each batch is going to be processed.

However, since we are dealing with time, the information that we really have at $t_P$ is limited to that of the last batch that is processed $b_{i-1}$, namely $BM_{i-1}$, $BI_{i-1}$ and $W_{i-1,min}$. Figure 7.4 illustrates an arrival period between two consecutive batch moments $BM_{i-1}$ and $BM_i$ of which all cases have the same information about the context parameters of all batches up until the last closing batch. When we have a case arriving in such an arrival period, i.e. $t_P > BM_{i-1}$, we want to know the parameters of the batch $b_i$ that correspond to this arrival, not those of the preceding batch $b_{i-1}$.

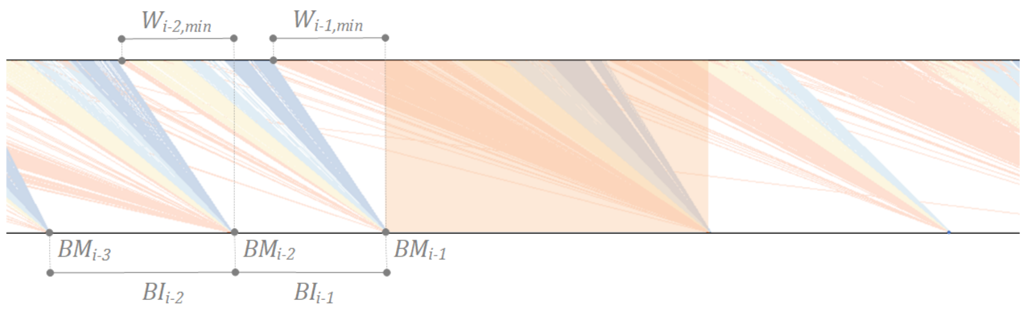

**Figure 7.4:** Illustration of an arrival period annotated with past parameter values

We want to derive the parameter values of a current batch $b_i$ from the parameter values of the preceding batches $b_1, ..., b_{i-1}$ by the use of *forecasting*. Forecasting encompasses a collection of techniques that aim to construct a time series model that estimates a trend in the data [32]. We want to use these techniques to create a model that can forecast $BI_i$ and $W_{i,min}$ based on past observations $BI_h ... BI_{i-1}$ and $W_{h,min} ... W_{i-1,min}$, respectively. While traditional forecasting techniques assume data that is recorded at fixed intervals, e.g. the monthly sales or market growth, the parameter $BI$ encompasses the interval itself, which has varying lengths and is thus not fixed. However, since this is the target we actually want to forecast, we will for this case overlook this discrepancy.

Multiple forecasting techniques have been introduced, but for simplicity, we will only consider the two most basic trend estimation methods [32]. The first, *moving averages*, takes into account a specific number of past observations and computes the average, which will then be the forecast for the next period. The second, *exponential smoothing*, contrary to moving averages, does not weigh the past observations equally, but instead uses a smoothing factor $\alpha$ to assign exponentially decreasing weights over time, with $0 < \alpha < 1$. The higher the values of $\alpha$, the more weight is assigned to recent observations and the lower the values of $\alpha$, the less responsive they are to recent changes. By using the actual intervals from the training data, we can evaluate which forecasting technique or which $\alpha$ performs best by comparing it with the actual lengths of the intervals and choose the best configuration.

Based on this method, we can create forecasting models $F_{BI}$ and $F_{W_{min}}$ for $BI$ and $W_{min}$, respectively. We now want to use these forecasting models to derive all necessary parameters for the context of $R$. First we forecast $\overline{BI}_i$ and $\overline{W}_{i,min}$, which we then use together with the previous batch moment $BM_{i-1}$ to derive the forecasted point in time until which cases will still be included in the current batch: $\bar{t}_{i,end} = BM_{i-1} + \overline{BI}_i - \overline{W}_{i,min}$. Figure 7.5 again illustrates an arrival period, this time annotated with all previously derived parameters.

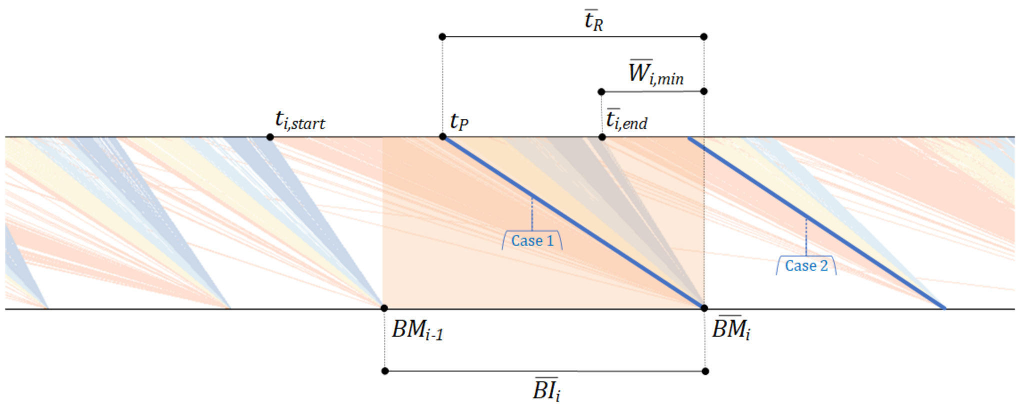

**Figure 7.5:** Illustration of an arrival period annotated with predicted parameters values

## 7.4.3 Predicting the Time Until the Next Batch

Now that we have all the necessary parameters, we can use these to predict the time until batching $\bar{t}_R$ for each case that arrives in $S$ based on its arrival time, i.e. its moment of prediction $t_P$. For each of these cases, we consider the parameter values that are known for the period it arrives in, for which we again refer to Figure 7.5.

Within such an arrival period, cases can end up in the current batch $b_i$ (see ase 1 in Figure 7.5) or in the next batch $b_{i+1}$ (see case 2 in Figure 7.5). If a case arrived in the current batch $b_i$, i.e. $t_P \leq \bar{t}_{i,end}$, we predict $\bar{t}_R = BM_{i-1} + \overline{BI}_i - t_P$. If a case arrived in the next batch $b_{i+1}$, i.e. $t_P > \bar{t}_{i,end}$, we predict $\bar{t}_R = BM_{i-1} + 2 \cdot \overline{BI}_i - t_P$. In the exceptional case where $BI_i$ is much longer than $\overline{BI}_i$ and the next batch moment $BM_i$ is therefore still not known, we additionally need to check for cases that arrive after the arrival of the last case of the second next batch $\bar{t}_{i+1,end} = BM_{i-1} + 2 \cdot \overline{BI}_i - \overline{W}_{i,min}$. In this case, $\bar{t}_R = BM_{i-1} + 3 \cdot \overline{BI}_i - t_P$.

## 7.4.4 Predicting the Batch Partition

In addition to $\bar{t}_R$, we also want to create features using an abstraction of the "distance" until the next batch. Figure 7.6 shows how we want to derive this abstraction.

We want to divide a batch $b_i$ into $n$ equal parts and then we want to derive in which of these parts a case arrived based on the arrival time of that case within $S$, i.e. the moment of prediction $t_P$. For this we need to know the length of the period in which cases arrive in batch $b_i$, i.e. the time from $t_{i,start}$ to $t_{i,end}$, which we will define as $BI'_i$, also illustrated in Figure 7.6. The time the first case of $b_i$ arrives is the same as the time the last case of $b_{i-1}$ arrives and therefore $t_{i,start} = t_{i-1,end} = BM_{i-1} - W_{i-1,min}$. The predicted time the last case of $b_i$ arrives $\bar{t}_{i,end}$ we have already derived in the previous subsection. Now we can predict the length of the arrival period of $b_i$ as follows: $\overline{BI'}_i = \bar{t}_{i,end} - t_{i-1,end}$.

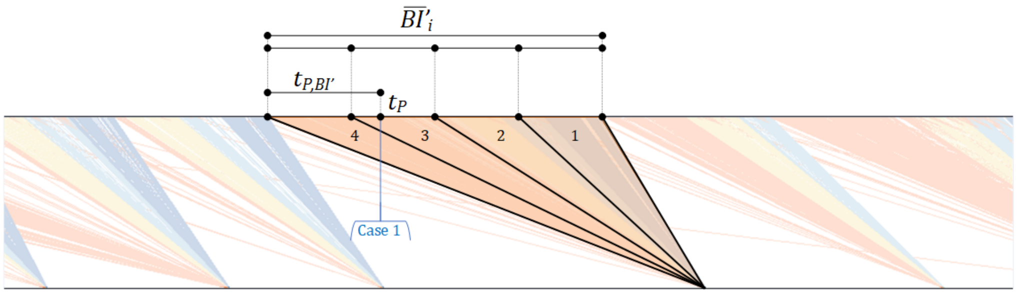

**Figure 7.6:** Illustration of a batch partitioned into $n = 4$ equal parts

To know in which part of $b_i$ to partition a case, we want to know its arrival time within this arrival period $\overline{BI'}_i$, which we will define as $t_{P,BI'}$. This is the time that has lapsed since the first arrival of a case within $b_i$ and the arrival time of the case in question, i.e. $t_{P,BI'} = t_P - t_{i,start}$, also illustrated in Figure 7.6. Then we derive for each case in which of the partitions of $b_i$ it arrived by using the fraction $f_P = t_{P,BI'}/\overline{BI'}_i$. If $f_P < (1 \cdot \overline{BI'}_i)/n$ the case arrived in the leftmost partition (annotated with 4 in Figure 7.6), if $f_P < (2 \cdot \overline{BI'}_i)/n$ the case arrived in the second to leftmost partition (annotated with 3 in Figure 7.6) and so on. The partition that each case arrives in we will use as the *predicted batch partition* $\overline{p}_n$, where $n$ is the number of partitions. These numbers are decreasing towards $BM_i$, since the actual time towards that moment is also decreasing and would therefore be better interpretable for a machine learning algorithm.

Figure 7.6 shows how the arrival period $\overline{BI'}_i$ is partitioned into $n = 4$ equal parts and additionally highlights the arrival of a case. In this example, we would create a feature with value 3 for case 1.

Running Example

From the previous step, we receive a set of prefixes $\mathcal{P}_{\text{(-,SC),batch}}$ that are all subject to pattern $R =$batch(e) in segment $S =$(-,SC). For each of these prefixes we are going to predict the time until batching $\overline{t}_R$ or the batch partition $\overline{p}_n$ with $n \in \{4, 8, 10, 20\}$, which will in the end be used as input $\overline{d}$ for model $RM$.

**Deriving Context Parameters**

For this we have applied the batch miner from [2], such that we can extract $BM_i$, $BI_i$ and $W_{i,min}$ for each batch in the segment (-,Send for Credit Collection) of the RF log. The results are depicted in Table A.1. We subsequently applied exponential smoothing using $\alpha \in \{0.1, 0.2, 0.3, 0.4, 0.5\}$ for $BI_i$ and $W_{i,min}$ and calculated the MAE of these forecasts. The results are depicted in Tables A.2 and A.3. We found $\alpha = 0.3$ to have both the lowest MAE for $W_{min}$ as well as for $BI$ and we will use these forecasts to calculate the parameters of each current batch $b_i$.

**Predicting the time until the next batch**

We have for each prefix in $\mathcal{P}_{\text{(-,SC),batch}}$ predicted $\overline{t}_R$ using the last batch moment $BM_{i-1}$ and the forecasts of the current batch $BI_i$, $W_{i,min}$. To illustrate the result of these predictions, we have created a new event log where we replaced the actual timestamps for *Send for Credit Collection* with our predicted timestamps. This event log we loaded in the PSM and the results for the segment *Add Penalty:Send for Credit Collection* are presented alongside the actual segment in Figure 7.7.

As can be seen, the forecasted batch moments and actual batch moments coincide quite nicely. Unfortunately, somewhere after half of the time span we can see that there is quite a long batch interval that our forecasts have not yet picked up on and there the batch moments do not correspond to one another. But since our smoothing parameter of $\alpha = 0.4$ assigns quite some weight to recent observations, the long batch interval is picked up on and the length of the forecasted batch intervals increases a bit and as a result the forecasted batch moments coincide again with the actual batch moments. We will report on exact errors of this method used in a remaining time prediction approach in Chapter 9.

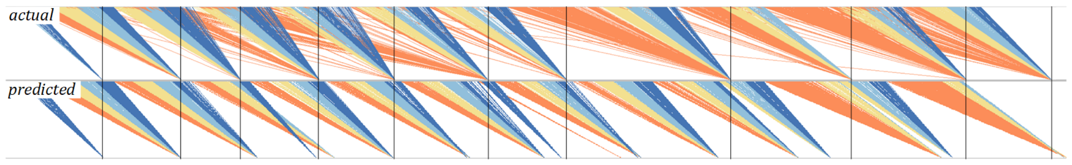

**Figure 7.7:** Performance spectrum of the actual (top) and estimated (bottom) timestamps for the segment *Add Penalty:Send for Credit Collection*

### Abstracting the time until the next batch

In addition to the $\bar{t}_R$, we also want to create features using the predicted batch partition $\overline{p}_n$. For this we have predicted batch partitions using $n \in \{4, 8, 10, 20\}$. The results of this we unfortunately cannot show on the PS.

As a result of these two methods and the pattern classification we retrieved in Section 7.3, we have created a set of inter-case features $\overline{d} \in \{\bar{t}_R, \overline{p}_4, \overline{p}_8, \overline{p}_{10}, \overline{p}_{20}\}$. The result of selecting one of these as an additional feature for the remaining time prediction model will be presented in Section 9.1.

# Chapter 8

# Derivation of an Inter-Case Evaluation for Batching

In Chapter 6 we introduced a methodology to help identify inter-case patterns that cause high prediction errors. In this chapter we will use these insights to derive an inter-case error evaluation, which is the third step towards the inclusion of inter-case dynamics in the life cycle of remaining time prediction as is indicated in Figure 5.1.

Currently, performance of remaining time prediction methods have solely been evaluated using the MAE, which is an aggregate of all prediction errors of the remaining time. However, the inter-case dynamics (such as batching) that were uncovered in Chapter 6 clearly very much dominate the control-flow and case-level outcomes. Therefore, and also because we have presented methods for including inter-case features, e.g. batching, in the prediction model, we deem it consistent that predictions can be also measured from this batching perspective.

To manage that, we first want to introduce an additional measure that better respects these batching dynamics that are part of a process, which will be presented in Section 8.1. Second, as we have previously mentioned, the MAE of the predicted remaining time is just a single number and does not give any meaningful insights into its underlying distribution. In Section 8.2 we introduce the use of histograms to visualize the remaining time measurements and those of the measure we introduce in Section 8.1. Finally, in Section 8.3 we will split up these sets of measurements into those of cases that were part of a batch and those that were not part of a batch, such that we can compare the performance among these two sets of cases.

## 8.1 Measuring Interdeparture Time

When we evaluate the accuracy of the remaining time prediction of a process by calculating the MAE, we use a set of predicted remaining times and a set of actual remaining times and for each case separately compare the actual value $y_P$ to the predicted value $\overline{y}_P$. However, as has been discussed multiple times now, we are dealing with a process where it is inherent that cases behave as a collective as opposed to in isolation. While separately these outcomes all say something about a case individually, together they can say something about:

- The behavior of the process at a moment in time, e.g. a lot of cases to be predicted to end at approximately the same time as a result of batching.
- The order of cases being processed, e.g. cases to be predicted to end in the same or reverse order as a result of FIFO or LIFO, respectively.

Because we focus on batching we want to evaluate the first of these two; we want to evaluate if cases that go through a segment with batching are generally predicted to end up closer together. We want to do this my measuring the *interdeparture time* between cases, which is the amount of time that lapses between two successive end times of cases in the process. An example of the interdeparture time is illustrated on a PS segment in Figure 8.1, where the top of the segment is the point of prediction $e_k$ and the bottom of the segment is the end of the process $e_{|\sigma|}$.

For a set of predictions $\mathcal{P} = P_1...P_n$, we want to compare the interdeparture times of cases using their actual end times to the interdeparture times of cases using their predicted end times. The given predictions $\mathcal{P} = P_1...P_n$ give $n$ *actual* end times $t_{e_{|\sigma_1|}}...t_{e_{|\sigma_n|}}$ and $n$ *predicted* end times $\overline{t}_{e_{|\sigma_1|}}...\overline{t}_{e_{|\sigma_n|}}$. If we sort the actual and the predicted end times, we obtain two sequences $t_1...t_n$ and $\overline{t}_1...\overline{t}_n$ of actual and predicted end times, respectively. From these sequences we get $n-1$ actual interdeparture times $ID_i = t_{i+1} - t_i$ and predicted interdeparture times $\overline{ID}_i = \overline{t}_{i+1} - \overline{t}_i$.

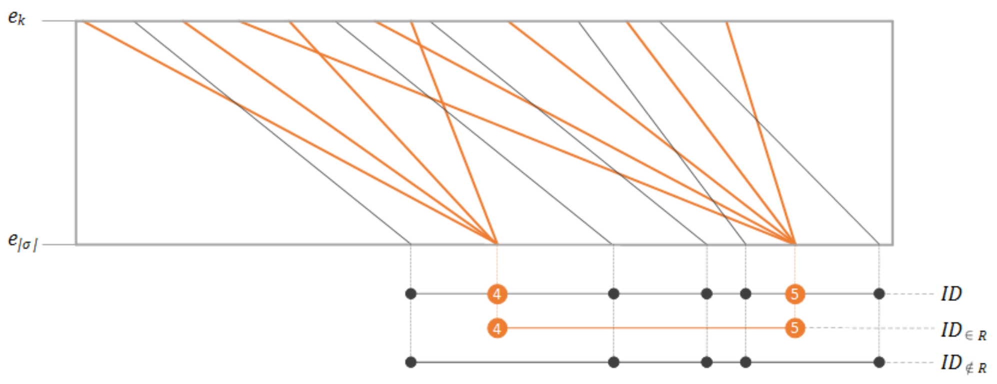

**Figure 8.1:** Illustration of interdeparture time measurements on the performance spectrum

As can already be seen in Figure 8.1, cases that are in the same batch have an interdeparture time of 0, whereas cases in two subsequent batches have an interdeparture time equal to $BI_i$ of the second batch. If the behavior of a process with batching is predicted accurately, then the interdeparture times of the predicted outcomes must also reflect this, i.e. lots of near zero interdeparture times and some long interdeparture times.

## 8.2 Introducing a Histogram-Based Evaluation

We have a set of predictions $\mathcal{P} = P_1...P_n$ with $n$ actual remaining times $y_1...y_n$ and $n$ predicted remaining times $\overline{y}_1...\overline{y}_n$ which we directly retrieved from model $RM$. We have also derived $n-1$ actual interdeparture times $ID_1...ID_{n-1}$ and $n-1$ predicted interdeparture times $\overline{ID}_1...\overline{ID}_{n-1}$ using the method in the previous section.

We now want to take both of these sets of measurements and compare the predicted against the actual values. Instead of calculating the error for each $P$ and aggregating this error over all $P$, we want to create histograms. This way we can avoid a one-on-one comparison of actual and predicted outcomes that the interdeparture time measure does not allow, and, above all, it allows us to gain insights into the underlying distributions.

We first create one histogram for all actual remaining times $y_1...y_n$ and one histogram for all predicted remaining times $\overline{y}_1...\overline{y}_n$. This could for example reveal that the outcomes of a prediction model more or less converge to some mean, while in reality there is a large variance in these outcomes, which we will also discover in our running example later on.

We also create histograms for all actual interdeparture times $ID_1 ... ID_{n-1}$ and one histogram for the predicted interdeparture times $\overline{ID}_1 ... \overline{ID}_{n-1}$. If batching would be present, the actual interdeparture times should reflect a high frequency of short interdeparture times (within a batch) and some longer interdeparture times (between the batches), as also illustrated by $ID$ in Figure 8.1. If the histogram of the predicted interdeparture times does not reflect this, it could indicate that the prediction model has not detected the batching mechanism.

In addition to these histograms, we would also like to measure the difference between the distributions of the predicted and actual outcomes. We aim to do this by calculating their statistical distance by means of the earth mover's distance *emd*.

Running Example

As discussed in Chapter 5, we have run model $RM_{p,a,x}$ from [4] for each prefix $hd^k(\sigma) \in L_{test}$ with $k = 4$, resulting in a set of predictions $\mathcal{P}_{4,test}$. We choose $k = 4$ because this is the step in the process at which the batching occurs.

From $\mathcal{P}_{4,test}$ we extract all actual and predicted remaining times and we calculate the actual and predicted interdeparture times as described in Section 8.1. The histograms for $y$ and $\overline{y}$ are depicted in Figure 8.2 and the histograms for $ID$ and $\overline{ID}$ are depicted in Figure 8.3

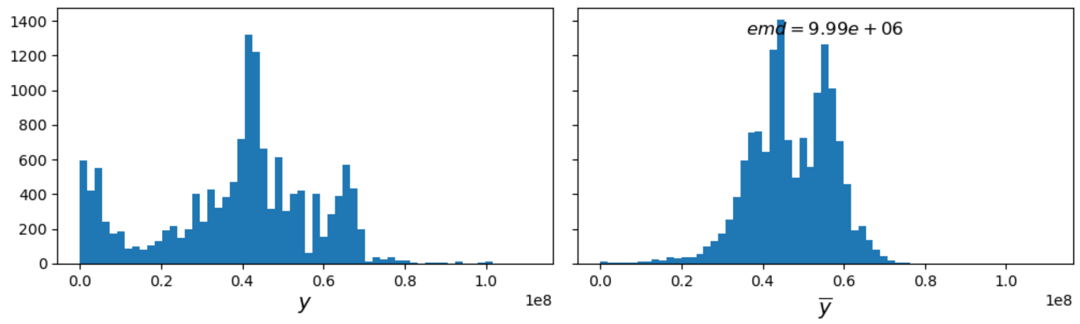

**Figure 8.2:** Histograms of $y$ and $\overline{y}$ of $P_{4,test}$ retrieved using model $RM_{p,a,x}$

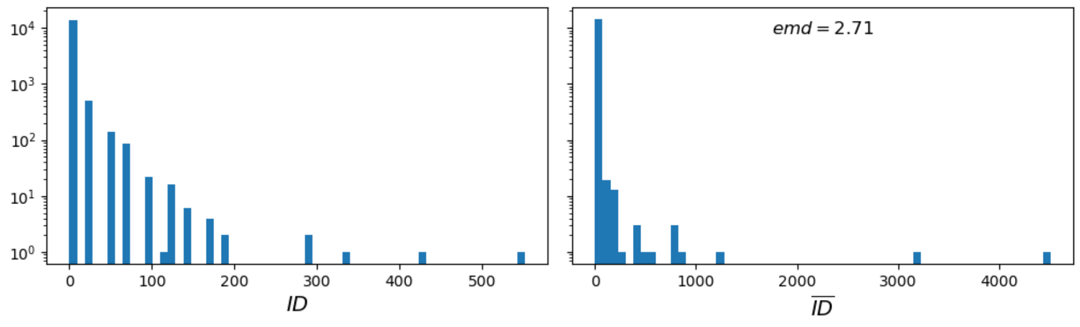

**Figure 8.3:** Histograms of $ID$ and $\overline{ID}$ of $P_{4,test}$ retrieved using model $RM_{p,a,x}$

In Figure 8.2 we observe that $\overline{y}$ is mostly distributed between $0.2 \cdot 10^8$ and $0.7 \cdot 10^8$ with peaks around $0.45 \cdot 10^8$ and $0.55 \cdot 10^8$. While $y$ has the same peak at around $0.45 \cdot 10^8$, it shows two other peaks at $0.05 \cdot 10^8$ and $0.65 \cdot 10^8$. Additionally, $y$ also shows several observations between $0.8 \cdot 10^8$ and $1.1 \cdot 10^8$, as opposed to $\overline{y}$. As we already expected to see in this case for batching behavior, $y$ shows a much higher variance than $\overline{y}$. We will elaborate more on this in the next running example section.

In Figure 8.3 we observe similar peaks close to 0 for both $ID$ and $\overline{ID}$. However, $\overline{ID}$ shows some observations $> 4000$, while $ID$ does not show observations $> 550$. This indicates a discrepancy in departure behavior we will further elaborate on in Section 8.3.

For both of the underlying distributions of the predicted outcomes, we have also computed the *emd*. For the distributions of the remaining time, $emd_y = 10 \cdot 10^6$ and for the interdeparture time, $emd_{ID} = 2.71$. These values are additionally depicted in Figures 8.2 and 8.3. For now, we can make no use of this metric, because we are currently only evaluating the baseline itself. Later it will be used to evaluate and compare the performance of different created features in prediction model $RM_{p,a,x}$.

## 8.3 Comparing Performance for Batched and Non-Batched Cases

In Sections 8.1 and 8.2 we have introduced an additional measure for comparing predicted and actual outcomes of the interdeparture time as well as those of the remaining time. We now want to split up these sets of measurements into those of cases that were part of a batch and those that were not part of a batch, such that we can evaluate the performance of a model $RM$ specifically for batched cases and non-batched cases. If we can see a clear difference in the distributions of predicted outcomes for batched and non-batched cases, the prediction model was apparently able to make a distinction. If this difference is substantial and reflects the characteristics of batching, it could imply that the prediction model has picked up on the batching mechanism.

We have predictions $\mathcal{P}$ from a bucket $B$ of predictions that we wish to evaluate. We need to select the subset $\mathcal{P}_{\in R} \subseteq \mathcal{P}$ that will be batched and the subset $\mathcal{P}_{\notin R} = \mathcal{P} \setminus \mathcal{P}_{\in R}$ that will not be batched. To detect which cases will be in a batch, we will again use the batch miner from [2] to label each $P \in \mathcal{P}$ after which we create the subsets based on this labeling.

We now have subsets $\mathcal{P}_{\in R} = \{P_{\in R,1}...P_{\in R,l}\}$ and $\mathcal{P}_{\notin R} = \{P_{\notin R,1}...P_{\notin R,m}\}$ with $l + m = n$ that have $l$ predicted and actual remaining times of batched cases $\overline{y}_{\in R,1}...\overline{y}_{\in R,l}$ and $y_{\in R,1}...y_{\in R,l}$ and $m$ predicted and actual remaining times of non-batched cases $\overline{y}_{\notin R,1}...\overline{y}_{\notin R,m}$ and $y_{\notin R,1}...y_{\notin R,m}$. Then, using these four sets of measurements we again create histograms and calculate the *emd*, such that we can evaluate the performance of the prediction model specifically for batched and non-batched cases. Additionally, we calculate the MAE of the remaining time for batched cases $MAE_{\in R} = \sum_{i=1}^{l}(|\overline{y}_{\in R,i} - y_{\in R,i}|)/l$ and non-batched cases $MAE_{\notin R} = \sum_{i=1}^{m}(|\overline{y}_{\notin R,i} - y_{\notin R,i}|)/m$.

Next, following the method from Section 8.1, we derive for $\mathcal{P}_{\in R}$ $l-1$ predicted and actual interdeparture times $\overline{ID}_{\in R,1}...\overline{ID}_{\in R,l-1}$ and $ID_{\in R,1}...ID_{\in R,l-1}$ and derive for $\mathcal{P}_{\notin R}$ $m-1$ predicted and actual interdeparture times $\overline{ID}_{\notin R,1}...\overline{ID}_{\notin R,m-1}$ and $ID_{\notin R,1}...ID_{\notin R,m-1}$. We use these four sets of measurements again to create histograms and calculate the *emd*. Contrary to the remaining time measure, we cannot calculate the $MAE$ of the batches and non-batched interdeparture time, because this measure does not allow for a one-on-one comparison.

Running Example

We take predictions $\mathcal{P}_{4,test}$ from the previous running example in Section 8.2 and we use the batch miner to split these up in subsets $\mathcal{P}_{4,test,\in R} \subseteq \mathcal{P}_{4,test}$ that will be batched and the subset $\mathcal{P}_{4,test,\notin R} = \mathcal{P}_{4,test} \setminus \mathcal{P}_{4,test,\in R}$. From each of these subsets we extract all actual and predicted remaining times and we calculate the actual and predicted interdeparture times following the method described in Section 8.1.

The histograms for the predicted and actual remaining times of batched cases, $\overline{y}_{\in R}$ and $y_{\in R}$, and of non-batched cases, $\overline{y}_{\notin R}$ and $y_{\notin R}$, are depicted in Figure 8.4.

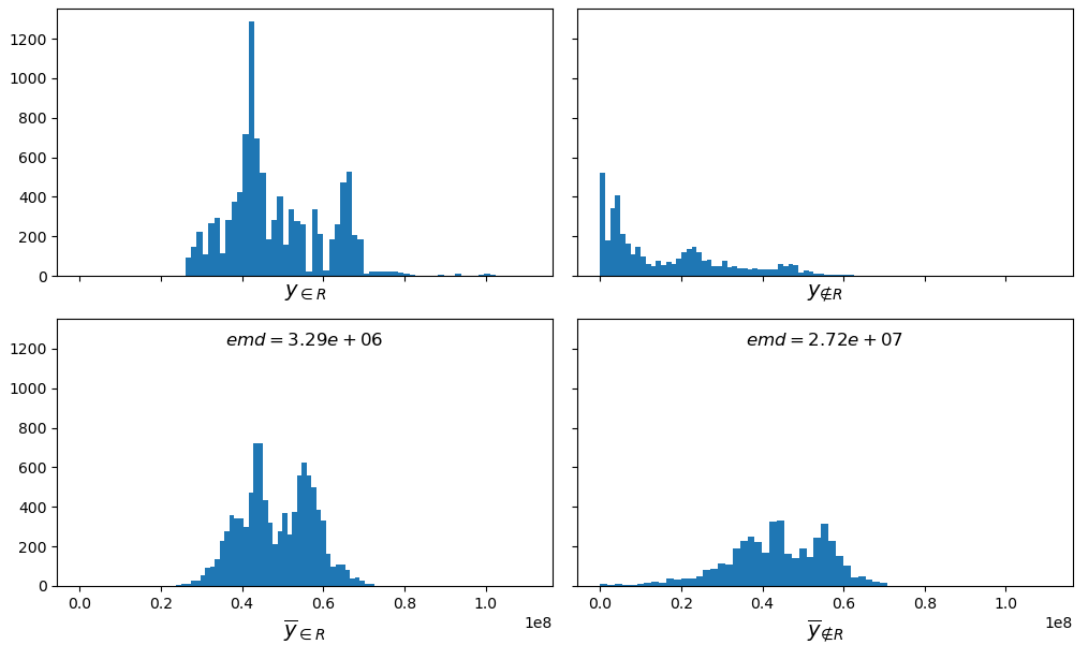

**Figure 8.4:** Histograms of $y_{\in R}$ and $\overline{y}_{\in R}$ of $P_{4,test,\in R}$ and of $y_{\notin R}$ and $\overline{y}_{\notin R}$ of $P_{4,test,\notin R}$ retrieved using model $RM_{p,a,x}$

In Figure 8.4 we can more clearly see where the differences we observed in Figure 8.2 originate from. We see that the distributions of $\overline{y}_{\in R}$ and $y_{\in R}$ are quite similar. They differ slightly, since $y_{\in R}$ shows a higher peak at $0.45 \cdot 10^8$ than $\overline{y}_{\in R}$. Also, $y_{\in R}$ shows a second peak at $0.67 \cdot 10^8$, whereas $\overline{y}_{\in R}$ already shows its second peak at $0.55 \cdot 10^8$.

Contrarily, the distributions of $\overline{y}_{\notin R}$ and $y_{\notin R}$ are completely different. The histogram for $\overline{y}_{\notin R}$ shows a distribution similar to that of $\overline{y}_{\in R}$, whereas the distribution of $y_{\notin R}$ is very much skewed to the left, indicating that the remaining times outside of a batch are on average much shorter. We additionally calculated the $MAE$ of the remaining time for $\mathcal{P}_{4,test,\in R}$ and $\mathcal{P}_{4,test,\notin R}$, which are 149.4 days and 330.5 days, respectively.

When we compare Figure 8.4 to Figure 8.2, we observe that distribution of $\overline{y}$ for model $RM_{p,a,x}$ (Figure 8.2) resembles the distribution of $y_{\in R}$ more than the distribution of $y_{\notin R}$, i.e. it ignores the left tail of $y_{\notin R}$ in Figure 8.4. However, the peaks in $\overline{y}_{\in R}$ are still in the wrong places compared to $y_{\in R}$.

In Figure 8.5 we have depicted the histograms for the predicted and actual interdeparture times of batched cases, $\overline{ID}_{\in R}$ and $ID_{\in R}$, and of non-batched cases, $\overline{ID}_{\notin R}$ and $ID_{\notin R}$.

Similar to $\overline{y}_{\in R}$ and $\overline{y}_{\notin R}$, the distributions of $\overline{ID}_{\in R}$ and $\overline{ID}_{\notin R}$ are also very much alike. We can also spot a clear difference between the distributions of $ID_{\in R}$ and $ID_{\notin R}$, just like we could earlier for $y_{\in R}$ and $y_{\notin R}$. We see that $ID_{\in R}$ is either 0 or $> 9000$, which is a result of all $ID_{\in R}$ within a single batch and the remainder $ID_{\in R}$ between two batch instances, respectively.

Altogether, we discovered that the model $RM_{p,a,x}$ picks up on the high processing times of batched cases and therefore on average predicts high remaining times for all cases. While this has a positive effect on the model's performance for batched cases, it results in overprediction of the non-batched cases, as can also be seen from their MAE of 330.5 days. This has a negative effect on the overall performance of model $RM_{p,a,x}$. In terms of batching behavior, the model has not picked up on the 0 length interdeparture times of batched cases, or the long interdeparture times between batch instances.

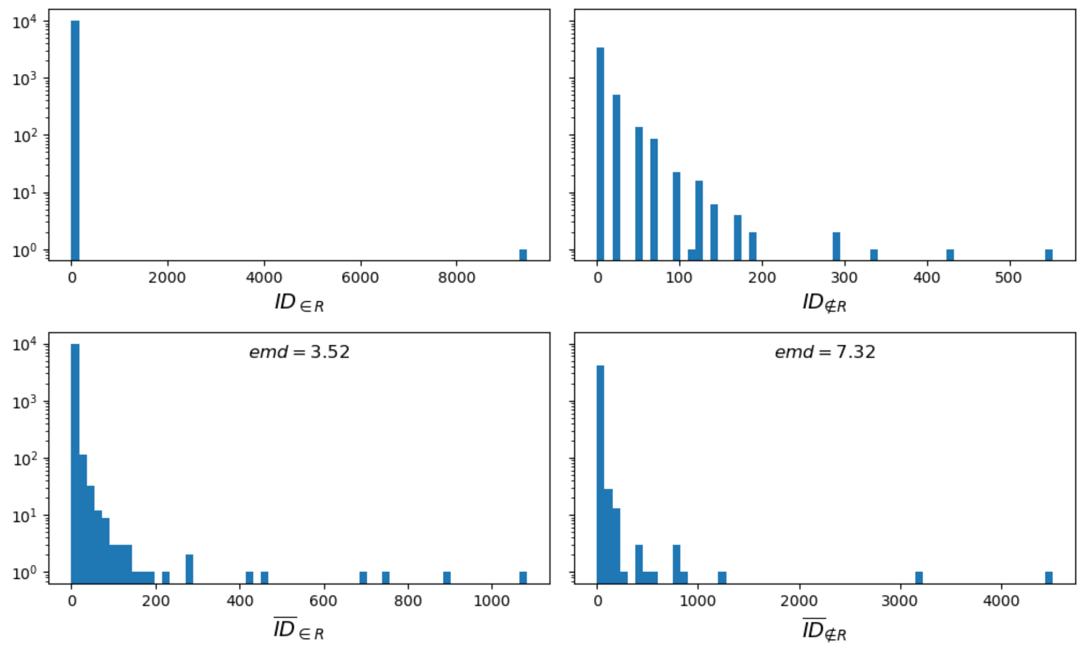

**Figure 8.5:** Histograms of $ID_{\in R}$ and $\overline{ID}_{\in R}$ of $P_{4,test,\in R}$ and of $ID_{\notin R}$ and $\overline{ID}_{\notin R}$ of $P_{4,test,\notin R}$ retrieved using model $RM_{p,a,x}$

For all of the above methods, it is important to keep in mind the fact that these histograms are only meant to compare how outcomes of a prediction model $RM$ are distributed and not compare the actual performance of $RM$. For instance, if two histograms that are compared show overlap, this overlap is only the result of a similar frequency of that measurement, but does not imply a one-on-one relation between the predictions that correspond to those measurements.

# Chapter 9

# Empirical Evaluation

In the previous chapters we presented several methodologies to extend the life cycle of remaining time prediction with. First, in Chapter 6 we introduced how to uncover inter-case mechanisms that cause high prediction errors in a remaining time prediction model $RM$ by performing a fine-grained error analysis. In Chapter 7 we have shown how to use these insights to derive inter-case features specifically for inter-case mechanisms caused by batching and in Chapter 8 we have proposed additional inter-case evaluation steps that better respect these batching mechanisms.

In this chapter we will evaluate the results of these methodologies on our running example (method $(p, a, x)$ and RF log) and on other methods. We have already performed a fine-grained analysis on the prediction results of the model (see running example in Chapter 6) and based on these insights, i.e. presence of batching in the *Send for Credit Collection* activity, we have created several inter-case features from which we can select one as an additional feature as input to method $(p, a, x)$ (see running example in Chapter 7). In Section 9.1 we will assess the quality of each of these features based on the performance of method $(p, a, x)$ with that selected feature using the evaluation steps proposed in Chapter 8. When we have found the best performing inter-case feature, we will test its performance on 17 additional methods in Section 9.2.

For the training and testing of the models for each of the inter-case features and prediction methods, we have again made used of the benchmark implementation available at `https://github.com/verenich/time-prediction-benchmark` [4]. For the creation of the inter-case features, the inter-case assessment and all other tasks we describe in the following, we have created additional scripts in Python which are available at `https://doi.org/10.5281/zenodo.3732446`.

## 9.1 Evaluation of Inter-Case Features for Batching

In the running example of Chapter 7 we have illustrated how to build models $CM_{S,R}$ and $TM$ required to create an inter-case feature $\overline{d}(\overline{c}_R)$ for prefixes of $k = 4$ that capture the batching mechanism $R$ present in the segment $S$=(-,SC). Based on this method we created five inter-case features $\overline{d}$: the predicted time until batching $\overline{t}_R$ and the predicted batch partition $\overline{p}_n$ with $n \in \{4, 8, 10, 20\}$.

The main aim of this section is to find the inter-case feature $\overline{d}(\overline{c}_R)$ that yields the best remaining time prediction performance. Additionally, we also want to evaluate what happens when we have perfect knowledge on whether a prefix will be in a batch, i.e the actual classification $c_R$, or perfect knowledge on the actual time until batching $d = t_R$, or both. Therefore, we first create one additional inter-case feature $d = t_R$, which is the actual time until batching. We subsequently use these six inter-case features (five inter-case features $\overline{d}$ and inter-case feature $d$) to derive five inter-case features $\overline{d}(\overline{c}_R)$ and inter-case feature $d(\overline{c}_R)$ based on the predicted pattern classification and five inter-case features $\overline{d}(c_R)$ and inter-case feature $d(c_R)$ based on the actual pattern classification.

To compare the performance of these inter-case features, we want to view the prediction method $(p, a, x)$ as a black box and create a different *input configuration* $I$ per inter-case feature to use as input to $(p, a, x)$. In 9.1.1 we will evaluate the performance of method $(p, a, x)$ for input configurations $I$ created for the features based on the predicted classification $\overline{c}_R$. In 9.1.2 we will evaluate the performance of method $(p, a, x)$ for different input configurations $I$ created for the inter-case features based on the actual classification $c_R$.

### 9.1.1 Evaluation of Inter-Case Features Based on Predicted Classification

We want to assess the performance of each of the inter-case features we created in the running example of Section 7.4 based on the predicted classification $\overline{c}_R$. For this we used model $CM_{S,R}$ to derive $\overline{c}_R$ for each prefix $hd^4(\sigma) \in L_{train} \cup L_{test}$. Based on this classification $\overline{c}_R$ we derived inter-case features $\overline{d}(\overline{c}_R)$ with $\overline{d} \in \{\bar{t}_P, \overline{p}_4, \overline{p}_8, \overline{p}_{10}, \overline{p}_{20}\}$ and "cheat" inter-case feature $d(\overline{c}_R)$ with $d = t_R$, where $t_R$ is the true time until batching.

We first create baseline configuration $I(0) = X_1...X_n$ without an inter-case feature. Then, for each inter-case feature $\overline{d}(\overline{c}_R)$ we create input configuration $I(\overline{d}(\overline{c}_R)) = X_1...X_n, \overline{d}(\overline{c}_R)$ and for inter-case feature $t_R(\overline{c}_R)$ we create input configuration $I(t_R(\overline{c}_R)) = X_1...X_n, t_R(\overline{c}_R)$.

For using each input configuration $I$, we train a model $RM_{p,a,x,I}$ using prefixes $hd^4(\sigma) \in L_{train}$ to predict label $y$. We subsequently deploy each model $RM_{p,a,x,I}$ to predict the remaining time $\overline{y}$ for configuration $I$ of prefixes $hd^4(\sigma) \in L_{test}$, resulting in $|I|$ sets of predictions $\mathcal{P}_I$. Following the inter-case evaluation from Chapter 8, we use these sets of predictions to calculate for each $I$ the MAE$_y$, $emd_y$ and $emd_{ID}$ for all cases, cases $\in R$ and cases $\notin R$, depicted in Table 9.1. We additionally created histograms of $\overline{y}_{\in R}$, $\overline{y}_{\notin R}$, $\overline{ID}_{\in R}$ and $\overline{ID}_{\notin R}$ for all $I$, depicted in Appendix B.1.1.

**Table 9.1:** Results of configurations $I(0)$, $I(\overline{d}(\overline{c}_R))$ with different inter-case features for $\overline{d}$ and $I(t_R(\overline{c}_R))$ for method $(p, a, x)$

| | MAE$_y$ | MAE$_{y\in R}$ | MAE$_{y\notin R}$ | $emd_y$ | $emd_{y\in R}$ | $emd_{y\notin R}$ | $emd_{ID}$ | $emd_{ID\in R}$ | $emd_{ID\notin R}$ |
|---|---|---|---|---|---|---|---|---|---|
| $I(0)$ | 202.92 | 149.37 | 330.46 | $9.99 \cdot 10^6$ | $3.29 \cdot 10^6$ | $2.72 \cdot 10^7$ | 3.62 | 3.52 | 7.32 |
| $I(\bar{t}_R(\overline{c}_R))$ | 236.32 | 183.07 | 363.15 | $1.14 \cdot 10^7$ | $4.31 \cdot 10^6$ | $2.97 \cdot 10^7$ | 4.19 | 3.49 | 7.36 |
| $I(\overline{p}_4(\overline{c}_R))$ | 219.60 | 165.32 | 348.89 | $1.02 \cdot 10^7$ | $3.65 \cdot 10^6$ | $2.85 \cdot 10^7$ | 3.78 | **3.19** | 6.74 |
| $I(\overline{p}_8(\overline{c}_R))$ | 184.19 | 120.70 | 335.43 | $9.56 \cdot 10^6$ | $3.40 \cdot 10^6$ | $2.76 \cdot 10^7$ | 3.96 | 3.71 | 5.98 |
| $I(\overline{p}_{10}(\overline{c}_R))$ | **180.06** | **117.64** | 330.93 | **9.07** $\cdot 10^6$ | $3.91 \cdot 10^6$ | **2.72** $\cdot 10^7$ | **3.64** | 3.56 | **5.42** |
| $I(\overline{p}_{20}(\overline{c}_R))$ | 181.38 | 118.85 | **330.32** | $9.42 \cdot 10^6$ | **3.29** $\cdot 10^6$ | **2.72** $\cdot 10^7$ | 4.13 | 3.61 | 6.91 |
| $I(t_R(\overline{c}_R))$ | 91.18 | 35.97 | 222.69 | $5.04 \cdot 10^6$ | $2.23 \cdot 10^6$ | $1.77 \cdot 10^7$ | 3.71 | 3.47 | 6.77 |

In Table 9.1 we see that both $I(\bar{t}_R(\overline{c}_R))$ and $I(\overline{p}_4(\overline{c}_R))$ do not improve upon $I(0)$ at all. In Figures B.1 and B.2 we see that $\overline{y}_{\in R}$ and $\overline{y}_{\notin R}$ are both significantly more overpredicted than the baseline $I(0)$, which also explains that all MAE$_y$ and $emd_y$ values are higher for both of these configurations. However, the $emd_{ID_{\in R}}$ of $I(\overline{p}_4(\overline{c}_R))$ is the lowest of all configurations, which we can also detect in the corresponding histogram in Figure B.3, where it somehow detected a large batch interval at around $\overline{ID}_{\in R} = 8500$, similar to the actual outcomes.

In the results of Table 9.1, we can also observe that $I(\overline{p}_8(\overline{c}_R))$, $I(\overline{p}_{10}(\overline{c}_R))$ and $I(\overline{p}_{20}(\overline{c}_R))$ yield comparable performance. In terms of the $\text{MAE}_y$, $\text{MAE}_{y_{\in R}}$, $\text{MAE}_{y_{\notin R}}$, all of these configurations improve upon $I(0)$. We can see in Figures B.1 and B.2 that, contrary to $I(\overline{t}_R(\overline{c}_R))$ and $I(\overline{p}_4(\overline{c}_R))$, the peaks in configurations $I(\overline{p}_8(\overline{c}_R))$, $I(\overline{p}_{10}(\overline{c}_R))$ and $I(\overline{p}_{20}(\overline{c}_R))$ are a bit more shifted towards the left as is also the case for the actual outcomes. As for the interdeparture times, we can derive from Table 9.1 that $I(\overline{p}_8(\overline{c}_R))$, $I(\overline{p}_{10}(\overline{c}_R))$ and $I(\overline{p}_{20}(\overline{c}_R))$ are all not able to improve upon $I(0)$ for $emd_{ID}$ and $emd_{ID_{\in R}}$, which is also reflected in Figure B.3. However, these configurations do perform better than $I(0)$ for $emd_{ID_{\notin R}}$, as can also be seen in Figure B.4, where they are better able to detect the shorter interdeparture times that emerge outside of a batch.

When we again consider Table 9.1, we see that $I(\overline{p}_{10}(\overline{c}_R))$ has the best overall performance of the configurations with the predicted inter-case features. We want to compare this performance with our cheat feature $I(t_R(\overline{c}_R))$ and therefore we additionally depicted the results of $\overline{y}_{\in R}$, $\overline{y}_{\notin R}$, $\overline{ID}_{\in R}$ and $\overline{ID}_{\notin R}$ for $I(0)$, $I(\overline{p}_{10}(\overline{c}_R))$ and $I(t_R(\overline{c}_R))$ in Figures 9.1 to 9.4.

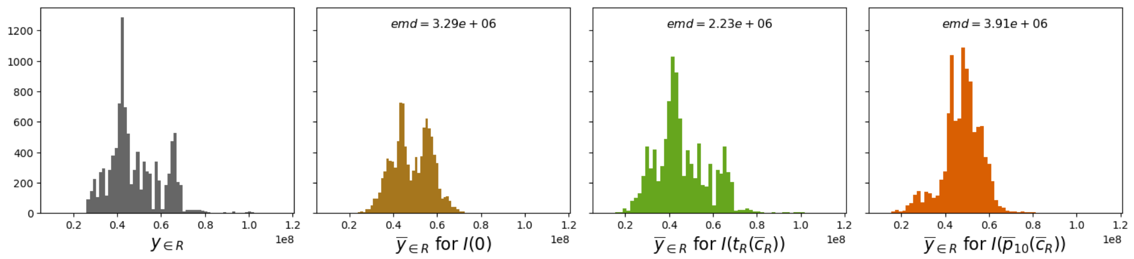

**Figure 9.1:** Histograms of actual $y_{\in R}$ and predicted $\overline{y}_{\in R}$ for $I(0)$, $I(t_R(\overline{c}_R))$ and $I(\overline{p}_{10}(\overline{c}_R))$

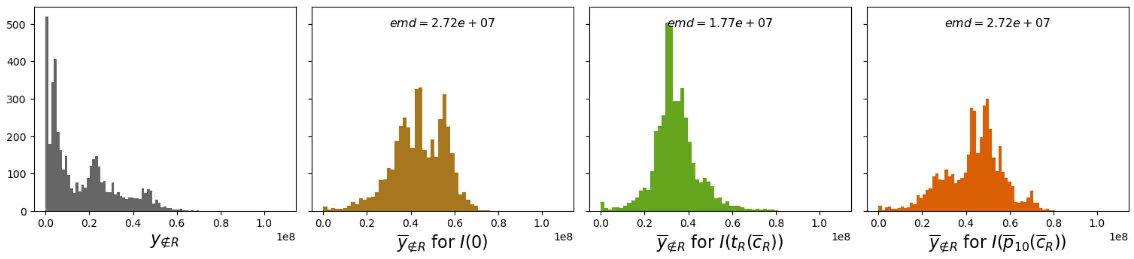

**Figure 9.2:** Histograms of actual $y_{\notin R}$ and predicted $\overline{y}_{\notin R}$ for $I(0)$, $I(t_R(\overline{c}_R))$ and $I(\overline{p}_{10}(\overline{c}_R))$

In Table 9.1 we see for both the MAE and $emd$ that $I(t_R(\overline{c}_R))$ is able to capture $\overline{y}$, $\overline{y}_{\in R}$ and $\overline{y}_{\notin R}$ quite nicely, which is also reflected in the histograms in Figures 9.1 and 9.2. While the distribution of $\overline{y}_{\in R}$ for $I(t_R(\overline{c}_R))$ closely resembles the actual distribution of $y_{\in R}$, the distribution of $\overline{y}_{\notin R}$ still needs some shifting to the left for it to resemble the actual distribution of $y_{\notin R}$. When we compare $I(0)$, $I(\overline{p}_{10}(\overline{c}_R))$ and $I(t_R(\overline{c}_R))$ in Figures 9.1 and 9.2, we see that for both $y_{\in R}$ and $y_{\notin R}$, the distributions of $I(\overline{p}_{10}(\overline{c}_R))$ seem to lie somewhere in between those of $I(0)$ and $I(t_R(\overline{c}_R))$. However, this is not reflected by the $emd_{y_{\in R}}$ and $emd_{y_{\notin R}}$.

Similar to the other configurations, Figure 9.3 shows that $I(\overline{p}_{10}(\overline{c}_R))$ and $I(t_R(\overline{c}_R))$ both have distributions of $ID_{\in R}$ that less resemble the actual distribution than the distribution of $I(0)$. However, also similar to other configurations, Figure 9.4 shows that the distributions of $ID_{\notin R}$ for $I(\overline{p}_{10}(\overline{c}_R))$ and $I(t_R(\overline{c}_R))$ better resemble the actual distribution than the distribution of the baseline, as is also reflected by the $emd_{ID_{\notin R}}$.

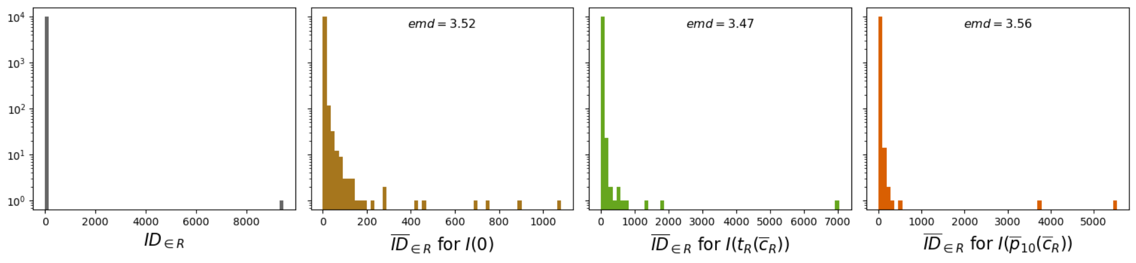

**Figure 9.3:** Histograms of actual $ID_{\in R}$ and predicted $\overline{ID}_{\in R}$ for $I(0)$, $I(t_R(\overline{c}_R))$ and $I(\overline{p}_{10}(\overline{c}_R))$

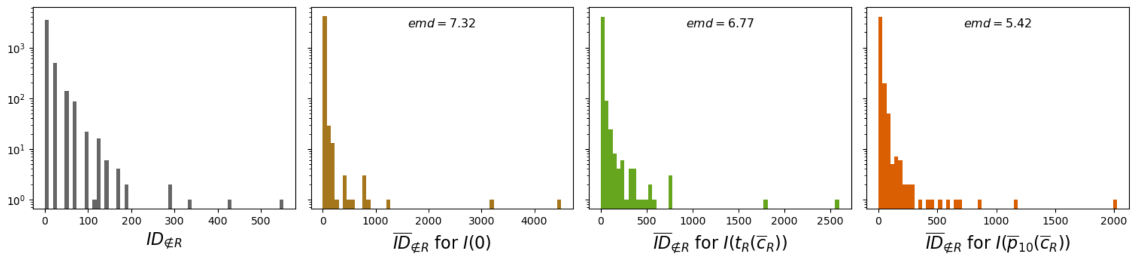

**Figure 9.4:** Histograms of actual $ID_{\notin R}$ and predicted $\overline{ID}_{\notin R}$ for $I(0)$, $I(t_R(\overline{c}_R))$ and $I(\overline{p}_{10}(\overline{c}_R))$

### 9.1.2 Evaluation of Inter-Case Features Based on Actual Classification

In the previous section we assessed the performance of each of the inter-case features we created in the running example of Section 7.4, based on their predicted classification $\overline{c}_R$. In this section we want to make this assessment based on the actual classification $c_R$. This way, we can evaluate the influence of our predicted classification on the performance of the inter-case features $\overline{d}$ and inter-case feature $d$.

For this we use the actual classification for each prefix $hd^4(\sigma) \in L_{train} \cup L_{test}$. We again derive inter-case features, this time using $c_R$, resulting in inter-case features $\overline{d}(c_R)$ with $\overline{d} \in \{\overline{t}_P, \overline{p}_4, \overline{p}_8, \overline{p}_{10}, \overline{p}_{20}\}$ and "cheat" inter-case feature $d(c_R)$ with $d = t_R$, where $t_R$ is the true time until batching.

For each inter-case feature $\overline{d}(c_R)$ we create input configuration $I(\overline{d}(c_R) = X_1...X_n, \overline{d}(c_R)$ and for inter-case feature $t_R(c_R)$ we create input configuration $I(t_R(c_R)) = X_1...X_n, t_R(c_R)$.

For each input configuration $I$, we train a model $RM_{p,a,x,I}$ using configuration $I$ of the prefixes $hd^4(\sigma) \in L_{train}$ to predict label $y$. We subsequently deploy each model $RM_{p,a,x,I}$ to predict the remaining time $\overline{y}$ for configuration $I$ of prefixes $hd^4(\sigma) \in L_{test}$, resulting in $|I|$ sets of predictions $\mathcal{P}_I$. For each $\mathcal{P}_I$ we calculate the $\text{MAE}_y$, $emd_y$ and $emd_{ID}$ for all cases, cases $\in R$ and cases $\notin R$, depicted in Table 9.1. We additionally created histograms of $\overline{y}_{\in R}$, $\overline{y}_{\notin R}$, $\overline{ID}_{\in R}$ and $\overline{ID}_{\notin R}$ for all $I$, depicted in Appendix B.1.2.

What first strikes is that the performance for $y$ of almost all $I(c_R)$ in Table 9.2 is significantly better than the performance of their counterparts $I(\overline{c}_R)$ depicted in Table 9.1. This is of course no surprise, since for this assessment we gave our configurations the actual information on whether prefixes would be part of a pattern or not. For the best performing configuration of Table 9.1, i.e. $I(\overline{p}_{10}(\overline{c}_R))$, we see that its corresponding configuration based on the actual pattern classification $I(\overline{p}_{10}(c_R))$ yields an error decrease of almost 40 days for the $\text{MAE}_y$. We see that this is mostly due to the error decrease of almost 125 days for the $\text{MAE}_{y_{\notin R}}$, as opposed to the slight error decrease of only 4 days for the $\text{MAE}_{y_{\in R}}$. We see similar error decreases of $\text{MAE}_y$, $\text{MAE}_{y_{\in R}}$ and $\text{MAE}_{y_{\notin R}}$ for the other configurations $I(c_R)$ compared to their counterparts $I(\overline{c}_R)$. However, for the $emd_{ID}$,

**Table 9.2:** Results of configurations $I(0)$, $I(\overline{d}(c_R))$ with different inter-case features for $\overline{d}$ and $I(t_R(c_R))$ for method $(p, a, x)$

| | MAE$_y$ | MAE$_{y \in R}$ | MAE$_{y \notin R}$ | $emd_y$ | $emd_{y \in R}$ | $emd_{y \notin R}$ | $emd_{ID}$ | $emd_{ID_{\in R}}$ | $emd_{ID_{\notin R}}$ |
|---|---|---|---|---|---|---|---|---|---|
| $I(\overline{p}_{10}(\overline{c}_R))$ | **180.06** | **117.64** | **330.93** | **9.07** $\cdot 10^6$ | **3.91** $\cdot 10^6$ | **2.72** $\cdot 10^7$ | **3.64** | **3.56** | **5.42** |
| $I(\overline{t}_R(c_R))$ | 178.64 | 166.53 | 207.49 | 8.14 $\cdot 10^6$ | 6.64 $\cdot 10^6$ | 1.31 $\cdot 10^7$ | 3.81 | 3.02 | 7.34 |
| $I(\overline{p}_4(c_R))$ | 180.08 | 167.26 | 210.60 | 7.41 $\cdot 10^6$ | 5.48 $\cdot 10^6$ | 1.34 $\cdot 10^7$ | 4.14 | 3.73 | 7.01 |
| $I(\overline{p}_8(c_R))$ | 142.10 | 114.30 | 208.32 | 6.74 $\cdot 10^6$ | 4.61 $\cdot 10^6$ | 1.33 $\cdot 10^7$ | 4.07 | 3.65 | 6.80 |
| $I(\overline{p}_{10}(c_R))$ | 140.88 | 113.35 | 206.45 | 6.96 $\cdot 10^6$ | 4.95 $\cdot 10^6$ | 1.32 $\cdot 10^7$ | 3.88 | 3.66 | 6.26 |
| $I(\overline{p}_{20}(c_R))$ | 140.38 | 111.57 | 209.01 | 6.67 $\cdot 10^6$ | 4.49 $\cdot 10^6$ | 1.33 $\cdot 10^7$ | 3.88 | 3.35 | 7.08 |
| $I(t_R(c_R))$ | 67.8 | 11.78 | 201.36 | 3.77 $\cdot 10^6$ | 4.01 $\cdot 10^5$ | 1.26 $\cdot 10^7$ | 4.38 | 3.30 | 9.04 |

$emd_{ID_{\in R}}$ and $emd_{ID_{\notin R}}$, we see a decrease in performance for almost all of the configurations, of which the most significant decrease is that of $emd_{ID_{\notin R}}$. Even more surprising, the largest performance decrease of $emd_{ID_{\notin R}}$ is for configuration $I(t_r(c_R))$, which is actually the most precise inter-case feature we have used, i.e. the actual time until batching based on the actual pattern classification.

What we secondly observe is that the performance differences among configurations $I(\overline{c}_R)$ (see Table 9.1) are quite similar to the performance differences among configurations $I(c_R)$ that we see in Table 9.2. We again see that $I(\overline{t}_R(c_R))$ and $I(\overline{p}_4(c_R))$ are the worst performing, as is also similarly reflected in Figures B.5 and B.6. But also in this assessment, one of these two configurations (this time $I(\overline{t}_R)(c_R))$ performs best in terms of $emd_{ID_{\in R}}$ (in 9.1.1 this was configuration $I(\overline{p}_4)(\overline{c}_R))$.

In Table 9.1 we can also spot similarities for $I(\overline{p}_8(c_R))$, $I(\overline{p}_{10}(c_R))$ and $I(\overline{p}_{20}(c_R))$, among which the performance is quite similar, just as in Table 9.1. We also see these similarities for the distributions of remaining times in Figures B.5 and B.6 as well as for the distributions of the interdeparture times in Figures B.7 and B.8. Again here, $I(\overline{p}_{10}(c_R))$ and $I(\overline{p}_{20}(c_R))$ are the best performing (of the configurations that used a prediction $\overline{d}$). Since the configuration with $\overline{p}_{10}$ was also the best performing of those derived from $\overline{c}_R$, we will here stick with $I(\overline{p}_{10}(c_R))$ as the best performing configuration.

We additionally depicted the results of $\overline{y}_{\in R}$, $\overline{y}_{\notin R}$ for the baseline $I(0)$, $I(\overline{p}_{10}(c_R))$ and the configuration with the cheat inter-case feature $I(t_r(c_R))$ in Figures 9.5 and 9.6. Interestingly, the distributions of $\overline{y}_{\in R}$ for the configurations based on the actual classifications $\overline{c}_R$ resemble the actual outcomes less than the distributions of $\overline{y}_{\in R}$ for the configurations based on the predicted classifications $c_R$, which is also reflected by the higher $emd_{y_{\in R}}$. However, the distributions of $\overline{y}_{\notin R}$ did shift more towards the actual distribution, as is also reflected by the lower $emd_{y_{\notin R}}$.

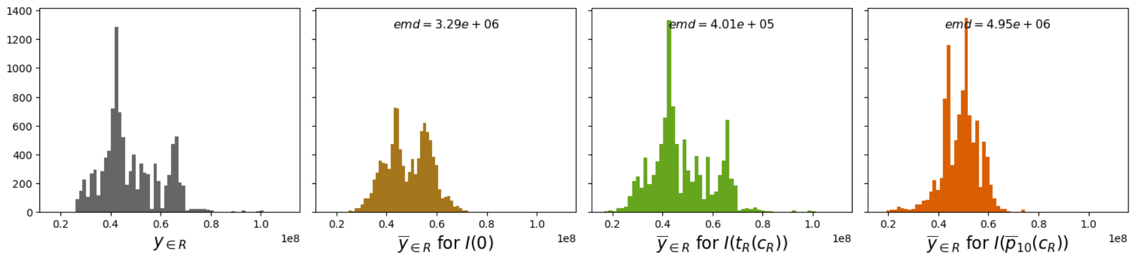

**Figure 9.5:** Histograms of actual $y_{\in R}$ and predicted $\overline{y}_{\in R}$ for $I(0)$, $I(t_R(c_R))$ and $I(\overline{p}_{10}(c_R))$

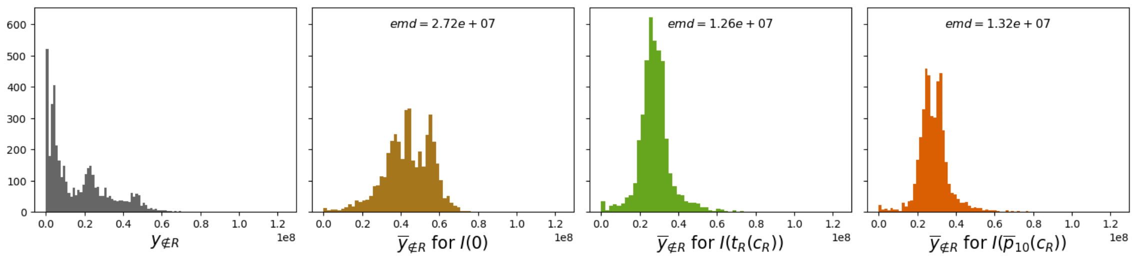

**Figure 9.6:** Histograms of actual $y_{\notin R}$ and predicted $\overline{y}_{\notin R}$ for $I(0)$, $I(t_R(c_R))$ and $I(\overline{p}_{10}(c_R))$

## 9.1.3 General Observations

In the previous subsections we have contrasted the performance of different input configurations $I$ using a variety of evaluation measurements and histograms. These input configurations in their turn included a variety of inter-case features that allowed us to assess the performance of these inter-case features on a remaining time prediction method $(p, a, x)$. Based on this assessment, we first discuss some general observations regarding the accuracy of the inter-case features itself and thereby also the influence of the models $CM_{S,R}$ and $TM$ on the accuracy of these features. Second, we discuss the inter-case features created using both model $CM_{S,R}$ and $TM$ and motivate the choice for the final inter-case feature we will use in the next section. For completeness, we additionally created an OPS and OPS' that correspond to the results of each created feature, which can be found in Appendix B.2 and Figures 9.7 to 9.10.

### Feature accuracy

In Section 9.1 we have generally observed that if we provide the remaining time prediction model with more accurate information on the batching mechanism, i.e. information on whether a prefix will be part of a batch or how long the prefix will be in a batch, the performance of the model increases. This makes sense, because predicting something based on something that is actually incorrect will most likely also lead to incorrect outcomes. We see this for our cheat features based on the actual classifications $c_R$, for our cheat features with the actual time until batching $t_R$ and even more so for our feature $t_R(c_R)$ that has both.

Furthermore, we have observed that if we give the model more (accurate) information on the time until batching, the model's performance predominantly increases for batched cases. We first observe this in Table 9.1, where the $\text{MAE}_{y_{\in R}}$ decreases when we add predictions of the time until batching/batch partition, while the $\text{MAE}_{y_{\notin R}}$ stays the same (or increases slightly). We also observe this in Table 9.2, where the $\text{MAE}_{y_{\in R}}$ decreases significantly when we use the actual time until batching $t_R$, while the $\text{MAE}_{y_{\notin R}}$ only decreases slightly.

Conversely, when we give the model more certainty on whether a case is in a pattern of not, i.e. we give the actual classification $c_R$, we observe a significant increase in performance for the non-batched cases. Specifically, when we compare the $\text{MAE}_{y_{\notin R}}$ of configurations $I(\overline{c}_R)$ from Table 9.1 to those of classifications $I(c_R)$ from Table 9.2, we observe a decrease of roughly 120 days on average. Additionally, when we compare the $emd_{y_{\notin R}}$ from Table 9.1 to the corresponding $emd_{y_{\notin R}}$ from Table 9.2 we also observe a significant decrease.

All of the above concerns results of configurations that carry information that we in reality never have up front, but it does give us a perspective on what is the ideal situation and what more can still be gained. Therefore, we want to contrast the results of the best performing non-cheat inter-case feature $\overline{p}_{10}(\overline{c}_R)$ with more "ideal" results, e.g. $\overline{p}_{10}(c_R)$, $t_R(\overline{c}_R)$ and $t_R(c_R)$. The OPS' for the results of these configurations are depicted in Figures 9.7 to 9.10.

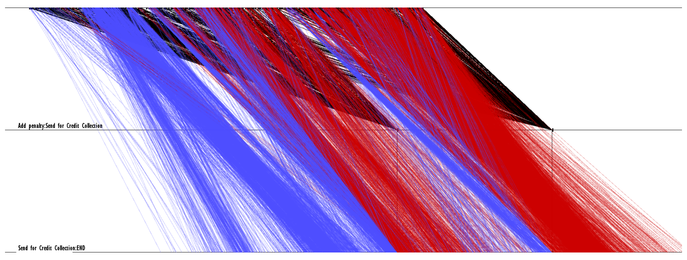

**Figure 9.7:** OPS' of predictions $I(\overline{p}_{10}(\overline{c}_R))$ for segment *Add Penalty:Send for Credit Collection* for $k = 4$

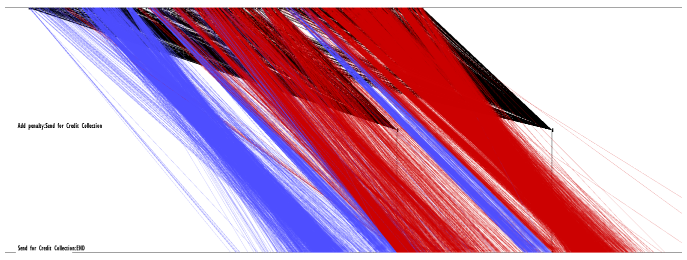

**Figure 9.8:** OPS' of predictions $I(\overline{p}_{10}(c_R))$ for segment *Add Penalty:Send for Credit Collection* for $k = 4$

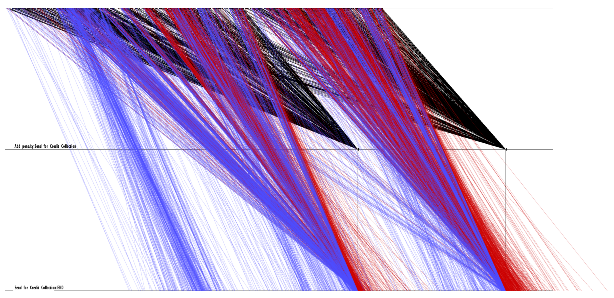

**Figure 9.9:** OPS' of predictions $I(t_R(\overline{c}_R))$ for segment *Add Penalty:Send for Credit Collection* for $k = 4$

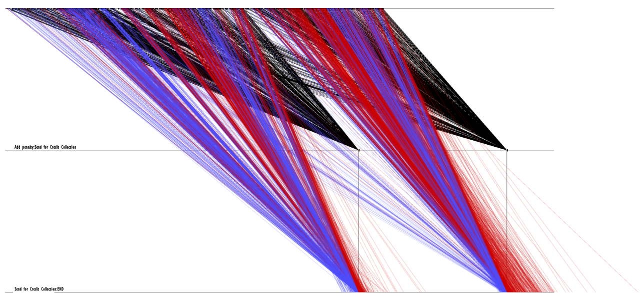

**Figure 9.10:** OPS' of predictions $I(t_R)(c_R)$ for segment *Add Penalty:Send for Credit Collection* for $k = 4$

Unfortunately, from the predictions in the OPS' in Figure 9.7 we observe that the model for configuration $I(\overline{p}_{10}(\overline{c}_R))$ was still not able to pick up on the batching pattern, similar to our baseline configuration (see Figure 6.12). While Figure 9.7 does show predictions going faster than others (as is also the case with batching), these do not coincide with the actual outcomes, nor do they converge in a similar way. When we consider the result of the corresponding configuration based on the actual classification $I(\overline{p}_{10}(c_R))$ in Figure 9.8, we observe that the predictions converge a bit more, but they still do not form a batching pattern. However, the predictions of our configurations with the actual time until batching $t_R$ in Figures 9.9 and 9.10 do form clear batching patterns. In Figure 9.9 we can see that some predictions still not form a batching pattern, which is most likely a result of the use of the predicted classification $\overline{c}_R$. However, in Figure 9.9 we do see that the predictions that do not form a batch are predicted to go faster than the predictions in Figures 9.7 and 9.8 that also do not form a batch, meaning that the model with configuration $I(t_R(\overline{c}_R))$, is already better able to distinguish the differences in remaining time for batched and non-batched cases. In Figure 9.10 we see the results of our ideal input configuration using the actual time until batching $t_R$ and the actual classification $c_R$, of which the predictions form a clear batching pattern that closely resembles the actual batching pattern.

**Prediction based inter-case features**

We first found that the inter-case feature $\overline{t}_R$, i.e. the prediction of the remaining time until batching, is the worst performing feature we created (both using $c_R$ and $\overline{c}_R$). We think this is because it is a prediction in time units, which takes a lot of different values in a very large range, and is therefore hard to interpret for a machine learning algorithm. However, the cheat feature $t_R$ with the actual time until batching, which also takes values in time units, did yield good performance results. Based on this we assume that the combination of a value in time units and the fact that these values are predictions themselves, results in a decrease in performance.

Contrarily, the machine learning algorithm seems to react quite well to the inter-case features based on batch partitions $\overline{p}_n$. Where the performance for $\overline{p}_4$ was almost as bad as that for $\overline{t}_R$, the performance improved when we used more partitions, i.e. larger $n$. However, when we used $n = 20$ the performance started to decrease slightly. Based on this we assume that we need a certain level of abstraction, i.e. not too many partitions, but a too low level of abstraction, such as $n = 4$, will not provide the model with the information necessary for predicting the remaining process time.

While some features clearly led to better performance than others, we still do not see a lot of difference when we compare the OPS and OPS' visualization in Figures B.9 to B.18 and we most definitely do not see the predictions forming a batching pattern in any of these visualizations. When we compare the created features $\overline{d}$ based on the actual classification $c_R$ in Figures B.21 to B.30, we do see that the predictions of the inter-case features $\overline{d} = \overline{p}_n$ with higher $n$ already start to converge a bit more like a batching pattern. While the predictions still do not resemble a batching pattern, the configuration $I(\overline{p}_{10})$ yielded the best performance out of the configurations created using both model $CM_{S,R}$ and $TM$ and we will therefore use this configuration to evaluate its performance on other remaining time prediction methods.

## 9.2 Evaluation of Inter-Case Feature on Additional Remaining Time Prediction Methods

In Section 9.1 we have evaluated the performance of different inter-case features on remaining time prediction method $(p, a, x)$. Based on this evaluation, we have selected configuration $I(\overline{p}_{10}(\overline{c}_R))$, which consists out of features $X_1 ... X_n, \overline{p}_{10}(\overline{c}_R)$ where $\overline{p}_{10}(\overline{c}_R)$ is the inter-case feature created based on a predicted classification $\overline{c}_R$ using model $CM_{S,R}$ and a predicted batch partition $\overline{p}_{10}$ using model $TM$. This configuration yielded an overall performance increase of 22.86 days for the MAE$_y$.

In this section we want to evaluate how this configuration performs on other remaining time prediction methods. For this we will again use the benchmark implementation from [4], which consists of different bucketing methods, encoding methods and machine learning (ML) algorithms. We have selected the cluster-, prefix- and single bucketing methods, the aggregate-, index- and last state encoding methods (see Section 3.5) and the extreme gradient boosting and random forest ML algorithms (see Section 3.6). By using combinations of each of these bucketing methods $B$, encoding methods $E$ and ML algorithms $A$, we derived 17 additional remaining time prediction methods $(B, E, A)$ (we already used the method with prefix bucketing $p$, aggregate encoding $a$ and extreme gradient boosting $x$, i.e. $(p, a, x)$, in our running example and Section 9.1).

We want to evaluate the performance of configuration $I(\overline{p}_{10}(\overline{c}_R))$ on each of these methods $(B, E, A)$ by comparing it to the performance of our baseline $I(0)$. For each input configuration $I \in \{I(0), I(\overline{p}_{10}(\overline{c}_R))\}$ and method $(B, E, A)$, we train remaining time prediction model $RM_{B,E,A,I}$ using configuration $I$ of the prefixes $hd^4(\sigma) \in L_{train}$ to predict label $y$. We subsequently deploy each model $RM_{B,E,A,I}$ to predict the remaining time $\overline{y}$ for configuration $I$ of prefixes $hd^4(\sigma) \in L_{test}$, resulting in 2 sets of predictions $\mathcal{P}_I$ for each method $(B, E, A)$. For all these sets of predictions we have calculated the overall MAE, depicted in Table 9.3.

**Table 9.3:** MAE of remaining time prediction methods $(B, E, A)$ using baseline configuration $I(0)$ and inter-case configuration $I(\overline{p}_{10}(\overline{c}_R))$ for $k = 4$

| Method | Bucketing | Encoding | ML Algorithm | $I(0)$ | $I(\overline{p}_{10})$ |
|---|---|---|---|---|---|
| $(c, a, x)$ | Cluster | Aggregate | Xgboost | 196.50 | 182.07 |
| $(c, i, x)$ | Cluster | Index | Xgboost | 202.64 | 186.35 |
| $(c, l, x)$ | Cluster | Laststate | Xgboost | 191.77 | 183.27 |
| $(p, a, x)$ | Prefix | Aggregate | Xgboost | 202.92 | 180.06 |
| $(p, i, x)$ | Prefix | Index | Xgboost | 197.90 | 192.84 |
| $(p, l, x)$ | Prefix | Laststate | Xgboost | 200.27 | 183.14 |
| $(s, a, x)$ | Single | Aggregate | Xgboost | 200.26 | 197.95 |
| $(s, i, x)$ | Single | Index | Xgboost | 199.40 | 208.45 |
| $(s, l, x)$ | Single | Laststate | Xgboost | 186.36 | 192.12 |
| $(c, a, r)$ | Cluster | Aggregate | Random forest | 200.73 | 179.74 |
| $(c, i, r)$ | Cluster | Index | Random forest | 202.15 | 202.19 |
| $(c, l, r)$ | Cluster | Laststate | Random forest | 226.38 | 199.98 |
| $(p, a, r)$ | Prefix | Aggregate | Random forest | 201.89 | 180.71 |
| $(p, i, r)$ | Prefix | Index | Random forest | 204.96 | 200.77 |
| $(p, l, r)$ | Prefix | Laststate | Random forest | 225.47 | 203.86 |
| $(s, a, r)$ | Single | Aggregate | Random forest | 201.87 | 195.53 |
| $(s, i, r)$ | Single | Index | Random forest | 206.04 | 211.90 |
| $(s, l, r)$ | Single | Laststate | Random forest | 230.25 | 203.04 |

Our general observation from Table 9.3 is that the inter-case configuration leads to an increase in performance for almost all methods $(B, E, A)$. This increase in performance varies from a somewhat insignificant MAE decrease of only 2.31 days for method $(s, a, x)$ to a MAE decrease of 27.21 days for method $(s, l, r)$.

We observe that methods including cluster bucketing or prefix bucketing all lead to increases in performance when using the inter-case configuration. However, methods including single bucketing show varying results. While $(s, l, r)$ shows the most significant MAE decrease for all methods, $(s, i, x)$, $(s, l, x)$ and $(s, i, r)$ show an increase of the MAE varying from roughly 5 to 9 days. Apparently, the performance of a method where prefixes are partitioned into a single bucket highly depends on the encoding method or ML algorithm. In terms of encoding methods, we observe the most significant performance increases when aggregate or last state encoding is used, as is the case for methods $(p, a, x)$, $(c, a, r)$, $(c, l, r)$, $(p, a, r)$, $(p, l, r)$ and $(s, l, r)$, where we see at least a decrease of 20 days for the MAE. Of these two encoding methods, aggregate encoding exclusively leads to error decreases, while last state encoding in some cases results in some error increases. As for index encoding, only the method $(c, i, x)$ shows a significant performance increase when using the inter-case configuration, while the other methods with index encoding show similar performance or decreases in performance.

When considering the bucketing and encoding methods together, we observe that the combinations $(c, a, A)$, $(c, l, A)$, $(p, a, A)$, $(p, l, A)$ all lead to significant performance increases, while the combination $(s, i, A)$ always results in a performance decrease. We think that the partitioning of prefixes into buckets with other similar prefixes (either based on prefix-length or based on a cluster), already gives the bucket of prefixes some structure that later positively influences the effect of the inter-case feature. For instance in the current situation, where we have created a feature for prefixes of length $k = 4$ and we use prefix-length buckets, we already have filtered out a lot of prefixes for which we do not need this inter-case feature, making it better interpretable for the ML algorithm. In the case of using a single bucket, the ML algorithm receives prefixes of varying lengths and with varying characteristics. If some of these prefixes have inter-case features and some have not, the ML algorithm will have a much harder time making a distinction than it would in the case where irrelevant prefixes have already been filtered out.

When we consider the two ML algorithms, we observe that extreme gradient boosting in general performs better than random forest. When we compare the performance for $I(0)$ to that for $I(\overline{p}_{10}(\overline{c}_R))$, we see that performance increases are on average higher when using random forest. Apart from that, we see similar increases/decreases in performance among the two ML algorithms depending on the bucketing and encoding methods. We think this is not surprising, since random forest and extreme gradient boosting are both tree ensemble methods. While they still differ in the way the trees are built, we do not think that this difference in structure significantly influences the interpretation of the inter-case feature.

Chapter 10

# Conclusions

In this thesis, we have analyzed the state of the art in remaining time prediction along the phases of the CRISP-DM life cycle and observed that this life cycle is adopted under the wrong assumption that cases progress in isolation. We have shown how this intra-case perspective results in inadequate models and inadequate predictions. Based on this, we addressed the need for the awareness of inter-case dynamics in the life cycle of remaining time prediction.

We have made an inventory of the steps necessary to increase awareness of inter-case dynamics in this life cycle. Based on this we first presented methods for an inter-case error analysis that together enable a trigger of an inter-case dynamic aware feedback loop. We subsequently presented methods that transfer these inter-case insights into the main life cycle, specifically aimed at inter-case dynamics caused by batching. For this we introduced methods for inter-case feature creation for batching and an inter-case evaluation for batching.

We have applied these methods to a primary remaining time prediction approach and were able to identify subsets of cases subject to inter-case dynamics caused by batching that led to an inadequate model and inadequate predictions. Based on these subsets and their characteristics we were able to create inter-case features and an inter-case evaluation that better respect these inter-case dynamics caused by batching. We evaluated the performance of the different inter-case features using the inter-case evaluation and were able to better explain the influence of inter-case dynamics on the prediction outcomes and, using the best performing inter-case feature, were able to decrease the MAE of the primary remaining time prediction method by 22.86 days. We finally evaluated the performance of this best performing inter-case feature on 17 other remaining time prediction methods and were able to decrease the MAE for 14 of these methods and decrease the MAE with more than 15 days for 8 of these. Based on this we consider our main and sub-research questions answered and our research objective fulfilled.

Despite the previous, we still want to address the following. In this thesis, we have come up with methods to increase inter-case awareness in case-centered prediction models that use case-centered features and output case specific prediction results. This feels a bit contradicting, given that inter-case dynamics are not case specific, but they actually apply to all cases together. Because of this, we think it is necessary to consider whether it is in some situations, e.g. certain types of processes or certain types of organizations, even logical to try to predict case-specific outcomes. This is of course an entirely new research problem in itself, but it does give us some food for thought on the way remaining time prediction - and PPM in general - is currently approached.

## 10.1 Limitations

The first limitation is one of validity. While all data sets and method implementations used in this thesis are available and all results and visualizations are thus reproducible, the interpretations of those might not be. This concerns the OPS visualizations that have up until now only been analyzed in this research and therefore the interpretation in this thesis is currently the only one. While this process can still be carried out quite systematically, these inter-case patterns still need to be deducted visually, leading to a possible different interpretation if this process were to be carried out by someone else.

The second limitation concerns the earliness of the predictions we make. Since one of the main goals of predictive process monitoring concerns the *timely* identification of errors, bottlenecks and other deviations, the aim of most methods is to improve prediction performance early on, i.e. for cases that have not yet progressed far into the process. For the specific case of the RF log, we observed inter-case dynamics caused by batching that at the earliest occur after the fourth activity in the process. For this reason we only created features for prefixes that are at the start of this step, i.e. $k = 4$ and as a result do not improve prediction performance for prefixes of length $k < 4$. It would be better to incorporate this feature also for smaller prefix-lengths, but in these cases, we do not yet know the exact arrival time of that case within a batching pattern. Since this arrival time is crucial when predicting how a case progresses subject to batching, we are at the moment not yet able to incorporate such inter-case features prior to the start of such mechanisms.

The third limitation, somewhat related to the previous, is that our method is currently very much limited to creating features for mechanisms that arise towards the end of a process. This is due to the fact that we are dealing with remaining time prediction and those predictions only concern the end of the process, not the steps in between. Because of the output of this type of prediction, i.e. a single number for the remaining time, the overlaid performance spectrum also only visualizes these predictions until the end, not until the end of certain activities. As a result, if an inter-case mechanism occurs early on in the process and a lot of activities still need to occur, its corresponding pattern might very well not even be reflected anymore in the end, as was the case with the batching pattern in the segment *Create Fine:Send Fine*, making it hard to determine whether that mechanism is the cause of a high prediction error. If we were able to know the predicted remaining time of a case within a segment, we would be much better able to analyze whether the high prediction error is caused by the mechanism in that particular segment. However, this would be remaining time prediction until the next step and not remaining cycle time prediction until the end of the process. In that sense, the fact that we are dealing with remaining time prediction until the end of the process limits the degree in which we can perform an inter-case error analysis.

## 10.2 Future Work

A first direction for future work is to improve the accuracy of the current created inter-case features. As we have shown, we were able to improve the prediction error using these features. However, for their creation, we first used a classification model and on top of it, we used a model that predicts the time until batching. Both of these prediction models already produce an error on their own, but when put on top of one another, these errors pile up. We have also evaluated more ideal scenario's in which we provided the inter-case features with more accurate information on the batching mechanisms. An avenue for future work would be to improve these prediction components, or even completely replace them with something different, to get closer to the desired situation we have sketched with these more accurate inter-case features.

A second direction for future work is to apply the proposed methods to other types of remaining time prediction approaches. In this thesis, we have specifically addressed approaches with different bucketing and encoding techniques, but in the end, all were based on a machine learning algorithm. As we have already discussed, there are also approaches based on process model representations, neural networks or other modeling techniques. Testing the techniques proposed in this thesis on other methods would increase their validity.

A third direction for future work is the inclusion of other inter-case dynamics to the life cycle of remaining time prediction. The pattern taxonomy in [1] already defines a lot more patterns other than batching that are worthy to investigate. For example in an instance of the LIFO pattern, where the order of cases is actually switched around and where, as a result, the first arriving case waits the longest and the last arriving case waits the shortest. If current remaining time prediction approaches are not aware of batching mechanisms, they will most likely also not be aware of LIFO mechanisms. The inclusion of other types of inter-case patterns might very well increase the prediction performance of remaining time prediction methods as well.

A fourth direction is to address the problem of not only making predictive process monitoring more explainable w.r.t. dynamics between process instances, but also w.r.t. other types of dynamics. These process instances not only rely on each other and on resources, but also on materials, other systems and even on other processes for that matter, which cannot be discovered when analyzing the path of a single case. As the process analytics literature lacks concepts and models for such *multi-dimensional process dynamics*, a future avenue would be to make a start with deriving these patterns and concepts that can describe these multi-dimensional process dynamics from event data.

# Bibliography

[1] V. Denisov, D. Fahland, and W. M. P. van der Aalst, "Unbiased, Fine-Grained Description of Processes Performance from Event Data," in *BPM 2018*, vol. 11080 of *LNCS*, pp. 139–157, Springer, 2018. xii, 6, 7, 22, 34, 38, 39, 73

[2] E. L. Klijn and D. Fahland, "Performance Mining for Batch Processing Using the Performance Spectrum," *Lecture Notes in Business Information Processing*, vol. 362 LNBIP, pp. 172–185, 2019. xvii, 23, 49, 50, 51, 53, 58, 82

[3] M. zur Muehlen and R. Shapiro, *Business Process Analytics*, pp. 137–157. Springer Berlin Heidelberg, 08 2010. 1

[4] I. Verenich, M. Dumas, M. La Rosa, F. M. Maggi, and I. Teinemaa, "Survey and Cross-Benchmark Comparison of Remaining Time Prediction Methods in Business Process Monitoring," *ACM Trans. Intell. Syst. Technol.*, vol. 10, July 2019. 1, 2, 10, 11, 12, 13, 14, 16, 17, 18, 21, 29, 40, 57, 61, 69

[5] M. Dumas, M. La Rosa, J. Mendling, and H. A. Reijers, *Fundamentals of Business Process Management.* Springer, 2013. 1

[6] D. T. Larose and C. T. Larose, *Discovering Knowledge in Data: An Introduction to Data Mining.* USA: Wiley-Interscience, 2014. 1

[7] N. Tax, I. Verenich, M. La Rosa, and M. Dumas, "Predictive Business Process Monitoring with LSTM Neural Networks," in *Proceedings of the 29th International Conference on Advanced Information Systems Engineering (CAiSE)*, pp. 477–492, Springer, 2017. 2, 11, 13, 15, 16, 22, 48

[8] V. Denisov, E. Belkina, D. Fahland, and W. M. P. van der Aalst, "The Performance Spectrum Miner: Visual Analytics for Fine-Grained Performance Analysis of Processes," in *BPM 2018 Demos*, vol. 2196 of *CEUR Workshop Proceedings*, pp. 96–100, CEUR-WS.org, 2018. 2, 22, 51

[9] W. M. P. van der Aalst, *Process Mining: Data Science in Action.* Springer, 2 ed., 2016. 5, 16

[10] R. Khanna and M. Awad, *Efficient Learning Machines: Theories, Concepts, and Applications for Engineers and System Designers.* Apress, 04 2015. 7

[11] D. Sarkar, R. Bali, and T. Sharma, *Practical Machine Learning with Python: A Problem-Solver's Guide to Building Real-World Intelligent Systems.* USA: Apress, 1st ed., 2017. 9, 11, 16, 17, 49

[12] M. De Leoni and F. Mannhardt, "Road Traffic Fine Management Process." Dataset, 2015. https://doi.org/10.4121/uuid:270fd440-1057-4fb9-89a9-b699b47990f5. 10

[13] E. Cesario, F. Folino, M. Guarascio, and L. Pontieri, "A Cloud-Based Prediction Framework for Analyzing Business Process Performances.," in *CD-ARES*, vol. 9817 of *Lecture Notes in Computer Science*, pp. 63–80, Springer, 2016. 11, 15

[14] A. Rogge-Solti and M. Weske, "Prediction of Business Process Durations Using Non-Markovian Stochastic Petri Nets," *Information Systems*, vol. 54, pp. 1–14, 2015. 11, 13, 15, 16

[15] W. M. P. van der Aalst, M. H. Schonenberg, and M. Song, "Time Prediction Based on Process Mining," *Information Systems*, vol. 36, no. 2, pp. 450–475, 2011. 11, 15, 16

[16] M. Polato, A. Sperduti, A. Burattin, and M. de Leoni, "Time and Activity Sequence Prediction of Business Process Instances," *Computing*, vol. 100, pp. 1005–1031, 2016. 11, 15, 16, 48

[17] I. Verenich, H. Nguyen, M. La Rosa, and M. Dumas, "White-Box Prediction of Process Performance Indicators via Flow Analysis," in *Proceedings of the 2017 International Conference on Software and System Process*, ICSSP 2017, pp. 85–94, Association for Computing Machinery, 2017. 11, 15, 16

[18] A. Senderovich, M. Weidlich, A. Gal, and A. Mandelbaum, "Queue Mining for Delay Prediction in Multi-Class Service Processes," *Information Systems*, vol. 53, pp. 278–295, 2015. 11, 16

[19] F. Folino, M. Guarascio, and L. Pontieri, "Discovering Context-Aware Models for Predicting Business Process Performances," in *On the Move to Meaningful Internet Systems: OTM 2012*, (Berlin, Heidelberg), pp. 287–304, Springer Berlin Heidelberg, 2012. 11, 12, 15, 16

[20] M. De Leoni, W. M. P. van der Aalst, and M. Dees, "A General Process Mining Framework for Correlating, Predicting and Clustering Dynamic Behavior Based on Event Logs," *Information Systems*, vol. 56, pp. 235–257, 2016. 11, 15, 16

[21] A. Senderovich, C. Di Francescomarino, and F. M. Maggi, "From Knowledge-Driven to Data-Driven Inter-Case Feature Encoding in Predictive Process Monitoring," *Information Systems*, vol. 84, pp. 255–264, 2019. 11, 12, 13, 15, 16

[22] C. C. Aggarwal, *Data Mining: The Textbook.* Springer, 2015. 12, 13, 14

[23] A. Zheng and A. Casari, *Feature Engineering for Machine Learning: Principles and Techniques for Data Scientists.* Wiley & Sons, 2nd ed., 2018. 13

[24] I. Teinemaa, M. Dumas, M. La Rosa, and F. M. Maggi, "Outcome-Oriented Predictive Process Monitoring: Review and Benchmark," *ACM Transactions on Knowledge Discovery from Data*, vol. 13, no. 2, pp. 1–57, 2019. 14, 15, 49

[25] I. Verenich, *Explainable Predictive Monitoring of Temporal Measures of Business Processes.* PhD thesis, Queensland University of Technology, 2018. 15

[26] R. Andrews, S. Suriadi, M. T. Wynn, A. H. ter Hofstede, A. Pika, H. H. Nguyen, and M. La Rosa, "Comparing Static and Dynamic Aspects of Patient Flows via Process Model Visualisations," 2016. 15

[27] B. F. van Dongen, R. A. Crooy, and W. M. P. van der Aalst, "Cycle Time Prediction: When Will This Case Finally Be Finished?," *Lecture Notes in Computer Science*, vol. 5331 LNCS, pp. 319–336, 2008. 15

[28] A. Leontjeva, R. Conforti, C. Di Francescomarino, M. Dumas, and F. M. Maggi, "Complex Symbolic Sequence Encodings for Predictive Monitoring of Business Processes," in *Business Process Management*, vol. 1, pp. 297–313, 2015. 15

[29] A. Pika, W. M. P. van der Aalst, C. J. Fidge, A. H. M. ter Hofstede, and M. T. Wynn, "Predicting deadline transgressions using event logs," in *Business Process Management Workshops* (M. La Rosa and P. Soffer, eds.), (Berlin, Heidelberg), pp. 211–216, Springer Berlin Heidelberg, 2013. 16

[30] A. Pika, W. M. P. van der Aalst, C. J. Fidge, A. H. M. ter Hofstede, and M. T. Wynn, "Profiling event logs to configure risk indicators for process delays," in *Advanced Information Systems Engineering* (C. Salinesi, M. C. Norrie, and Ó. Pastor, eds.), (Berlin, Heidelberg), pp. 465–481, Springer Berlin Heidelberg, 2013. 16

[31] J. P. Mueller and L. Massaron, "Machine learning: Creating your own features in data." 45

[32] P. J. Brockwell and R. A. Davis, *Introduction to Time Series and Forecasting.* Springer, 3rd ed., 2016. 51

# Appendix A

# Methodology Results of Running Example

## A.1 OPS & OPS' Visualizations of predictions of model $RM_{p,a,x}$

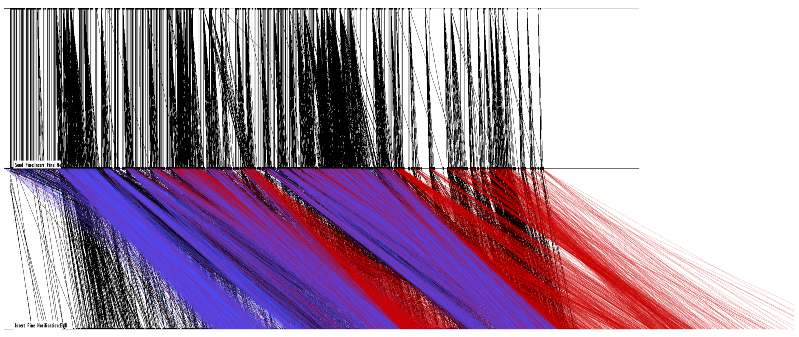

**Figure A.1:** OPS of predictions of $RM_{p,a,x}$ for segment *Send Fine:Insert Fine Notification* for $k = 2$

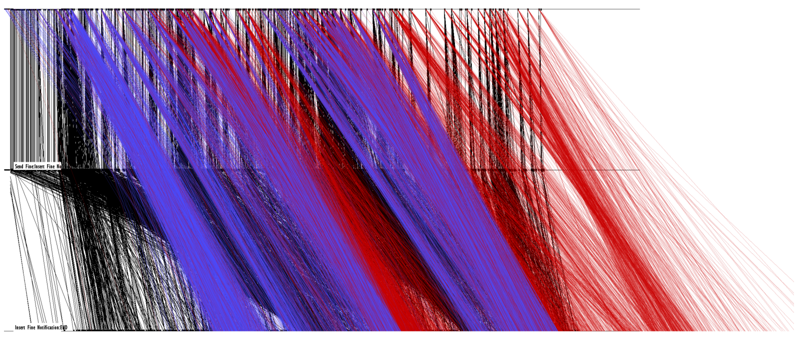

**Figure A.2:** OPS' of predictions of $RM_{p,a,x}$ for segment *Send Fine:Insert Fine Notification* for $k = 2$

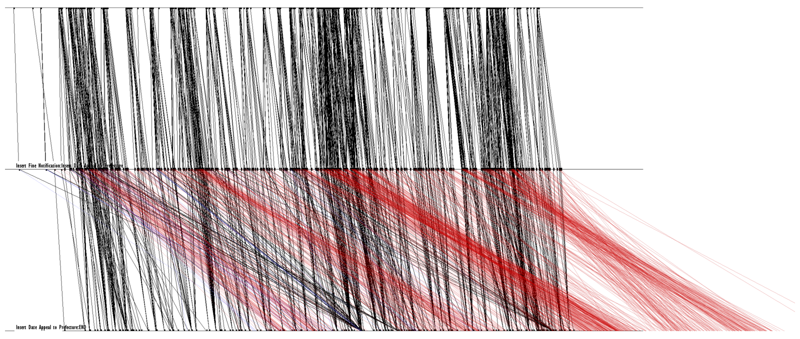

**Figure A.3:** OPS of predictions of $RM_{p,a,x}$ for segment *Insert Fine Notification:Insert Date Appeal to Prefecture* for $k = 3$

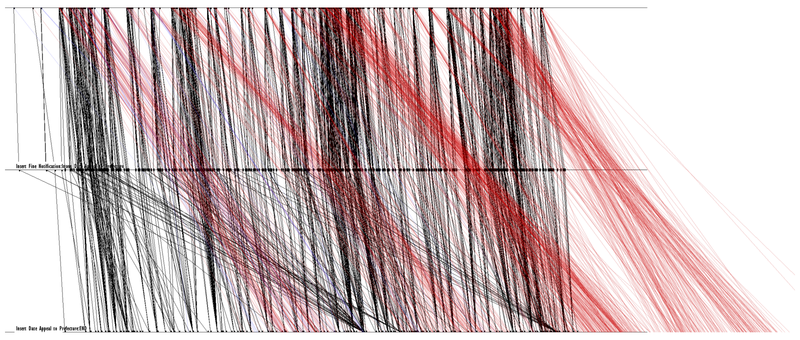

**Figure A.4:** OPS' of predictions of $RM_{p,a,x}$ for segment *Insert Fine Notification:Insert Date Appeal to Prefecture* for $k = 3$

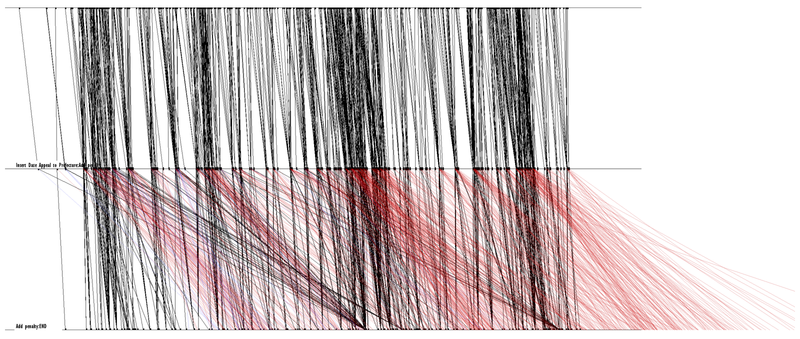

**Figure A.5:** OPS of predictions of $RM_{p,a,x}$ for segment *Insert Date Appeal to Prefecture:Add Penalty* for $k = 4$

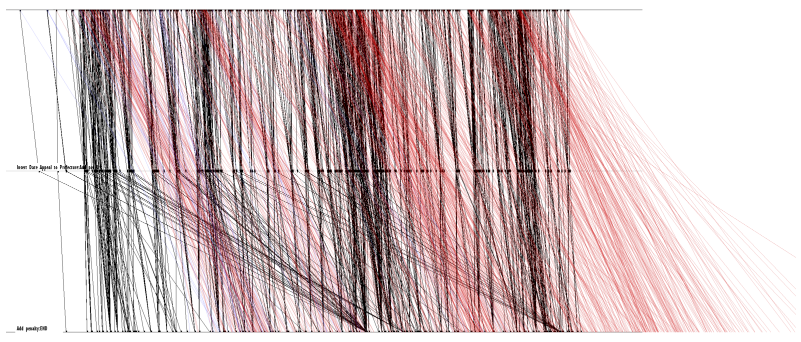

**Figure A.6:** OPS' of predictions of $RM_{p,a,x}$ for segment *Insert Date Appeal to Prefecture:Add Penalty* for $k = 4$

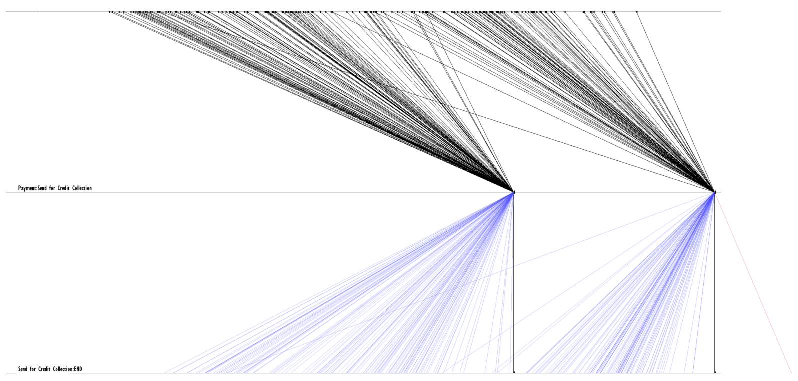

**Figure A.7:** OPS of predictions of $RM_{p,a,x}$ for segment *Payment:Send for Credit Collection* for $k = 5$

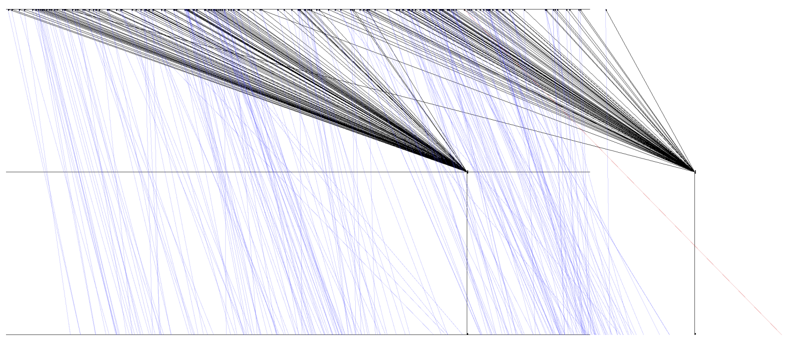

**Figure A.8:** OPS' of predictions of $RM_{p,a,x}$ for segment *Payment:Send for Credit Collection* for $k = 5$

## A.2 Batch Parameter Derivation for $S$=(-, Send for Credit Collection) of RF Log

**Table A.1:** Batch parameter values for $S$ = (-, Send for Credit Collection) of RF log, extracted with batch miner [2]

| $i$ | $BM_i$ | $BI_i$ (days) | $W_{i,min}$ (days) |
|---|---|---|---|
| 1 | 09-04-2001 23:00 | - | 315 |
| 2 | 09-04-2002 23:00 | 365 | 315 |
| 3 | 10-01-2003 00:00 | 275 | 225 |
| 4 | 10-01-2004 00:00 | 365 | 230 |
| 5 | 25-12-2004 00:00 | 350 | 209 |
| 6 | 28-02-2006 00:00 | 430 | 274 |
| 7 | 28-02-2007 00:00 | 365 | 274 |
| 8 | 29-03-2009 23:00 | 760 | 251 |
| 9 | 14-10-2010 23:00 | 564 | 412 |
| 10 | 25-03-2012 23:00 | 528 | 301 |
| 11 | 23-04-2013 23:00 | 394 | 330 |

**Table A.2:** $\overline{BI}_i$ results of exponential smoothing with $\alpha \in \{0.1, 0.2, 0.3, 0.4, 0.5\}$ for $L_{train}$ & $L_{test}$

| | $i$ | $BI_i$ | $\alpha = 0.5$ | $\alpha = 0.4$ | $\alpha = 0.3$ | $\alpha = 0.2$ | $\alpha = 0.1$ |
|---|---|---|---|---|---|---|---|
| $L_{train}$ | 1* | 365* | | | | | |
| | 2 | 365 | 365.00 | 365.00 | 365.00 | 365.00 | 365.00 |
| | 3 | 275 | 365.00 | 365.00 | 365.00 | 365.00 | 365.00 |
| | 4 | 365 | 320.00 | 365.00 | 338.00 | 347.00 | 356.00 |
| | 5 | 350 | 342.50 | 342.50 | 346.10 | 350.60 | 356.90 |
| | 6 | 430 | 346.25 | 342.50 | 347.27 | 350.48 | 356.21 |
| | 7 | 365 | 388.13 | 344.38 | 372.09 | 366.38 | 363.60 |
| | 8 | 760 | 376.56 | 366.25 | 369.96 | 366.11 | 363.73 |
| | 9 | 564 | 568.28 | 371.41 | 486.97 | 444.89 | 403.36 |
| MAE$_{train}$ | | | 91.01 | 91.01 | 96.83 | 100.36 | 105.43 |
| $L_{test}$ | 10 | 528 | 566.14 | 469.84 | 510.08 | 468.71 | 419.42 |
| | 11 | 394 | 547.07 | 517.99 | 515.46 | 480.57 | 430.28 |
| MAE$_{test}$ | | | 95.60 | 95.61 | 69.69 | 72.93 | 72.43 |

**Table A.3:** $\overline{W}_{i,min}$ results of exponential smoothing with $\alpha \in \{0.1, 0.2, 0.3, 0.4, 0.5\}$ for $L_{train}$ & $L_{test}$

| | $i$ | $W_{i,min}$ | $\alpha = 0.5$ | $\alpha = 0.4$ | $\alpha = 0.3$ | $\alpha = 0.2$ | $\alpha = 0.1$ |
|---|---|---|---|---|---|---|---|
| $L_{train}$ | 0* | 7560* | | | | | |
| | 1 | 7560 | 7560.00 | 7560.00 | 7560.00 | 7560.00 | 7560.00 |
| | 2 | 7560 | 7560.00 | 7560.00 | 7560.00 | 7560.00 | 7560.00 |
| | 3 | 5401 | 7560.00 | 7560.00 | 7560.00 | 7560.00 | 7560.00 |
| | 4 | 5521 | 6480.50 | 6696.40 | 6912.30 | 7128.20 | 7344.10 |
| | 5 | 5017 | 6000.75 | 6226.24 | 6494.91 | 6806.76 | 7161.79 |
| | 6 | 6577 | 5508.88 | 5742.54 | 6051.54 | 6448.81 | 6947.31 |
| | 7 | 6577 | 6042.94 | 6076.31 | 6209.18 | 6474.45 | 6910.28 |
| | 8 | 6024 | 6309.97 | 6276.60 | 6319.52 | 6494.96 | 6876.96 |
| | 9 | 9888 | 6166.98 | 6175.56 | 6230.87 | 6400.77 | 6791.66 |
| MAE$_{train}$ | | | 1213.93 | 1230.48 | 1234.27 | 1218.11 | 1347.47 |
| $L_{test}$ | 10 | 7224 | 8027.50 | 7660.54 | 7328.01 | 7098.21 | 7101.29 |
| | 11 | 7920 | 7625.75 | 7485.92 | 7296.80 | 7123.37 | 7113.56 |
| MAE$_{test}$ | | | 548.87 | 435.31 | 363.60 | 461.21 | 464.57 |

# Appendix B

# Inter-Case Feature Testing Results

This appendix chapter contains the results of the different input configurations $I$ used for the inter-case feature testing on method $(p, a, x)$ in Section 9.1. Section B.1 contains the histogram results and Section B.2 contains the OPS and OPS' visualizations.

## B.1 Histogram Results

This appendix section contains the histograms of different input configurations $I$. Section B.1.1 contains the results of the input configurations $I(\overline{c}_R)$ that have inter-case features based on the predicted classification and Section B.1.2 contains the results of the input configurations $I(c_R)$ that have inter-case features based on the actual classification.

### B.1.1 Histograms of configurations $I(\bar{c}_R)$

This subsection contains histograms of the actual and predicted remaining times of cases part of a batch $\overline{y}_{\in R}$ (Figure B.1) and cases not part of a batch $\overline{y}_{\notin R}$ (Figure B.2) and histograms of the actual and predicted interdeparture times of cases part of a batch $\overline{ID}_{\in R}$ (Figure B.3) and cases not part of a batch $\overline{ID}_{\notin R}$ (Figure B.4).

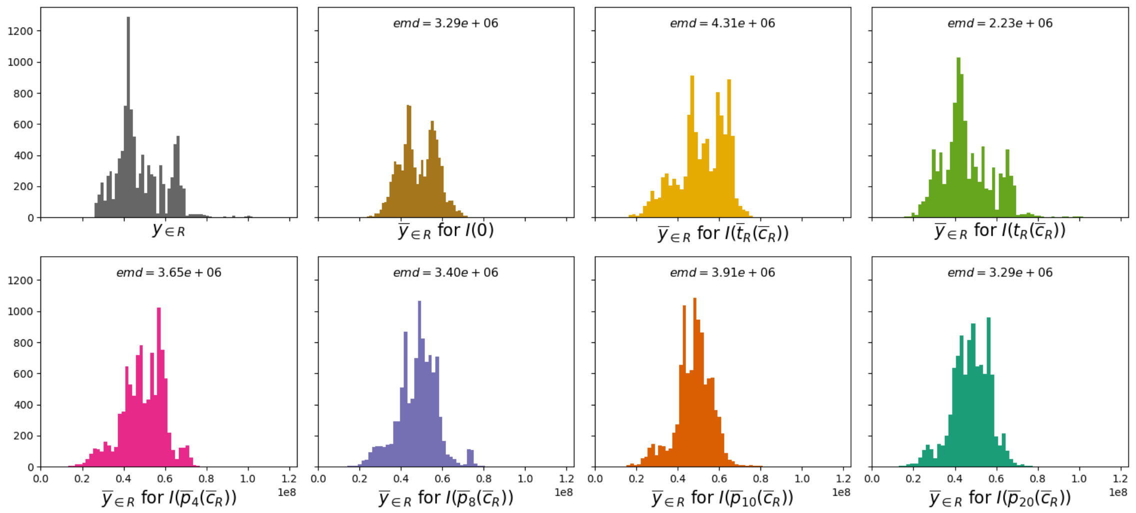

**Figure B.1:** Histograms of actual $y_{\in R}$ and predicted $\overline{y}_{\in R}$ for $I(0)$, $I(\overline{d}(\bar{c}_R))$ with different inter-case features for $\overline{d}$ and $I(t_R(\bar{c}_R))$ for method $(p, a, x)$

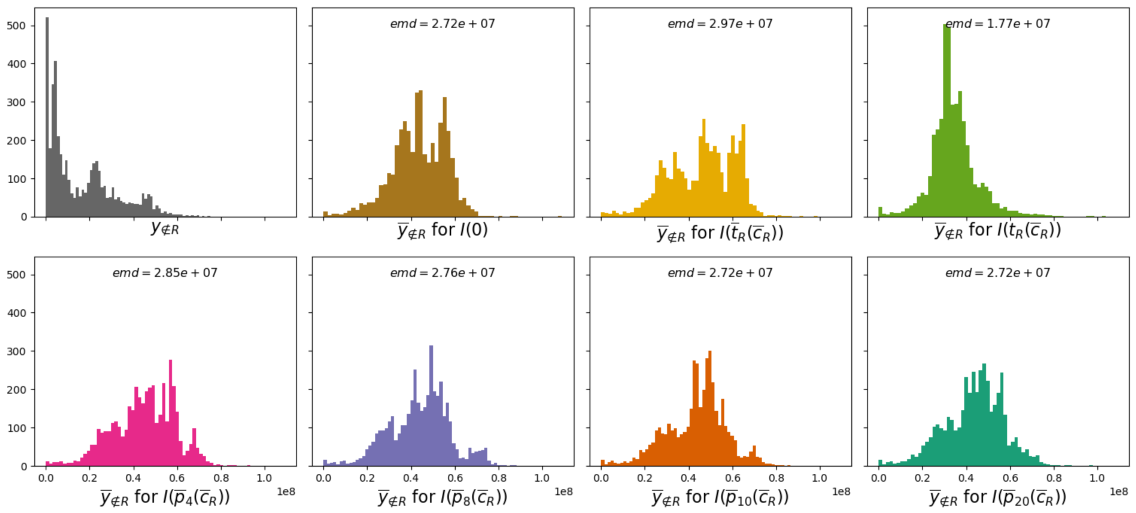

**Figure B.2:** Histograms of actual $y_{\notin R}$ and predicted $\overline{y}_{\notin R}$ for $I(0)$, $I(\overline{d}(\bar{c}_R))$ with different inter-case features for $\overline{d}$ and $I(t_R(\bar{c}_R))$ for method $(p, a, x)$

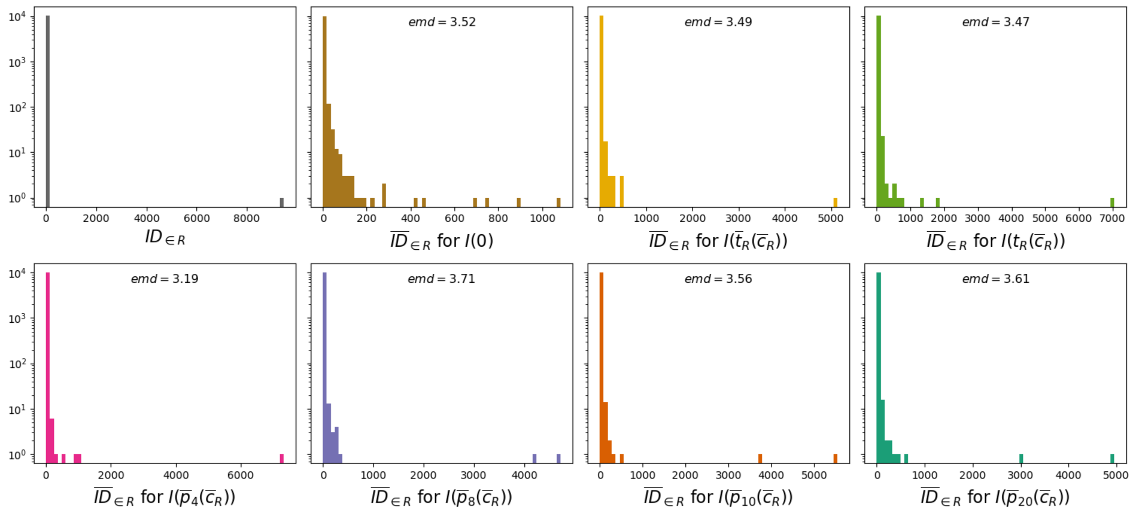

**Figure B.3:** Histograms of actual $ID_{\in R}$ and predicted $\overline{ID}_{\in R}$ for $I(0)$, $I(\overline{d}(\overline{c}_R))$ with different inter-case features for $\overline{d}$ and $I(t_R(\overline{c}_R))$ for method $(p, a, x)$

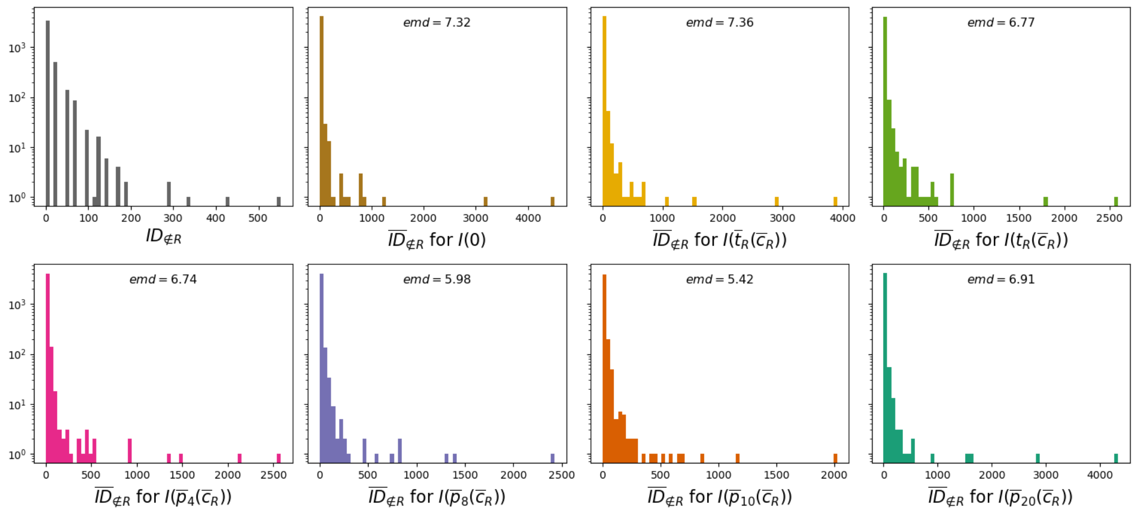

**Figure B.4:** Histograms of actual $ID_{\notin R}$ and predicted $\overline{ID}_{\notin R}$ for $I(0)$, $I(\overline{d}(\overline{c}_R))$ with different inter-case features for $\overline{d}$ and $I(t_R(\overline{c}_R))$ for method $(p, a, x)$

### B.1.2 Histograms of configurations $I(c_R)$

This subsection contains histograms of the actual and predicted remaining times of cases part of a batch $\overline{y}_{\in R}$ (Figure B.5) and cases not part of a batch $\overline{y}_{\notin R}$ (Figure B.6) and histograms of the actual and predicted interdeparture times of cases part of a batch $\overline{ID}_{\in R}$ (Figure B.7) and cases not part of a batch $\overline{ID}_{\notin R}$ (Figure B.8).

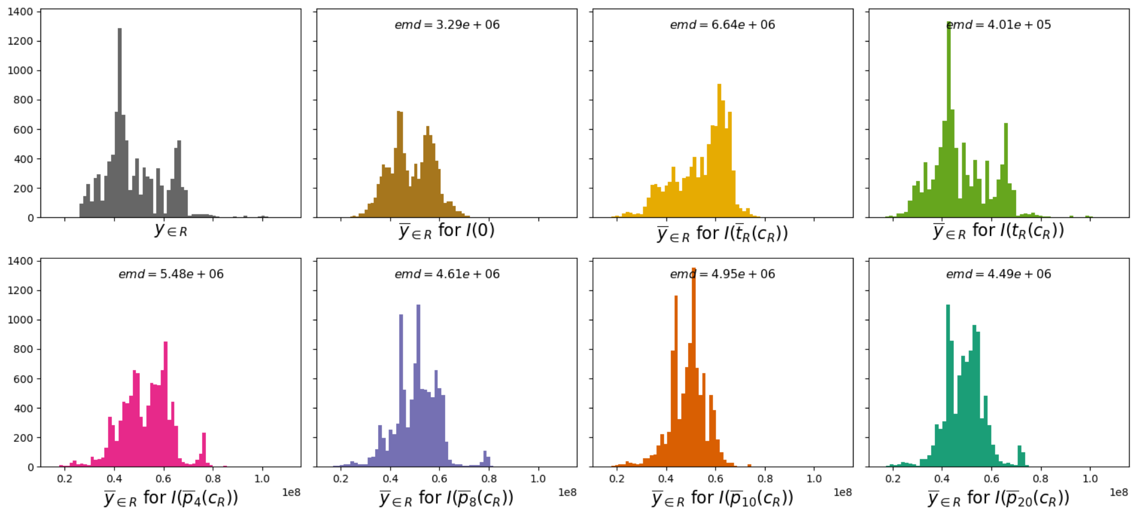

**Figure B.5:** Histograms of actual $y_{\in R}$ and predicted $\overline{y}_{\in R}$ for $I(0)$, $I(\overline{d}(c_R))$ with different inter-case features for $\overline{d}$ and $I(t_R(c_R))$ for method $(p, a, x)$

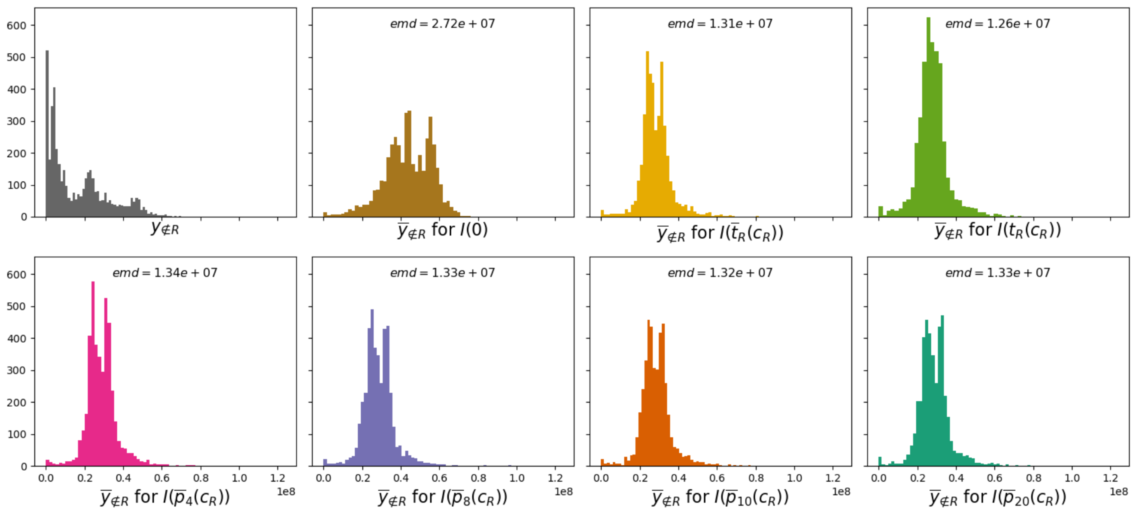

**Figure B.6:** Histograms of actual $y_{\notin R}$ and predicted $\overline{y}_{\notin R}$ for $I(0)$, $I(\overline{d}(c_R))$ with different inter-case features for $\overline{d}$ and $I(t_R(c_R))$ for method $(p, a, x)$

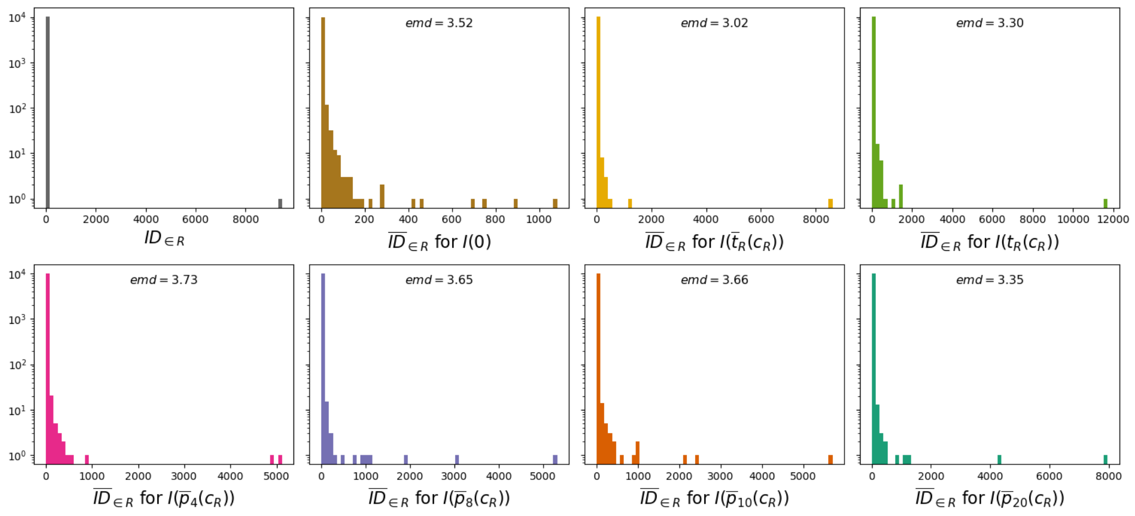

**Figure B.7:** Histograms of actual $ID_{\in R}$ and predicted $\overline{ID}_{\in R}$ for $I(0)$, $I(\overline{d}(c_R))$ with different inter-case features for $\overline{d}$ and $I(t_R(c_R))$ for method $(p, a, x)$

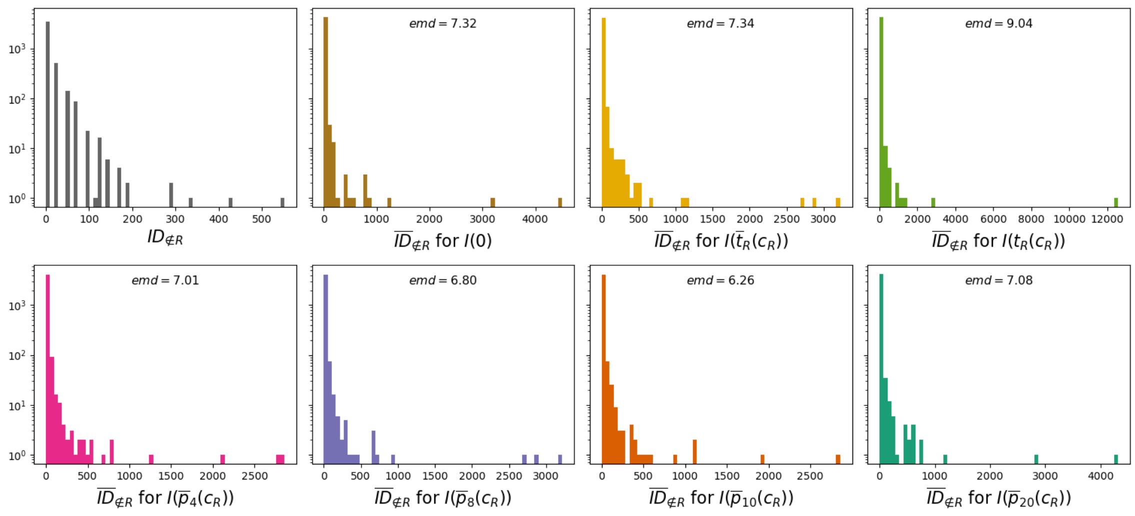

**Figure B.8:** Histograms of actual $ID_{\notin R}$ and predicted $\overline{ID}_{\notin R}$ for $I(0)$, $I(\overline{d}(c_R))$ with different inter-case features for $\overline{d}$ and $I(t_R(c_R))$ for method $(p, a, x)$

## B.2 OPS and OPS' Visualizations

Similar to Section B.1, this section contains the OPS and OPS' results of different input configurations $I$. Section B.2.1 contains the results of the input configuration $I(\overline{c}_R)$ and Section B.2.2 contains the results of the input configurations $I(c_R)$.

### B.2.1 Results of configurations $I(\overline{c}_R)$

This subsection contains the OPS and OPS' visualizations of $\mathcal{P}_I$ of all input configurations $I(\overline{c}_R)$.

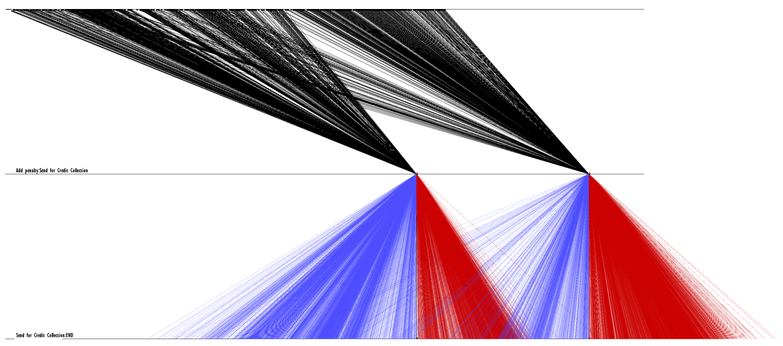

**Figure B.9:** OPS of predictions of $I(\bar{t}_R(\overline{c}_R))$ and method $(p, a, x)$ for segment *Add Penalty:Send for Credit Collection* for $k = 4$

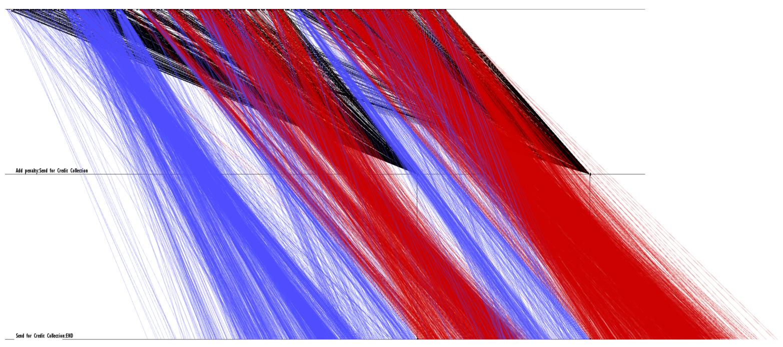

**Figure B.10:** OPS' of predictions of $I(\bar{t}_R(\overline{c}_R))$ and method $(p, a, x)$ for segment *Add Penalty:Send for Credit Collection* for $k = 4$

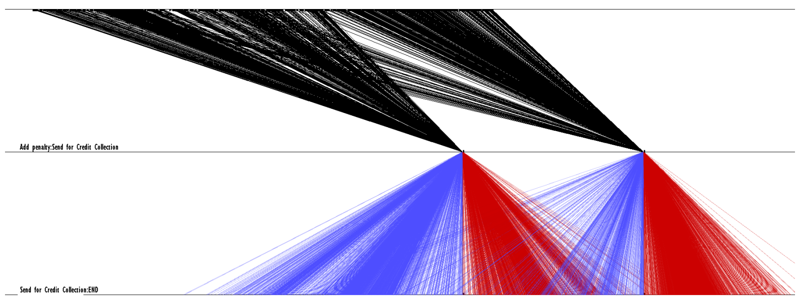

**Figure B.11:** OPS of predictions of $I(\overline{p}_4(\overline{c}_R))$ and method $(p, a, x)$ for segment *Add Penalty:Send for Credit Collection* for $k = 4$

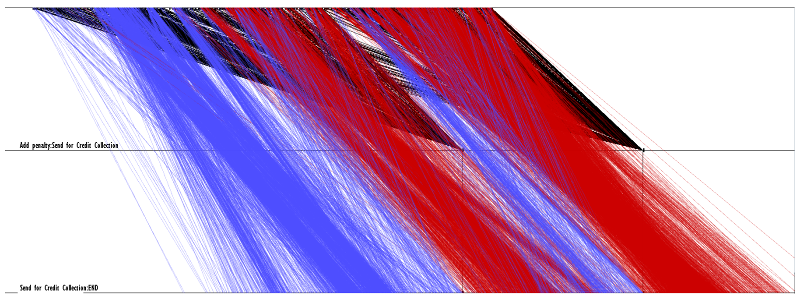

**Figure B.12:** OPS' of predictions of $I(\overline{p}_4(\overline{c}_R))$ and method $(p, a, x)$ for segment *Add Penalty:Send for Credit Collection* for $k = 4$

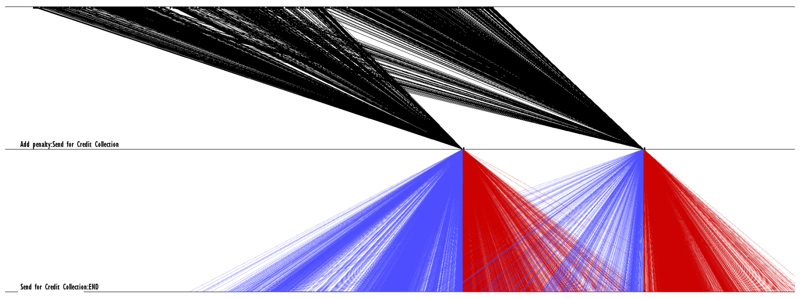

**Figure B.13:** OPS of predictions of $I(\overline{p}_8(\overline{c}_R))$ and method $(p, a, x)$ for segment *Add Penalty:Send for Credit Collection* for $k = 4$

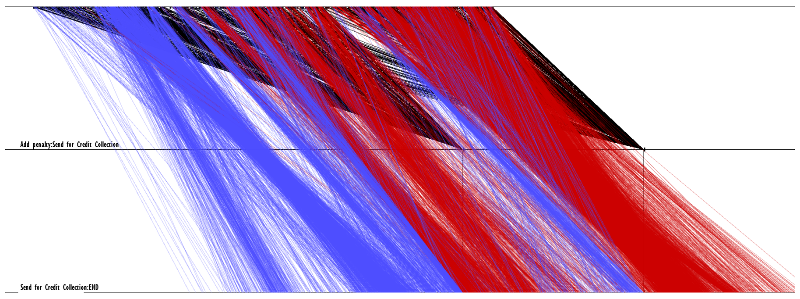

**Figure B.14:** OPS' of predictions of $I(\overline{p}_8(\overline{c}_R))$ and method $(p, a, x)$ for segment *Add Penalty:Send for Credit Collection* for $k = 4$

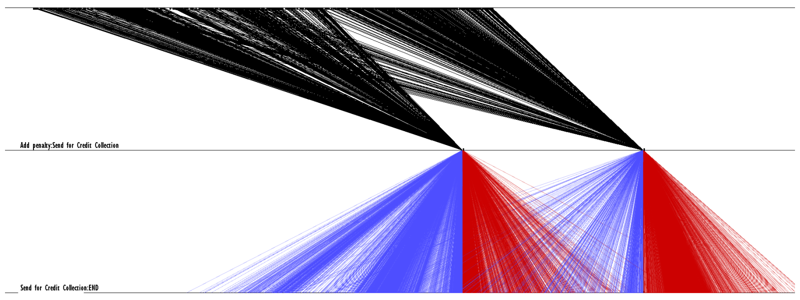

**Figure B.15:** OPS of predictions of $I(\overline{p}_{10}(\overline{c}_R))$ and method $(p, a, x)$ for segment *Add Penalty:Send for Credit Collection* for $k = 4$

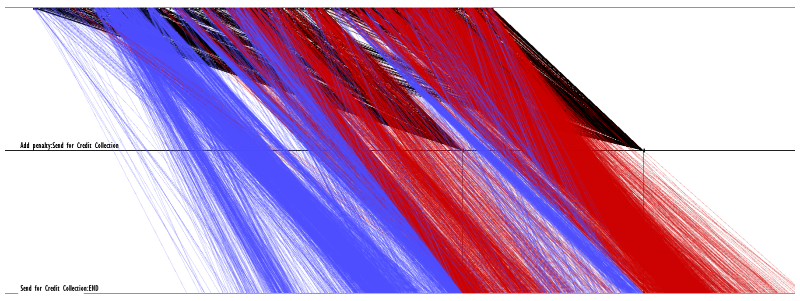

**Figure B.16:** OPS' of predictions of $I(\overline{p}_{10}(\overline{c}_R))$ and method $(p, a, x)$ for segment *Add Penalty:Send for Credit Collection* for $k = 4$

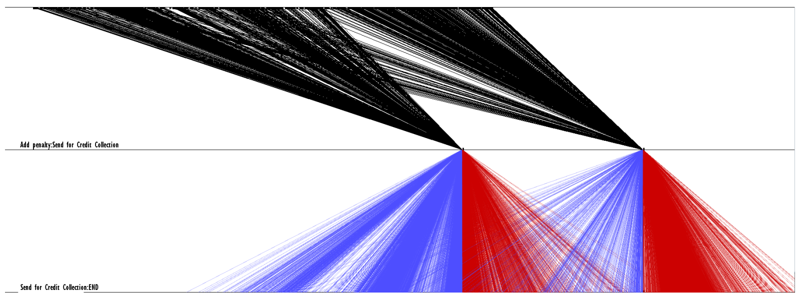

**Figure B.17:** OPS of predictions of $I(\overline{p}_{20}(\overline{c}_R))$ and method $(p, a, x)$ for segment *Add Penalty:Send for Credit Collection* for $k = 4$

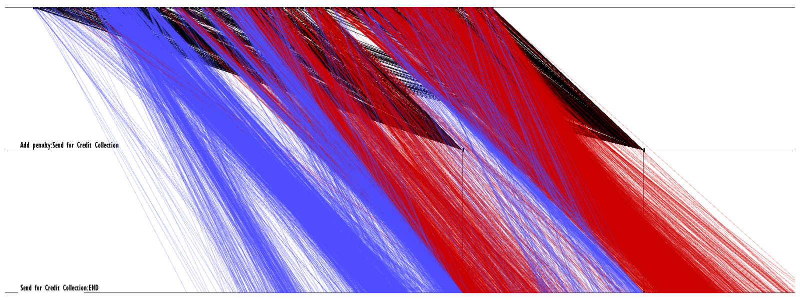

**Figure B.18:** OPS' of predictions of $I(\overline{p}_{20}(\overline{c}_R))$ and method $(p, a, x)$ for segment *Add Penalty:Send for Credit Collection* for $k = 4$

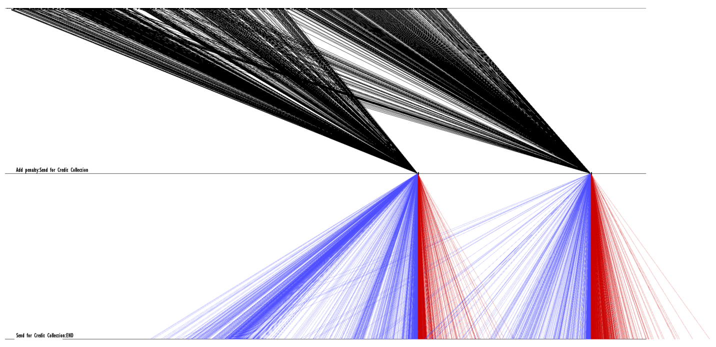

**Figure B.19:** OPS of predictions of $I(t_R(\overline{c}_R))$ and method $(p, a, x)$ for segment *Add Penalty:Send for Credit Collection* for $k = 4$

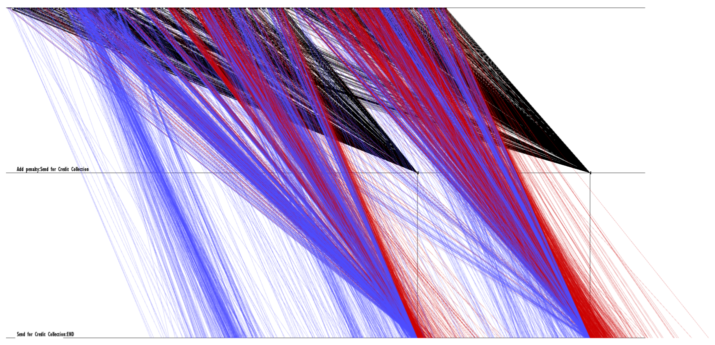

**Figure B.20:** OPS' of predictions of $I(t_R(\bar{c}_R))$ and method $(p, a, x)$ for segment *Add Penalty:Send for Credit Collection* for $k = 4$

### B.2.2 Results of $I(c_R)$

This subsection contains the OPS and OPS' visualizations of $\mathcal{P}_I$ of all input configurations $I(c_R)$.

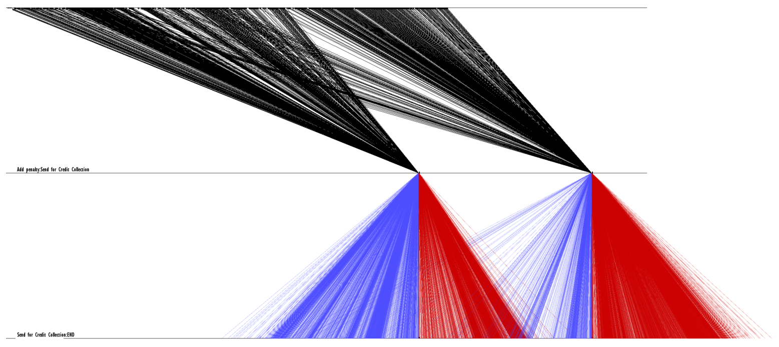

**Figure B.21:** OPS of predictions of $I(\bar{t}_R(c_R))$ and method $(p, a, x)$ for segment *Add Penalty:Send for Credit Collection* for $k = 4$

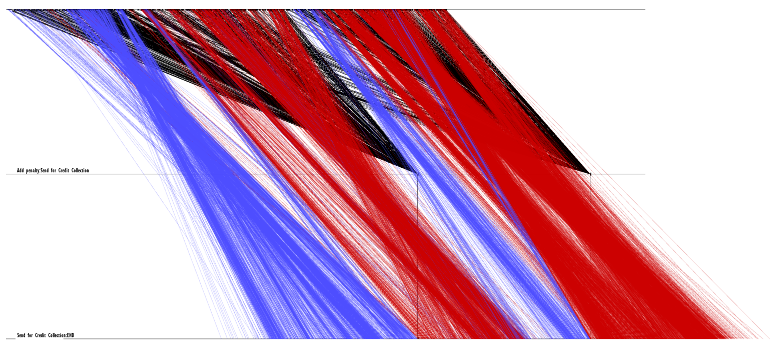

**Figure B.22:** OPS' of predictions of $I(\bar{t}_R(c_R))$ and method $(p, a, x)$ for segment *Add Penalty:Send for Credit Collection* for $k = 4$

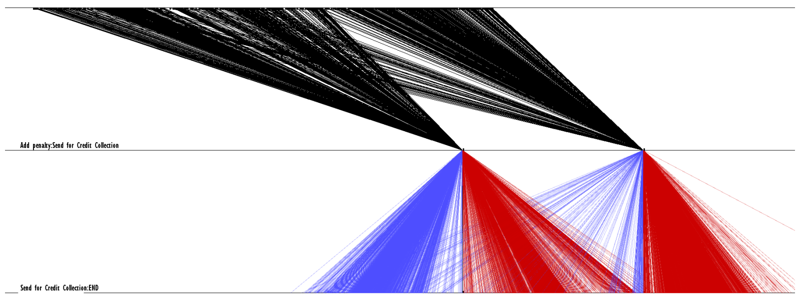

**Figure B.23:** OPS of predictions of $I(\overline{p}_4(c_R))$ and method $(p, a, x)$ for segment *Add Penalty:Send for Credit Collection* for $k = 4$

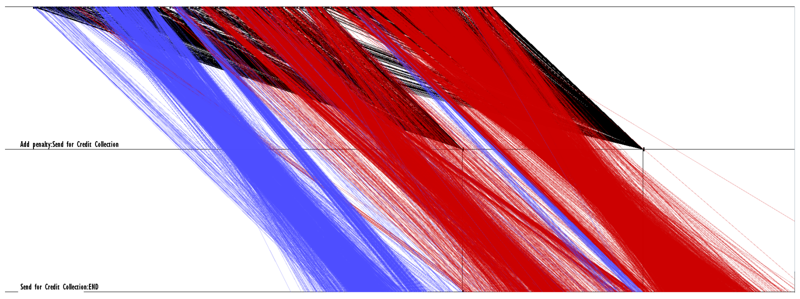

**Figure B.24:** OPS' of predictions of $I(\overline{p}_4(c_R))$ and method $(p, a, x)$ for segment *Add Penalty:Send for Credit Collection* for $k = 4$

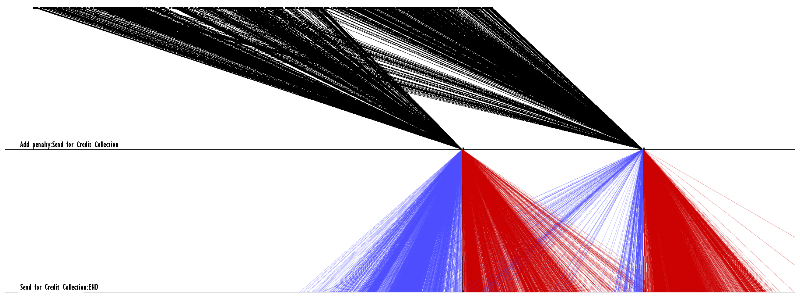

**Figure B.25:** OPS of predictions of $I(\overline{p}_8(c_R))$ and method $(p, a, x)$ for segment *Add Penalty:Send for Credit Collection* for $k = 4$

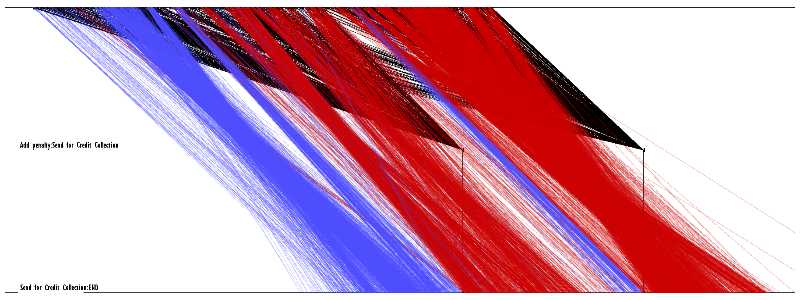

**Figure B.26:** OPS' of predictions of $I(\overline{p}_8(c_R))$ and method $(p, a, x)$ for segment *Add Penalty:Send for Credit Collection* for $k = 4$

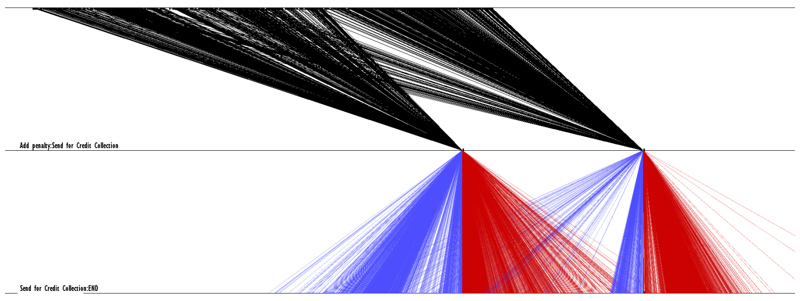

**Figure B.27:** OPS of predictions of $I(\overline{p}_{10}(c_R))$ and method $(p, a, x)$ for segment *Add Penalty:Send for Credit Collection* for $k = 4$

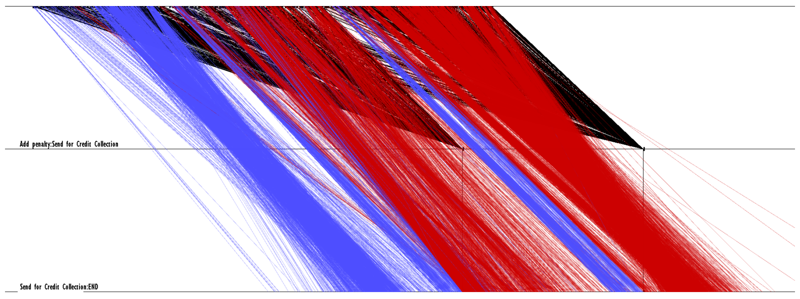

**Figure B.28:** OPS' of predictions of $I(\overline{p}_{10}(c_R))$ and method $(p, a, x)$ for segment *Add Penalty:Send for Credit Collection* for $k = 4$

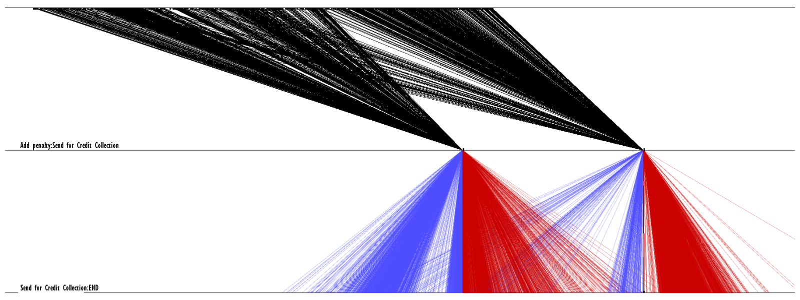

**Figure B.29:** OPS of predictions of $I(\overline{p}_{20}(c_R))$ and method $(p, a, x)$ for segment *Add Penalty:Send for Credit Collection* for $k = 4$

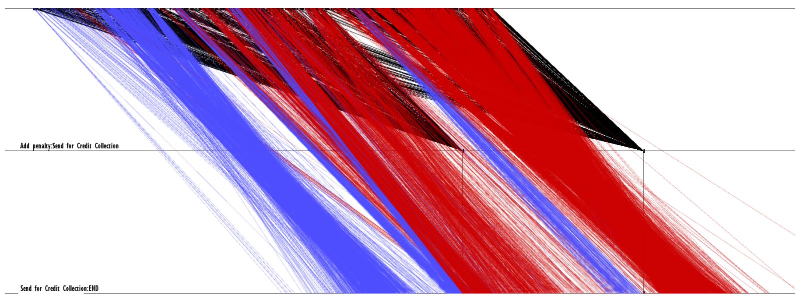

**Figure B.30:** OPS' of predictions of $I(\overline{p}_{20}(c_R))$ and method $(p, a, x)$ for segment *Add Penalty:Send for Credit Collection* for $k = 4$

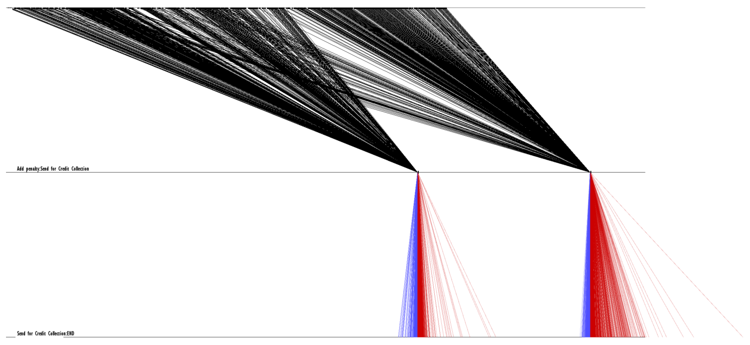

**Figure B.31:** OPS of predictions of $I(t_R(c_R))$ and method $(p, a, x)$ for segment *Add Penalty:Send for Credit Collection* for $k = 4$

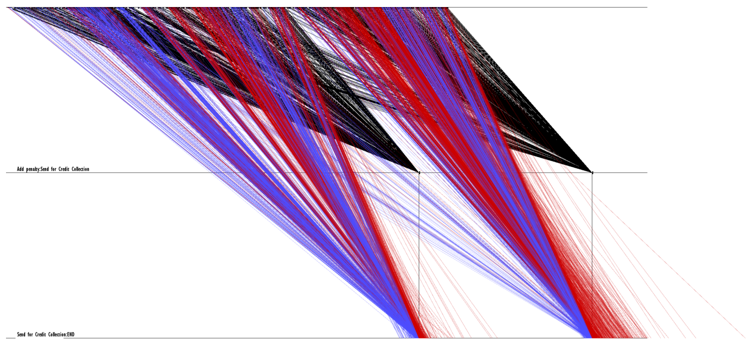

**Figure B.32:** OPS' of predictions of $I(t_R(c_R))$ and method $(p, a, x)$ for segment *Add Penalty:Send for Credit Collection* for $k = 4$