Eindhoven University of Technology

MASTER

Design, implementation and evaluation of a KPI-driven recommender system based on predictive process monitoring

Reulink, L.T.W.

Award date:
2019

Link to publication

Disclaimer
This document contains a student thesis (bachelor's or master's), as authored by a student at Eindhoven University of Technology. Student theses are made available in the TU/e repository upon obtaining the required degree. The grade received is not published on the document as presented in the repository. The required complexity or quality of research of student theses may vary by program, and the required minimum study period may vary in duration.

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain

EINDHOVEN UNIVERSITY OF TECHNOLOGY

Department of Mathematics and Computer Science
Architecture of Information Systems Research Group

Design, Implementation and Evaluation of a KPI-driven Recommender System based on Predictive Process Monitoring

Master Thesis

Laurens Theodorus Wilhelmus Reulink

Supervisors:
dr. Massimiliano de Leoni (TU/e)
ir. Marcus Dees (TU/e, UWV)
dr. Anna Wilbik (TU/e)

final version

Eindhoven, December 2018

Abstract

Predictive (business) process monitoring is a technique that makes predictions about the future state of the executions of a business process [17]. Data related to the business process is used to predict a process' future outcome state. When a future outcome for a process is predicted, process managers can use this prediction to assess the performance of the process and decide whether to use mitigation actions. The performance of a process is determined by Key Performance Indicators (KPI's). This research presents a methodology that generates predictions about the KPI of a running process instance. On top of that, this study also introduces a methodology for the generation of recommendations. The goal of these recommendations is to concretize the required mitigation actions to prevent negative KPI outcomes.
This study is executed in cooperation with UWV, the Dutch employee insurance agency. The reintegration process within UWV is used as the case for this study. The main goal of this process is to help unemployed people get back to work before they run out of their entitled unemployment benefits. The most important KPI in this process is whether a customer finds a job before running out of benefits. Data related to the reintegration is used in the design, implementation and evaluation of a predictive process monitoring methodology that also incorporates the ability to generate recommendations.
This study shows that techniques for predictive process monitoring can be extended for the generation of KPI-driven recommendations. This thesis discusses the outline of the methodology and shows advantages and drawbacks of the system. Suggestions for future work are also provided.

Preface

First of all I would like to thank Marcus Dees, my supervisor at UWV. Marcus has helped me out a lot throughout the project. He has helped me settle at UWV and also spend a lot of time retrieving and improving the data at UWV such that I could keep working on the project. Marcus and I also had a lot of fruitful discussions about the subject during the project, which often resulted into useful insights. On top of that Marcus has kept me sharp and provided me with the necessary feedback whenever I needed to.

Next I would also like to thank Massimiliano de Leoni for being my supervisor. Our weekly meetings were very interesting and I am very grateful that Massimiliano made time for me week in week out. Massimiliano's feedback was always useful, and together with Marcus, he challenged me to reach for the top.

I would also like to thank UWV for facilitating this research and providing me with all the resources I needed. I had a great time getting to know the organization, and the people inside it.

I would also like to thank my fellow students Mike de Roode and Teun Graafmans. Without their help and support I would never have achieved such good grades throughout my master program. Working with them on projects was always a joy, and they inspired me in many ways.

I would also like to thank my close friends in Eindhoven, who have always believed in me and did not refrain of expressing their admiration for all the things I was doing.

Finally, I would like to thank my family for the patience they have had with me throughout my study years. I know they have always wanted the best for me and they kept supporting me no matter what.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Data can be a powerful resource to monitor, analyze and improve business processes. Many organizations are sitting on large amounts of data and are not always using this data to its full potential. Data science is the field of study that creates value out of data. Data and process mining techniques are examples of applications of data science in practice, along with machine learning. One type of machine learning algorithms can generate models which have great predictive power when trained properly. One of the predictions such a model can make is what the outcome of a process is going to be. This outcome is measured in the form of Key Performance Indicators (KPI), which state when and to what extend the outcome of a process is satisfactory as defined by its stakeholders. Using predictions to analyze and monitor running processes early is called *predictive process monitoring*. Gaining insights in KPI outcomes, especially early on in the process, can have great benefits for organizations. For example, process managers can opt to interfere in the process if it is moving towards an unfavorable outcome. Being able to predict process outcomes with respect to a certain KPI can help to prevent problems and achieve overall better results.

When an organization is able to identify high risk process instances, it can also act on them. A next step in the management of these processes is to be able to steer the process instance into the right direction. Again, process data can be used to find the best ways to positively influence a process. When knowing which actions to undertake process managers and organizations as a whole have the opportunity to manage their processes in a different way. Data can be used to find the actions to take within a process. This thesis focuses on leveraging data mining techniques to monitor process executions to predict their expected KPI outcome and, when negative, enact appropriate mitigation action.

1.1 Thesis Context

This research is facilitated by UWV (Uitvoeringsinstituut Werknemersverzekeringen), which is the Dutch employee insurance agency. This government body is responsible for the handling of employee insurances in the Netherlands [1]. One of UWV's responsibilities is the handling of unemployment insurances under the Unemployment Insurance Act (WW, or Werkloosheidswet Werknemers). One of these aspects is helping people get back to work, a process called the reintegration process. Process data related to the reintegration of a customer is used for the development of the methodology described in this thesis. The reintegration process starts when a person who recently got unemployed makes a claim on the insurance. The process can end for multiple reasons, for example when the unemployed person finds a new employer or when the process reaches the point in time where the person is no longer entitled to receive social security benefits. The latter implies that the process has a max duration for each case. After surpassing this max duration, the customer automatically leaves the process and ends up receiving only

welfare payments, which are usually way lower than the social security under the WW. This is a very unfavorable outcome for both UWV and the unemployed person. Whether a process instance is going to reach this max duration is one of the important key performance indicators for this process. Within the reintegration process UWV provides services to help people get back to work as fast as possible.

1.2 Research Goal

UWV sits on huge amounts of data related to the reintegration process and wants to use this data in order to be able to predict the process outcomes. It also wants to know when and how to act on it. Having a system in place that can predict KPI's can be very valuable to UWV. One of the KPI's is the actual reintegration of a customer, e.g. whether a customer finds a new job before running out of benefits. People who might need extra attention can now be identified. UWV's also uses work coaches, to help customers in their attempts to get back to work. These coaches can use KPI predictions of running cases to identify risky cases such that they can better divide their time. Therefore, UWV wants to be able to predict what the outcome is going to be for each of the process instances. Business owners at UWV want to know if the data that is currently available can be used to develop a system than make predictions on process related KPIs. In the case of UWV the most important KPI is whether a process is successfully completed, i.e. whether a person finds a new employer. The latter would be the case when a person has used up all its entitled months of benefit payments and proceeds to the welfare process. Therefore, the first research goal of this thesis is:

Research Goal 1: *Analyzing, processing and transforming process data and create a reliable prediction model for process instance outcomes.*

The methodology that can predict process outcomes should also be extendable to other processes, organizations and with different types of KPIs. Hence, this research tries to create a methodology for UWV that could also be applied to other cases.
Once high risk cases can be identified, the work coaches at UWV might want to focus their energy on the high risk cases and help these people with services that are more tailored to them. A recommendation system is needed that can suggest which activities can best be executed in a specific case at a certain point in time. Therefore a second goal is to generate a system that can generate recommendations for customers taking into account how these customers have been dealt till that moment. These recommendations are elements of the designed process and should decrease the predicted risk and as a result increase the overall performance of the process as described in the form of KPI's. Subsequently, the second goal of this research is:

Research Goal 2: *Create a methodology that can generate tailored recommendations in the form of executable process elements in order to improve process KPI values of a process.*
Once again this methodology is more valuable when it is applicable in other situations.

To make the techniques accessible for business owners and other end-users a graphical user interface (GUI) must be generated. This GUI must be easy to use and incorporate all basic functions of both the prediction and recommendation systems. The GUI must be able to process data sets into a new prediction model. It must also be able to evaluate running cases and generate recommendations for these instances. A prototype for such a graphical interface must be created. Therefore, the third research goal is:

Research Goal 3: *Create a prototype for a graphical user interface that incorporates both the prediction and recommendation system for a process.*

This GUI should be reusable and work on new process data, whenever this becomes available.

Therefore, the GUI is kept as general as possible such that it is usable with different datasets containing process data.
UWV also wants to know, in the case of a successfully developed prediction and recommendation system, if such systems could be implemented and if it would function in practice. In order to evaluate the usefulness and quality of such systems employees that work within the process must be addressed. Their opinions and expert knowledge on this matter might prove meaningful. The employees for which such a system is most relevant in the case of UWV would be the work coaches. The fourth and final research goal of this thesis focuses on this problem.

Research Goal 4: *Identify the opinions of UWV work coaches on the usefulness, feasibility and quality of the developed prediction and recommendation system.*

1.3 Research Approach

In order to reach these goals the project is divided in multiple phases. In the first phase the available data is analyzed, after which a preprocessing algorithm is developed that prepares the data for machine learning. Consecutively, different machine learning techniques have been applied to the data to find the best predictor model. The predictor model is evaluated in order to achieve Research Goal 1.
The prediction technique generated in the first stage proved useful to generate the recommendations. To fulfill Research Goal 2 the sequence of activities that happened in a running process so far is extended with another activity, as if that activity is the first activity that happens next. A prediction for this hypothetical scenario is made using the predictor model. The prediction is used to see if the added activity results in lower risk levels compared to the original prediction. When this is the case, it can be concluded that the activity that was added does have a positive effect on the KPI of the process, hence it is an activity that could be recommended. For each possible recommendation *and* each unique process instance these steps must be repeated, resulting in a large set of combinations that need to be tested. In order to cope with this complexity additional solutions must be developed. These solutions must also take into account that only activities that are meaningful from the business perspective are included in this analysis. The recommendations are eventually tested with historical data to check whether they have happened before in similar cases and, if so, if they improved the chances of a positive process outcome.
To develop the GUI required for Research Goal 3 a set of mockups is created to establish the general design. The mechanics of the prediction and recommendation system are optimized to work within the GUI. After evaluating the best Python programming packages for creating graphical user interfaces, the mockups are implemented. After some rounds of evaluation, additional functionalities are build into the design, which is eventually tested for flaws and further optimized.
For the fourth research goal a survey has been developed to be conducted among work coaches to address their opinions about the systems. However, reactions from business owners and managers on the contents and direction of the survey in an early evaluation of the setup of the survey were negative. There, the Research Goal 4 was not fully achieved due to external circumstances. Still we aim to sketch the results obtained in this respect and the direction to continue in to reach the goal.

1.4 Thesis Outline

This thesis is structured as follows. Chapter 2 introduces aspects from process mining, data mining and machine learning this research builds upon.
Chapter 3 sketches the current situation at UWV with respect to the process that is analyzed, as well as the process data that is available for this research.
Chapter 4 describes the methodology used to create prediction model for process outcomes as well as the evaluation of a model generated using the methodology. Implications for the GUI are also

discussed here.
Chapter 5 describes how the recommendation system is created and how the data is interpreted to generate meaningful recommendations. An evaluation of these recommendations based on historical data can also be found here as well as details on GUI implementation.
Chapter 6 describes shortly how the GUI was created and which functions are included. This chapter also shows the most important aspects of the final design.
The outcomes of this research are discussed in the conclusive chapter 7.

Chapter 2

Preliminaries

This research builds on topics from many academic fields of study. The most prominent academic fields in this research are those of process mining and data mining. The difference between process mining and data mining is often hard to formulate because they overlap in multiple ways. They are both centered around data and use similar algorithms and techniques. Process mining is about analyzing process data and differs from data mining on the fact that temporal and sequence-related aspects play a predominent role. Where data mining works with the data in general, process mining is concerned with data about events, which hold the information about the processes they belong to [19]. Although this project does consider the data as process data at first and start from a process mining perspective, the actual techniques used later on in the project fit better in the data mining perspective. Therefore this thesis can not be placed in one or the other perspective. Since it definitely uses aspects from data mining and statistics as well, it can better be described as a research in the field of data science, which combines elements from process mining, data mining and other related fields of study into a broader perspective. [18] defines data science as follows:

"Data science is an interdisciplinary field aiming to turn data into real value." ... "Value may be provided in the form of predictions, automated decisions, models learned from data, or any type of data visualization delivering insights. Data science includes data extraction, data preparation, data exploration, data transformation, storage and retrieval, computing infrastructures, various types of mining and learning, presentation of explanations and predictions, and the exploitation of results taking into account ethical, social, legal, and business aspects."

From this definition it becomes clear that data science is not just mining the data for models, but also taking into account all transformations both data and intermediate outputs might undergo, before actually bringing value to the business. Van der Aalst [2] names four questions data science can help answer:

1. What happened? (Reporting)
2. Why did it happen? (Diagnosis)
3. What will happen? (Prediction)
4. What is the best that can happen? (Recommendation)

This research aims to provide means to answer the last two questions in this list. It can be categorized in the direction of *prediction* and *recommendation* since the main goal of this research is to develop systems than can make predictions and/or recommendations from process data. But, in order to actually start answering these questions, the process data needs to be understood, explored and transformed first. Before this can be done, a deeper understanding of certain concepts in data science is needed.

This chapter will discuss the elements of data science which are preliminary to the rest of this research. Together, these preliminaries will provide in understanding the concepts introduced in this thesis.

2.1 Process Mining Principles

This section describes the basics of the principles which originate from the field of process mining. Process mining as defined in the Process Mining Manifesto is the set of techniques, tools, and methods to discover, monitor and improve real processes (i.e., not assumed processes) by extracting knowledge from event logs commonly available in today's (information) systems [21]. *Event logs* contain the data belonging to at least one process instance or *case*. Each activity that takes place in a process instance is recorded into a database. One such activity is called an *event record*, which represents the execution of a specific activity in a business process [17]. A trace is the sequence of all stored event records for one specific case. The concepts of cases, traces, events and event logs are explained below.

2.1.1 Events

A *case* is a process instance, one execution of a business process. For example, one handling of a mortgage at a bank for a certain customer is one case of a mortgage request handling process. Each activity in a case is recorded as an *event*. These events contain information about the activity in the form of attributes. For example, in the mortgage process, the application of the mortgage request at a bank employee is such an event. Teinemaa et al. [17] name three attributes that are always stored in an event record. These are:

1. The *event class* (or *activity name*) of the event, to identify what activity has been executed. For example, in case of the application of a mortgage the label would be 'Application'.
2. The *timestamp* of the activity, i.e. the date and time the activity is executed.
3. The *case id* or *unique id* of the process instance the activity belongs to. For example, when an application for a mortgage is requested at the bank, a process instance starts and the customer is assigned a case number which functions as the case id.

Through these three attributes a representation of a process as a sequence of events over time can be constructed. Next to the attributes introduced above, event records can also contain a number of other attributes which contain information either related to the case the event belongs to (*case attributes*) or the event itself (*event attributes*). Case attributes are static attributes that contain information about the case that does not change over time. For example, in the mortgage handling process at a bank such an attribute would be the amount of money requested within the mortgage or the annual income of the customer requesting the mortgage. Event-specific attributes are directly related to the specific activity in the event record. For example: when an order is taken in a restaurant ordering process,the type of drink ordered would be an event attribute. Event-specific attributes change from event to event, and thus are dynamic in nature.

All the attributes related to the event form the definition of an event [17]:

Definition 2.1. ***(Event)*** *An event e is a tuple $(a, u, t, (d_1, v_1), .., (d_m, v_m))$ where $\lambda_A(e), \lambda_U(e), \lambda_T(e)$ are functions that return the activity label a, the unique identifier u of the case the event belongs to and the timestamp t of an event. The tuples $(d_1, v_1), .., (d_m, v_m)$ where $m1$ are all combinations of names and values belonging to either an event attribute or case attribute.*

All events belong to the universe of events $\mathcal{E}$.

2.1.2 Traces

A process instance can be described as a sequence of events belonging to that instance, or a series of activities belonging to one specific case which have happened in direct succession of each other. This sequence of events generated by a process instance or case is called a trace [17]:

Definition 2.2. ***(Trace)*** *A trace is a non-empty sequence $\sigma = \langle e_1, .., e_n \rangle \in \mathcal{E}^*$ of events, such that $\forall e_i, e_j \in \sigma : \lambda_U(e_i) = \lambda_U(e_j$, which implies that all events in a trace share the same unique identifier, thus are part of the same case.*

The set of all possible sequences within an event log is $\mathcal{E}^*$. The set of traces is called an *event log.*

A sequence can be extended with additional events whenever a new activity is executed within a process instance. This can be defined as the concatenation of an element and a sequence:

Definition 2.3. ***(Concatenation)*** *Given a sequence $\langle e_1, .., e_n \rangle$ and an element e_x the concatenation of the sequence and the element $\langle e_1, .., e_n \rangle \oplus e_x = \langle e_1, .., e_n, e_x \rangle$.*

The focus of this research is on the outcome of processes. Each completed process instance has such an outcome, and therefore also each trace has a such an outcome. *Key performance indicators* express whether a trace outcome is satisfactory from a business point of view. A definition of a KPI is given below [13]:

Definition 2.4. ***(Key Performance Indicator)*** *Let L be an event log. Let U be the set of possible values for a key performance indicator. A key performance indicator is a pair (k, K) consisting of a function $k : L \to U$ that assigns a KPI value $k(\sigma)$ to each trace σ and of a set $K \subset U$ that contains the KPI values that are satisfactory from a business point of view.*

Note that in this definition it is only known whether an outcome is satisfactory or not, which is a binary decision. Definition 2.6 defines another solution for process outcome, which allows for tertiary and higher dimensions of outcomes.

2.1.3 (Outcome-Oriented) Predictive Process Monitoring

When a value $k(\sigma)$ is known, set K can be used to evaluate if a case was either successfully or unsuccessfully completed. By making predictions on these KPI values the likelihood of a successful case completion can be predicted as well. These predictions can be placed into the field of *business process monitoring.* Business process monitoring is the act of analyzing events produced by the executions of a business process at runtime, in order to understand its performance and its conformance with respect to a set of business goals [6]. When making predictions of the future state of a process at runtime, the performance of the process at runtime can be addressed. If a process is likely to complete with an unsatisfactory outcome, action is needed. When this prediction technnique is used for process monitoring, it is called *predictive process monitoring.* Predictive (business) process monitoring techniques go beyond traditional business process monitoring techniques by making predictions about the *future state* of the executions of a business process [17]. When this future state is the *final state* (or outcome) of a case this monitoring is called *outcome-oriented predictive process monitoring.* This thesis falls directly into this field.
In order to be able to make predictions, techniques from the field of data mining are needed. This shows how process mining aspects as described above can be directly related to aspects in data mining. Principles from data mining used in this thesis are described in the next section.

2.2 Data Mining Principles

Data mining is described in [3] as "the analysis of (often large) observational data sets to find unsuspected relationships and to summarize the data in novel ways that are both understandable and useful to the data owner". In data mining, the input data is typically given as a table and

the output may be rules, clusters, tree structures, graphs, equations or patterns [18]. The input data in this definition might as well be process data, if necessary in a transformed form.
To get to these outputs from the original data, machine learning is one of the techniques that can be applied. Machine learning is often mistaken as a synonym for, but is merely one of the techniques used in or in addition to data mining.

2.2.1 Machine Learning: Supervised vs. Unsupervised

Machine learning can be divided into *supervised* and *unsupervised* learning. Supervised learning assumes labeled data, i.e., there is a response or dependent variable that labels each instance [18]. A common technique in supervised learning is *predictive modelling*, in which a predictive model is learned from the data. Predictive models have the specific aim of allowing us to predict the unknown value of a variable of interest given known values of other variable [3]. These models can predict categorical variables (*classification*) or continuous variables (*regression*).
In unsupervised learning a labeling is absent, so it's main focus is discovery. In other words, unsupervised learning aims to gain new insights from the data. One of the techniques in unsupervised learning is clustering, which examines the data to find groups of instances that are similar [18].

2.2.2 Machine Learning in Predictive Process Monitoring

Recall that in outcome-oriented predictive process monitoring, the outcomes of unfinished cases are being predicted. These unfinished cases are represented as traces of an incomplete process, or a partial trace. Such partial traces are called *prefixes*. A prefix of a trace is defined as [17]:

Definition 2.5. ***(Prefix)*** *Let σ be a trace with events $\langle e_1, .., e_n \rangle$. Let l be a integer $0 \leq l \leq n$. Then the prefix of length l is denoted as $prefix(\sigma, l) = \langle e_1, .., e_l \rangle$.*

In predictive process monitoring the outcome of a prefix is to be predicted. This outcome can be defined as the outcome value of a trace generated from the KPI function $k(\sigma)$. Recall that K is the set of all positive KPI outcomes. When $k(\sigma) \in K$ than the outcome of the process is a success. Whether a process instance ends with a success, is what actually needs to be predicted. Therefore, *class labels* are used. A class label expresses the outcome of a process according to a business goal [17]. So it expresses whether the business goal is met with respect to a KPI, i.e., it expresses if the outcome of a process was positive or negative. Note that this is not always a binary situation, as mentioned earlier. E.g., an outcome can also be indecisive. It depends on the specification of the business goals. When a set of traces and their outcome labels are known, e.g. by checking if $k(\sigma) \in K$, the following can be defined:

Definition 2.6. ***(Labeling Function)*** *Let L be an event log. A labeling function $y : L \rightarrow \mathcal{Y}$ is a function that maps a trace $\sigma \in L$ to its class label $y(\sigma) \in \mathcal{Y}$ with $\mathcal{Y}$ being the domain of the class labels. When predicting the outcomes, $\mathcal{Y}$ is a finite set of categorical outcomes. For a binary outcome $\mathcal{Y} = \{0, 1\}$, e.g. 0 implies a failure, and 1 a success.*

With the prefixes and labels of unfinished traces known, machine learning can be used to train a model than can predict the outcome class label from a prefix. Because the data here is labeled, this is a supervised machine learning problem. A classification algorithm can be used to create a predictive model. Such algorithms use a vector of variables, called *features*, to train a model. In order to translate a sequence of events into a vector, a certain encoding is needed [17]:

Definition 2.7. ***(Sequence Encoder)*** *A sequence encoder $f : L \rightarrow \mathcal{X}_1 \times ... \times \mathcal{X}_p$ is a function that takes a (partial) trace σ and transforms it into a feature vector in the p-dimensional space $\mathcal{X}_1 \times ... \times \mathcal{X}_p$ with $\mathcal{X}_j \subseteq \mathbb{R}$, $1 \leq j \leq p$ being the domain of the j-th feature.*

Some features in the vector represent the sequence of activities. The features can represent the order of the activities in the sequence or the count of each activity in the sequence. These are *control-flow features*. Some features in the feature vector correspond to case and event attributes.

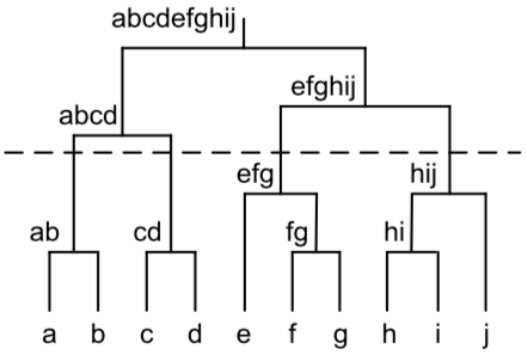

Figure 2.1: A dendogram representing a hierarchial clustering, from [18].

These can be the values of the attributes at the last event in the prefix, but they might also be a cumulative value of all the events up to the last event. How this encoding is applied and to which events and attributes is a design decision made by the data scientist/researcher that is working with the data. Many alternatives might have to be tried in order to end up with the best suitable encoding for the data that is available. Business knowledge of the process is often needed to make these decisions.

When a proper encoder has been set a classifier is trained through machine learning. A classifier can be defined as [17]:

Definition 2.8. ***(Classifier)*** *A classifier* $cls = \mathcal{X}_1 \times ... \times \mathcal{X}_p \rightarrow \mathcal{Y}$ *is a function that takes an encoded sequence and estimates its class label.*

Once a classifier is trained, the classifier can be applied to the traces of running processes in order to determine if a process is likely to succeed with respect to the business goal. By doing this a process manager can monitor how the process is doing and whether action is needed to steer a process towards a more favorable outcome.

2.2.3 Agglomerative Hierarchial Clustering

A common practice in machine learning is clustering, which is the task of partitioning a dataset into groups, called clusters [14]. The goal is to split up the data in such a way that points within a single cluster are very similar and points in different clusters are different. Clustering algorithms assign a number to each data point, indicating which cluster a particular point belongs to, similar to a classifier as mentioned above. A very straightforward clustering technique is hierarchial clustering. This creates a hierarchy of clusters, where each cluster is part of a higher order cluster. A common type of hierarchial clustering is *agglomerative hierarchial clustering*, where each instance starts in its own singleton cluster. Then pairs of clusters are formed by searching for the nearest cluster. The joined cluster is then moved up in the hierarchy. This is repeated until all instances are in the same cluster [18]. The result in a graphical representation in the form of a tree shaped dendogram. A dendogram is shown in figure 2.1.
Figure 2.1 also shows a horizontal line. This line represents the chosen level of clustering. Moving this line up and down in the dendogram results in larger or smaller clusters. Finding the optimal level can be found by applying internal clustering measures, which validate the quality of clustering. To optimize a hierarchial clustering, the Silhouette Score and the Calinski-Harabasz index(CH) can be used. The Silhouette Score validates the clustering performance based on the pairwise difference of between- and within-cluster distances [11]. The Calinski-Harabasz index evaluates the cluster validity based on the average between- and within-cluster sum of squares. By varying the level of a hierarchial clustering up to the point where both the Silhouette Score and CH-index are maximized the optimal level can be found.

Chapter 3

Initial Situation at UWV

3.1 The WW Reintegration Process

Removed for confidentiality reasons.

3.2 The Data

Removed for confidentiality reasons.

Chapter 4

Prediction of KPI Outcomes

Now the context of the reintegration process at UWV is clear and the process data is prepared, a predictive model for KPI's can be generated. In chapter 2 outcome-oriented predictive process monitoring has been introduced. In order to get to a predictive model that is usable in practice, there is a number of steps that is usually taken in predictive process monitoring. This Chapter defines a methodology which defines the steps that are used to create a predictive model and how it is used in the monitoring of the reintegration process at UWV. To start, Section 4.1 discusses (Outcome-Oriented) Predictive Process Monitoring in more detail. The state of the art in academic research is discussed here. It also introduces a baseline framework for outcome oriented predictive process monitoring as introduced by Teinemaa et al. [17]. This framework is the baseline of the methodology created and presented in this research. Section 4.2 builds upon this framework and discusses the design of the methodology. Then the implementation of the methodology using the UWV data is discussed in Section 4.3. Conclusively, an evaluation of the methodology using the UWV process data is given in Section 4.4.

4.1 State of the Art

The last couple of years have seen an increase in studies related to predictive process monitoring. Interestingly, each research has its own view on predictive process monitoring (PPM). In related work there are differences in the type of predictions that are made, as well as differences in the techniques that have been used to get to these predictions such as groupings of traces, encodings of traces and prediction techniques. With respect to the types of prediction, Tax et al. [16] define three groups in PPM: 1) case outcome predictions, 2) time-related predictions and 3) prediction of continuation of a case and/or related characteristics. Examples of *time related predictions* are delays, deadline violations or prediction of the remaining life cycle of a running case. Examples of continuation predictions are predictions of the next event in a trace and/or its attributes. This thesis focuses on the prediction of case outcomes only, therefore literature related to only outcome-oriented predictive process monitoring is discussed further.
According to Tax et al. the goal of *outcome-oriented* approaches is to predict cases that will end up in an undesirable state. Senderovich et al. [15] define Outcome Oriented Predictive Process Monitoring as:

Definition 4.1. *Given a (possibly) running case σ_x, the predictive process monitoring problem is to find a function $f : \mathcal{E}^* \to \mathcal{Y}$ that accurately maps σ_x to the corresponding outcome label $y(\sigma_x)$.*

Note that the function introduced in Definition 4.1 is similar to the function for the labeling function in Definition 2.6 as introduced in the preliminaries of this research. The two definitions differ since Definition 4.1 defines PPM as the process of finding a labeling function, where Definition 2.6 defines the function itself. Different papers address the predictive process monitoring problem but each of them uses a slightly different approach to solving it. Teinemaa et al. [17] have done a literature review on outcome oriented predictions in the field and have identified the most important

steps in PPM. The first step in solving this problem is identified as *prefix extraction* and filtering. In most studies prefixes are extracted directly from the event log as partial traces, but different approaches to generate the partial traces are taken. Some studies only consider the first certain number of events of a trace. For example, prefixes with *fixed-lengths* up to respectively 20 and 21 events are used in the studies by Leontjeva et al. [10] and Di Francescomarino et al. [5]. Another approach is used in Di Francescomarino et al. [8]. Prefixes of multiple lengths are extracted based on a certain *gap* between events. The resulting prefixes all have a length which is equal to a multiple of numbers within a gap, e.g. when the gap is 5 events, the prefix lengths will be 5, 10, 15 and so on. According to [17] this approach is especially suitable when smaller prefix logs are needed for example when calculations need to be efficient, or when dealing with already large event logs.
Prefix extraction is often followed by *trace bucketing* as concluded by Teinemaa et al. [17]. In trace bucketing groups of prefixes, called *buckets*, are created. This is done in approaches where multiple classifiers are trained. While making a prediction on a running case, the case is assigned to a bucket and then the appropriate classifier is used to make the prediction. In a single bucket approach all prefixes are stored in a single bucket and only one classifier is trained. This approach is suitable when process data is regularly refreshed and classifiers are trained on a regular basis. Another situation would be when large numbers of running cases are predicted at the same time such that applying a single classifier to all of them at once is more efficient. A popular approach to creating multiple buckets is clustering. Maggi et al. [12] use K-nearest clustering to divide the prefixes into buckets based on how similar they are. Interestingly, in this approach the bucketing is done in the online phase, and the offline phase is skipped entirely. The bucket with prefixes most similar to a running case is identified after which the classifier is trained and the case outcome is predicted. The downside of such an online approach is that it can be very time consuming when making predictions for large sets of running cases, since for each prediction a classifier needs to be trained. Di Francescomarino et al. [8] and Verenich et al. [20] experiment with different clustering approaches in the offline phase. Another approach to bucketing is using buckets of prefixes of equal lengths, often resulting in many buckets, hence many classifiers that need to be trained, as seen in the study by Leontjeva et al. [10]. Verenich et al. [20] combines *prefix-length* bucketing with clustering and applies clustering after prefix-length buckets have been created, resulting in even more buckets. Sometimes buckets are created by simply using *domain knowledge* to make the distinction [17]. This is often done when different scenarios of a process need to be analyzed, such as different process contexts or prefixes that are in different execution stages of the process. Another approach is the *state*-based approach introduced by Lakshmanen et al. [9], where important states are derived from a process model, e.g. a transition system or Petri net. These states are often important decision points in a process, after which the process moves into a certain direction. Prefixes can be considered to be at a certain decision point based on the sequence of events that have occurred, and are therefore belonging to a certain bucket. A classifier is then trained for each of these buckets. This approach is suitable when a proper process model is available or can be generated from the event log through process mining.
After trace bucketing the next step in PPM is the encoding of the prefixes in each bucket into a feature vector(*trace encoding* or *sequence encoding*, see chapter 2). Again, multiple approaches are used to realize this in relevant literature. As shown in chapter 2 a trace consists of case attributes and event attributes. A different encoding is needed for event attributes since these are dynamic in nature while the case attributes are static and do not change with each event execution. The case attributes can therefore just be added to the feature vector without the loss of information [17]. A way to deal with the event attributes is to use *last state encoding*, which takes a snapshot from the data at the last available moment, i.e. the values of attributes as stored with the last event in the prefix are included in the feature vector. A requirement of this technique is that the feature vector includes all possible event attributes. Last state encoding is a popular encoding technique as it is used in many studies [8, 9, 12] where De Leoni et al. [4] mention it as one of the trace manipulation solutions they suggest in the paper. A downside of this technique is that everything that has happened prior to the last event is left out. Teinemaa et al. [17] suggest to use the values of the attributes in the last m states as well, making the feature vector increase m times. Another

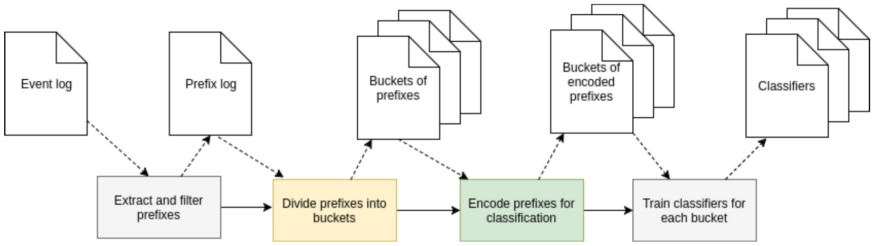

Figure 4.1: The offline predictive process monitoring workflow, from [17].

solution for the drawbacks of last state encoding is to use a different encoding technique called *aggregation*. In aggregation all previous events are considered but the order is neglected, which is also the drawback of this technique. In the aggregation of the execution of events each possible event label is a feature in the feature vector. These features can be assigned an integer value based on the frequency of the event in the prefix(*frequency-based encoding*). Frequency-based encoding has been used in the studies of Di Francescomarino et al. [5] and Leontjeva et al. [10]. Another option is to assign a boolean value to each event label in the feature vector which represents if that event has happened or not(*boolean-based encoding*) [4, 17]. According to Leontjeva et al. [10] frequency-based encoding outperforms boolean-based encoding. Aggregation techniques can also be used on the values of event attributes in the form of using means, minimums, maximums and sums of the values of these attributes throughout the trace [4]. If the order of the events needs to be preserved in the feature vector, a technique called *index-based encoding* can be used. This technique creates a feature in the vector for each event attribute for each event execution in the trace, i.e. for each *index* [17]. A major drawback of this technique is that it can only be used to prefixes with equal lengths, like in fixed-length buckets. Leontjeva et al. [10] did propose this encoding and it is also used in the study of Verenich et al. [20].
When prefixes have been encoded, in most studies on PPM a classifier is trained through machine learning which is then used to predict outcomes on running cases. Different classification algorithms have been used in different studies. The decision tree classifier yields results that are easy to interpret and is used in multiple studies [4, 5, 9, 12]. Another popular algorithm is the *random forest* algorithm, which creates many different decision trees with a random selection of features, hence the randomness in "random forest". The eventual prediction generated by a random forest classifier is the aggregated prediction of all these decision trees. In general random forest have better accuracy than a single decision tree but its results are harder to interpret [17]. Leontjeva et al. [10] have used random forests to make predictions as well as a techniques called gradient boosted regression (GBM) and support vector machines(SVM) but these techniques did not perform as good as random forest classification. Fernandez-Delgado et al. [7] have evaluated a large variety of classifiers on multiple datasets and they conclude that random forest algorithms generally perform best, but not significantly better than support vector machines. Out of the top five tested classification algorithms three are random forest algorithms and two are support vector machines.
When a classifier has been trained it can be used to predict the outcome of a running case. The proper bucket is assigned to a running cases and the classifier trained for that bucket is then used to generate a prediction. A classifier usually returns the outcome value of the prediction itself as well as the probability of that outcome.

The state of the art shows that there are multiple steps and multiple ways of executing these steps in order to create a working predictive process monitoring system. Therefore it is used to decide on the techniques that are going to be used in the design of prediction system as discussed later on in the chapter.

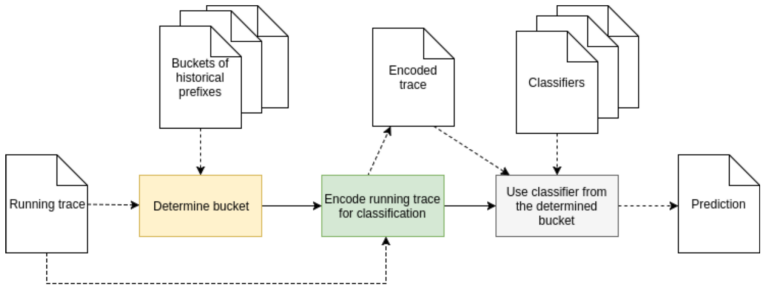

Figure 4.2: The online predictive process monitoring workflow, from [17].

A Framework for Predicitive Proces Monitoring

Teinemaa et al. [17] have analyzed a lot of the techniques mentioned above and identified the most common steps in PPM. These steps are embedded into a graphically represented set of workflows, consisting of both an online and an offline workflow. Figure 4.1 shows the offline workflow in predictive process monitoring according to the framework defined by Teinemaa et al. Prefixes are extracted, grouped into buckets, encoded into feature vectors and eventually a classifier is trained for each of the buckets. In the online phase, a running trace is assigned to one of the buckets created in the offline phase. Then the running trace is encoded such that the already trained classifier for the determined bucket can interpret the prefix. The classifier than makes a prediction. The online phase as defined by Teinemaa et al. is shown in figure 4.2.
From these workflows is is assumed that classifiers are trained offline and then applied an online setting. Online training of classifiers as in [12] can be seen as a combination of the two workflows where a running case is placed into a bucket with similar traces for which a classifier is trained. This framework functions as the baseline of the methodology presented in this thesis.

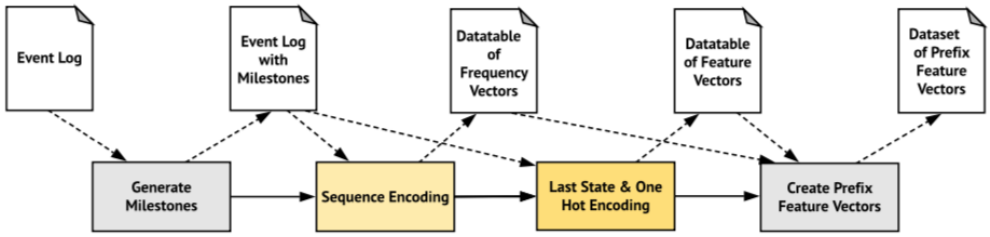

Figure 4.3: The preprocessing workflow as designed in this research.

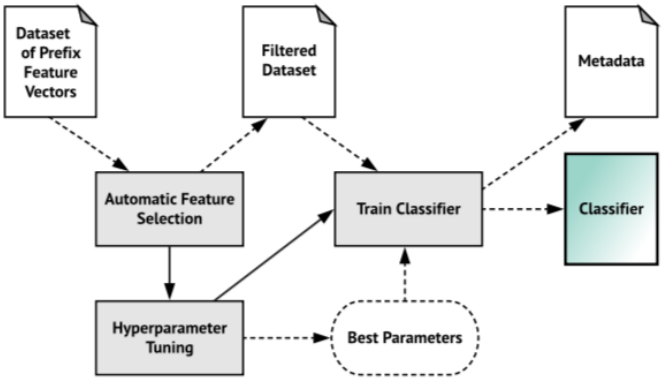

Figure 4.4: The training workflow as designed in this research.

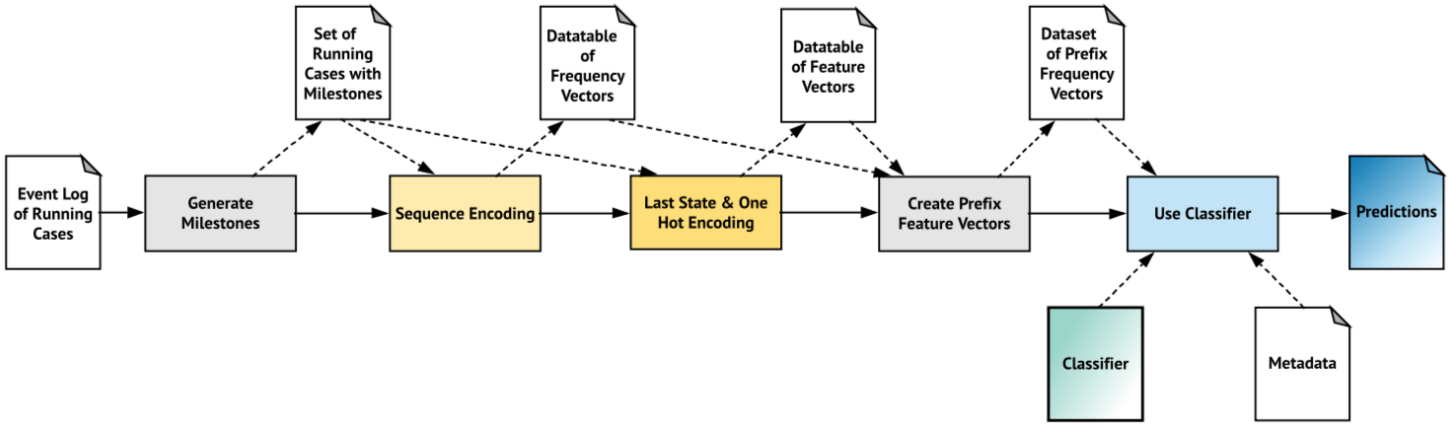

Figure 4.5: The workflow for the prediction phase in PPM as designed in this research.

4.2 Design

The methodology for predictive process monitoring introduced in this research is a derivative of the earlier presented workflow defined by Teinemaa et al. [17]. The offline and online phase for PPM are updated and a separate standardized workflow for preprocessing of the data is introduced. With respect to the workflows introduced in [17] the offline workflow is split up into a preprocessing workflow and a training workflow, which both take place in an offline setting. The online workflow is also updated and now includes additional preprocessing steps needed for making predictions. The online workflow is also renamed to prediction workflow. The preprocessing workflow, the training workflow and the prediction workflow are shown in figures 4.3, 4.4 and 4.5.

In the preprocessing phase an event log is transformed into set of feature vectors that represent prefixes of the traces in the log. The preprocessing starts with the generation of milestones. Milestones resemble the length a process with respect to its duration up to some point. When a process has been running for a certain amount of time it has reached a certain milestone. The process reaches a new milestone with the passing of each new month, week or any other time interval in the process. The milestones are generated by calculating the milestone value for each new event in a trace, and storing it as an attribute of the event. Next, these milestone values are then used to easily create prefixes resembling process executions of a certain duration. However, a prefix log, like in the workflow in Figure 4.1, is not created. In the proposed methodology this step is skipped and prefixes are encoded into a feature vectors directly from the event log using the milestone values stored in each event. For each milestone all the events in a trace up to the last event belonging to a certain milestone are selected and immediately encoded into a feature vector. The milestone value of the prefix is an attribute in the feature vector itself. The result is a prefix feature vector (or encoded prefix) for each milestone the process has reached. Doing this for each trace in the log results in a set of prefix feature vectors that is used to train the classifier. With respect to the workflow in Figure 4.1 buckets of prefixes are not created. A single classifier is trained on a single bucket containing all prefix feature vectors in the event log.
In the training phase automatic feature selection is applied to decrease the number of variables used to train a classifier. Only the best features are selected to be used in training the classifier and the features that are not used are filtered out. Then, on the filtered dataset a hyperparameter tuning is applied to find the optimal parameters for the classification algorithm that is going to be used to train the classifier. The best parameters are then used to train a classifier on the filtered dataset. Along with the classifier, a metadata file is generated that stores which features are used and in which order. This metadata is used in the prediction phase to format new process data in the exact same way such that the classifier can interpret the data to make predictions.
When making predictions with the classifier the encoding of running cases must be the same as for the traces that are used to train the classsifier. Therefore, the milestone values are generated for running cases as well but here a prefix feature vector is created using all the events of a running case. The same encoding as in the preprocessing phase is used. Then the metadata belonging to the trained classifier is used to remove and rearrange the features in the set of prefix feature vectors such that the classifier interprets the vectors properly. Eventually, the classifier is used to make predictions. The result is a predicted process outcome for each running case.

The next sections reason how the original framework by Teinemaa et al. introduced in Section 4.1 is used to create the methodology introduced above. The considerations that have been made with respect to the state of the art in PPM as well as the applicability for the reintegration process at UWV for each of the steps in the original framework are explained.

4.2.1 Preprocessing

Generally, the first step in PPM is the extraction of prefixes from the event log. Definition 2.5 states that a trace is cut off at a certain length l to create a prefix of length l. Since the amount of events per trace is generally quite large in the UWV dataset and likely to be large in other process data as well, cutting prefixes at every possible length creates a very large dataset to work with. This might prove complicated to work with in later steps, especially when dividing prefixes into buckets of prefixes of equal length. For example, training and optimizing a classifier for a large number of buckets can become time consuming. A solution to this problem is based on extracting prefixes over *gaps* between events as described in section 4.1. However, using gaps in the form of m events between the lengths of two consecutive prefixes is not the best way to approach this. The nature of the event execution of the reintegration process at UWV is that a lot of events are repeated. Customers can repeatedly make use of online tools. Each time they use a tool, visit a website or consult a coach this is logged as an event. These events are an indication of how active a customer is in using the reintegration tools made available. On the contrary this is not necessarily an indication of how far into the process the customer is. Therefore using gaps of a

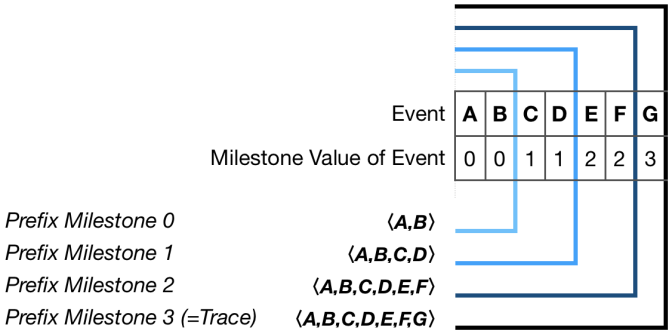

Figure 4.6: Selection of events based on milestone value to generate sequences needed for encoding.

certain length, might not be the best option. E.g. when creating prefixes for each multiple of 10 events, there might be two prefixes extracted from the same case which are one gap of 10 events apart where in reality two hours have passed over these 10 events, while in other cases maybe 10 days have passed for the same gap. When predicting, prefixes of equal length are not similar at all. Next to frequently repeated events customers also have a monthly obligation to communicate to UWV which actions they took with respect to job applications. Examples of these activities are sending a job application or having a job interview at a possible employer. These activities itself are not logged but when the customer communicates his or her activities to UWV, then the communication itself is logged as an event, indicating whether a customer has met the obligation. When using gaps of certain lengths a prefix might contain this activity twice. A solution for these issues might be to use gaps in the form of certain time periods. For example, a prefix is extracted for each time a month has passed in the process. Any other interval can be used, depending on the process characteristics. An advantage of this technique is that the time perspective of the process is better represented in each prefix. In the case of UWV, the amount of time in which certain events have been executed does matter, because it says something about how actively a customer is using and interacting with the reintegration tools provided by UWV. In order to create prefixes for which a multiple of a certain time period has passed, the elapsed time up to an event must be determined. To determine the time that has passed upon the execution of an activity, the timestamps of the event record of activities are used.

Definition 4.2. ***(Trace/Prefix Duration)*** *Let $\sigma = \langle e_1, .., e_n \rangle \in \mathcal{E}^*$ be a trace. The duration of σ is the difference in time between the last event and the first event of the trace: $\lambda_T(e_n) - \lambda_T(e_1)$.*

With the duration of a trace up to event e_x the duration of the prefix up to e_x is also known. This duration can be used to determine intervals in the trace. The points in time where one interval ends and a new one begins are called *milestones*. For each prefix in a trace the duration can be calculated to assign a milestone value to the prefix. A milestone function $m : \mathcal{E}^* \rightarrow \mathbb{N}_0$ maps a sequence to a milestone value. The milestone value is stored as an attribute in the last event belonging to that prefix. As a result each event in the log has a milestone attribute m, i.e. $\lambda_M(e)$. Now that each event has a milestone attribute a prefix for each milestone can be generated. A prefix for a certain milestone includes all the events which have a milestone value equal to and lower than the milestone for which the prefix is created. This is best illustrated by an example. Figure 4.6 shows a trace $\langle A, B, C, D, E, F, G \rangle$. For each event in this trace the assigned milestone is also shown. A prefix is created for each milestone by selecting all events with a milestone value that is equal to or lower than the prefix milestone. The result is a set of prefixes for each possible milestone, including the last milestone. The sequence in the prefix for the last milestone is always equal to the trace itself. Note that the first milestone value is always zero since in the first part of the trace a process has not reached the first milestone yet. A prefix for a milestone m is a simple selection of events in a trace that have a milestone value that is less or equal to m:

$$prefix_{milestone}(\sigma, m) = \sigma\prime$$

such that

$$milestone(\sigma\prime) = m$$

and

$$\exists l : \Big(prefix(\sigma, l) = \sigma\prime\Big) \wedge \Big((l+1 \leq |\sigma|) \Rightarrow milestone(prefix(\sigma, l+1)) > m\Big)$$

The prefixes generated using the technique above are basically still (a copy of) a selection of rows in a datatable where the rows in the selection represent the events that have a milestone value below or equal to the milestone value of the prefix. The next step is to find an encoding that is able to transform these selections of rows into a feature vector.

4.2.2 Trace Bucketing

As discussed in Section 4.1, there are multiple approaches towards the encoding. Many studies use buckets to divide prefixes into groups. For each of these groups a classifier is trained. In [4] a single bucket approach is used. In this approach all prefixes are stored into one single bucket, for which one classifier is trained. In this study this approach is initially tried. Since the quality of the predictive model generated by using one bucket proved already very good, which is shown later in this chapter, the single bucket approach is used throughout the rest of the study as well. Considering the limited time that was available for this research this decision is reasonable.
With respect to the original framework by Teinemaa et al. the bucketing step is removed, and a log of prefixes is also not stored. Instead a set of traces with milestone values for each event is stored. The events can be directly selected from this set for each milestone and encoded directly.

4.2.3 Sequence Encoding

According to the framework prefixes need to be encoded into a feature vector which is interpretable by a classifier. Recall the definition of a sequence encoder in Definition 2.7 which states that an encoding is need to create a vector from a sequence. The most important aspect in choosing an encoding is how to interpret the events and its attributes as a feature. There is a distinction to be made between how to treat the occurrences of the events itself, i.e. what the process looks like up to a certain point, and the attributes of these events. The challenge here is that each trace in a bucket (independent of the number of executed events) should still be represented with the same number of features, i.e. feature vectors of equal length as mentioned in [17]. A solution to this problem is to use an aggregation method for both events and event attributes, which is discussed later in this chapter.

Encoding Events as Frequency Vectors

In another research about predictive process monitoring [8] the frequency of events is used to interpret the execution of a trace or prefix. A vector of event frequencies is used to represent the executed events. For example, let t_1 be a trace (A, B, B, A, C, D, D, E). From this trace an ordered alphabet of events $\langle A, B, C, D, E\rangle$ can be created. Trace t_1 can now be encoded as a vector of frequencies $\langle 2, 2, 1, 2, 1\rangle$. This encoding is achieved by replacing each event in the alphabet with the number of executions of that event in the trace. The alphabet of events should be created based on all the traces in the log since some traces contain different events than others. For example let $t_2 = (A, A, B, C, D, E, F)$ be another trace in the event log which also contains t_1. The alphabet used above can no longer be used to encode trace t_2 as well since event F is not in it. Instead, alphabet $\langle A, B, C, D, E, F\rangle$ should be used instead. In order to create a successful encoding for all events in all traces, all events of the log need to be used to generate an alphabet that suits all traces, and thus all prefixes.
This *aggregation* approach is used in this study instead of an *index* based encoding (see Section 4.1), because the frequency of events is more important than the order of events, taken into account the nature of the reintegration process where some activities are repeated and not necessarily executed in a specific order. In order to create the feature vector for a prefix with milestone value k all

events with a milestone value $m \leq k$ are selected and directly encoded into a feature vector. The trace ID u and milestone value m of the sequence are also in the vector since the combination (u, m) is used as the unique identifier of the vector. The definition of the frequency vector for a sequence σ as it is created in this research is:

Definition 4.3. ***(Frequency Vector)*** *Let $\mathcal{L}$ be an event log. For log $\mathcal{L}$ the ordered list of all unique activity labels in the log is $\mathcal{A} = \bigcup_{\sigma \in \mathcal{L}} \bigcup_{e \in \sigma} \lambda_A(e)$. Let $C_a(\sigma) = \Big|\{e \in \sigma : \lambda_A(e) = a\}\Big|$ be the number of occurrences of the event with activity label $a \in \mathcal{A}$ in σ. Then, $encode\_freqVector(\sigma) = \big(u, milestone(\sigma), C_{a_1}(\sigma), .., C_{a_n}(\sigma)\big)$.*

For each combination of case id (u and milestone value m the rows are selected from the dataset. These rows, representing the events in a prefix, are then transformed in to a frequency vector. As a result a frequency vector is stored in a separate datatable that stores all feature vectors for all prefixes in the dataset.

Last State Encoding of Case Attributes

Now an encoding for the event execution has been designed, both the case and event attributes for each event record in a prefix need to be encoded into features as well. Eventually, these features are combined with the frequency vectors of the prefix. But first a distinction is made between case (static) and event (dynamic) attributes in the event record. Since the index based encoding is not used, the intermediate values of event attributes at a certain index of the prefix can no longer be used. Last state encoding can be used to capture the last state of the attributes. For the static case attributes this approach works well since these attributes can as-is be used as a feature in the feature vector. When using last state encoding for the event attributes a lot of information of prior events is lost. An *aggregation* technique is needed to interpret the values of the other events in the feature vector as well. For these event attributes feature engineering is applied to aggregate the values of these attributes in the prefix. This solution is discussed later on.
The feature vectors that are a result of last-state encoding are uniquely identified by the combination of trace ID and milestone ($u \times m$), exactly like the frequency vectors created earlier so the two can easily be combined later on. When last state encoding is applied to the case attributes in the dataset a case feature vector is created:

Definition 4.4. ***((Case Attribute) Feature Vector)*** *Let $\mathcal{L}$ be an event log. Let $\{c_1, .., c_n\}$ be the set of all case attributes in $\mathcal{L}$, which is the same for each event in the log. Let $\lambda_{C\_last}(\sigma, c)$ be a function that returns the value for a case attribute c in the last event of σ. Let $V_c(\sigma) = \lambda_{C\_last}(\sigma, c)$ be the value of case attribute c in the last state.*
Then, $encode\_caseFeatureVector(\sigma) = \big(u, milestone(\sigma), V_{c_1}(\sigma), .., V_{c_n}(\sigma)\big)$.

These feature vectors for each prefix are created and stored in a similar way as the frequency factors but in a separate datatable. The remaining attributes that need to be encoded are the dynamic event attributes.

Feature Engineering

Concerning the event attributes in the event log of the reintegration process, an aggregation method works well. In chapter 3 the existence of other event attributes next to *activity label*, *timestamp* and *case id* has already been mentioned, but details about them are not yet given. This section discusses these attributes in more detail. The event attributes are related to the communication with the customer and the behavior of the customer within the process. E.g., these attributes store whether a customer has read an email or SMS message send by UWV. The values of these attributes are generally stored with the event record concerned with the occasion of such a message. When predicting the outcome of a process instance using only a snapshot of an event attribute at the last possible state, only the last values of this attribute is taken into account. This ignores the previous values of the attribute in the prefix. When an aggregated attribute is used

instead, a snapshot at the last state would still capture a part of the information that would be lost without aggregation. De Leoni et al. [4] suggest using the average and the sum of these values to aggregate event attributes. When using an aggregation technique, last state encoding can be applied at the same time for both case and event attributes, capturing their values in one feature vector. Carefully analyzing attributes and their values and engineering them into useful features for a classifier is called feature engineering, i.e. feature engineering is the process that aims to represent your data best for a particular machine learning application [14]. In this research feature engineering starts with a careful analysis of the attributes. Next, one or more proper aggregation techniques are selected for each attribute. Then, for each event record the aggregated attribute of all the attribute values of the events that happened before is added to the event record, i.e., for each event that is present in the dataset these features are calculated and stored. The values of these features are cumulative which means that for each new event record these features are recalculated with the latest value of the event attribute taken into account and then added to the event record itself as a new attribute. This way each event record always carries the aggregate value of the values in events that happened before. The result is a feature that is interpretable by both people and classifiers, and makes sense from a business perspective.

In order to ease last state encoding, each event record should carry the last known possible value of each possible aggregated attribute in the trace up to that point, even for the aggregated attributes which are not related to that event. Once the last state of a prefix is captured using the attributes in the last event record, the last possible values for all aggregated event attributes in the prefix end up in the prefix as well. In order to end up with a feature vector with a fixed length for all prefixes, each possible aggregated event attribute is a feature in the vector for a specific prefix even if this attribute is void for that prefix because a specific event is not present in the prefix. This results in *null*-values in some of the feature vectors, which is a drawback of using this approach. An easy fix is to replace $null - values$ in a feature by the median value of the feature in other vectors.
The most interesting aggregate event attributes derived from the UWV dataset are shown in table 4.1.

Table 4.1: A selection of the aggregated event attributes in the dataset after feature engineering.

Removed for confidentiality reasons.

Last State Encoding of (Aggregated) Event Attributes

For the dataset of the reintegration process, the aggregated values are calculated and stored in each event record. Then the original event attributes are removed, except for the *case id* and *milestone* since these are needed to uniquely identify the resulting vector later. Now the aggregated event attributes are stored in each event record, and the independent event attributes are removed from the dataset a snapshot can be made at any intermediate point in time. This allows for a last state encoding with less information loss.

While using aggregation, last state encoding can be applied to event attributes as well. This means that the values for both the case and the event attributes can be grapped at the same time, resulting in a feature vector for all attributes. The definition of the encoding for case attributes can be extended to feature the aggregated event attributes as well:

Definition 4.5. ***(Feature Vector)*** *Let $\mathcal{L}$ be an event log. Let $\{v_1, .., v_n\}$ be the set of all attributes in $\mathcal{L}$, which is the same for each event in the log. Let $\lambda_{V\_last}(\sigma, v)$ be a function that returns the value for an attribute v in the last event of σ. Let $V_v(\sigma) = \lambda_{V\_last}(\sigma, v)$ be the value of case attribute v in the last state.*
Then, $encode\_caseFeatureVector(\sigma) = \big(u, milestone(\sigma), V_{v_1}(\sigma), .., V_{v_n}(\sigma)\big)$.

A feature vector is created for each determined prefix in the event log and stored in a datatable.

Combining the Vectors

Two datatables of vectors remain. One containing the frequency vectors, the other containing the feature vectors for the case and event attributes. To create a final feature vector, the datatables are combined through a natural join on the unique identifier (u, m) of the vectors(or rows) in both datatables. The natural join on the unique identifier makes sure the vectors are correctly joined, hence, there are no duplicate features in the final feature vector datatable.

One Hot Encoding of Categorical Attributes

Since the classifier algorithm is only able to interpret numerical variables, the categorical attributes remaining in the last state also need to be encoded into numerical attributes. A technique called *one hot encoding* is needed to do this [14, 17]. Let m be the number of possible levels of a categorical attribute in the dataset. The categorical attribute is transformed into a bitvector $(v_1, .., v_m)$ where $v_i = 1$ if the given value is equal to the i-th level of the attribute, and $v_i = 0$ otherwise. For example, if a categorical attribute in the last state of a prefix has value A out of possible values A, B, C, the feature A in the resulting feature vector for that prefix has value 1,i.e. $A = True$, where the other features related tot that categorical attribute have value 0, i.e. $B = False$ and $C = False$. One hot encoding is generally applied at the very last moment, creating features only for the possible levels of categorical attributes that have ended up in the feature vector.

Prefix Feature Vectors

The result of all the steps discussed above is one preprocessed dataset in the form of a datatable where each row is feature vector for a prefix. In the remainder of this thesis these *prefix feature vectors* are mentioned and used a lot. Therefore the following definition is introduced to formulate the prefix feature vector, or *prefix* for short from here on.

Definition 4.6. ***(Prefix Feature Vectors)*** *A prefix feature vector $\sigma\prime$ is a tuple $(u, m, a_1, .., a_n, cat_{1,1}, .., cat_{k,l}, num_1, .., num_p, o)$ where u is the case id of the prefix, m is the milestone the prefix has reached, $(a_1, .., a_n)$ are the frequencies of activity a_1 up to and including a_n in the prefix. $cat_{1,1}, .., cat_{k,l}$ are the boolean values of the one hot encoded categorical attributes of the prefix, where $cat_1, .., cat_k$ are the different categorical attributes in the dataset, and indices 1-l indicate the possible values cat_i can hold. $num_1, .., num_p$ are the values of the numerical attributes at the last event of the prefix. o is the KPI outcome of the original trace.*

The end product is going to be used in training the classifier. This completes the workflow as shown in Figure 4.3.
The next step in the methodology is choosing the best classifier and training a model.

4.2.4 Training

The second workflow in the introduced methodology is the *training*-workflow. In the original workflow by Teinemaa et al. [17], a classifier is trained for each bucket of prefixes and this training is part of the online phase in predictive process monitoring. Training a predictive model that can accurately predict a process outcome is not as straightforward as shown in the original workflow. A number of steps need to be executed before a training algorithm can be applied to train a predictive model. The first step is to select the best features to use in training the classifier. Reducing the number of features reduces the required time to train a model and it can even increase prediction accuracy. The risk of overfitting is also reduced when using less features [14]. After the best features are selected, an algorithm needs to be chosen that suits the data best. Through evaluation of the different options with respect to training time and prediction accuracy, a classifier can be chosen that suits the data best. Tuning the hyperparameters of this algorithm is also important, which is the third step. Since decisions on how to handle feature selection, model selection and hyperparameter tuning are related to the type of data is used.

(Automatic) Feature Selection

Training a model on datasets with high dimensionality makes models more complex which increases the risk of overfitting [14]. The UWV dataset, for example, has a high number of features, and thus a high dimensionality. Therefore it can be a good idea to decrease the number of features, since it reduces a model's complexity and allows for models to generalize better. Identifying the features that can predict an outcome well is hard to do manually. Luckily, there are techniques in machine learning that can help identify the best features or even automatically select the best features. Müller and Guido [14] propose three techniques for automatic feature selection:

1. Univariate Statistics: select best features based on statistics.
2. Model Training: Train a model, derive feature importances from the model, and select the best features based on these importances.
3. Train a series of models: Recursively train a model to identify the best features.

The first option proposed by Müller & Guido is the use of univariate statistics to find the features which have a statistically significant relationship to the target. The second approach is to train a model on the data and then derive the most important features from the model afterwards. Machine learning libraries for classification algorithms often have a build-in method that can return the feature importances in the form of scores. This technique requires to set a scoring threshold prior to using the method to select the best features, which is often hard to interpret and chosen quickly. Instead of a threshold, a percentage of features or a fixed amount of features can be used to make a final selection, e.g. select only the best k features. A third technique is to recursively train a series of models with a varying number of features to identify most important features. Percentage and amount of features can be used here as well to make a final selection. The models do not need to be the model that is eventually going to be used in training the final

model. Which method that is used to determine the features to be used in the model depends on the data that is used.

Model Selection & Hyperparameter Tuning

Each time a prediction model needs to be created for a process, the most suitable classifier needs to be determined. The decision tree classifier (DT), the random forest classifier (RF) and the support vector machine classifier (SVC) have been used in other studies (see Section 4.1. As shown in section 4.1 the RF and SVC are algorithms that have generated reliable models in the past, where RF performs slightly better than the SVC algorithm in some cases. To choose an algorithm for a predictive process monitoring problem, different algorithms can be compared with respect to training times, prediction accuracy and area-under-the-ROC-curve to see which algorithm performs best. When determining a classifier to use, we suggest to test at least the RF and SVC algorithms since these have proven to perform well. The decision tree algorithm can also used, since it has been succesfully used in other studies (Section 4.1) and its models are easy to interpret. It also has lower training times than RF and especially SVC.

When a classification algorithm is chosen, the parameters of the algorithm can be tuned to find the parameter settings that result in the best possible model. Each combination of parameters related to an algorithm can be tested, a technique that is called a *grid search*. How to set up a grid search and which parameters to test is dependent on the data that is used.

Once the algorithm and corresponding parameters have been chosen a final classifier can be trained.

Training the Final Model

The dataset of prefix feature vectors generated in the preprocessing phase is a datatable where each row is a prefix feature vector and each column can be seen as a feature in the vector. In the feature selection step the best features are selected. From the datatable the features that are not going to be used can be removed. The resulting datatable is used to train the classifier. It is important to know which features are used, so when using the classifier to predict outcomes in the prediction phase, the same features can be selected from the set of preprocessed traces for running cases. Therefore, a metadata file is stored with the classifier which holds the list of features that is used in training. The order of the features does matter.

4.2.5 Prediction

The prediction workflow consists of the same steps as in the preprocessing workflow, but is extended with one more step in which the actual predictions are made. First, a set of prefix feature vectors is created from the set of running cases. When making predictions we only care about the prediction of the entire trace so only one prefix feature vector is created for each case. The same encoding is used as in the preprocessing phase. Once a set of prefix feature vectors is created, the metadata belonging to the classifier is used to format the vectors. Next, the classifier is applied to make predictions for each of the cases.

4.3 Implementation

Removed for confidentiality reasons.

Table 4.2: Parameters and statistics for the created random forest classifier.

Number of Estimators	Max Features	AUC	Accuracy	Precision	Recall	F1
1000	4	0.8145	0.8775	0.81	0.82	0.82

4.4 Evaluation of the Entire Methodology

Algorithms have been created in Python to incorporate the workflow introduced above and some data cleansing functions such as the removal and imputation of null-values have been applied. In order to evaluate the entire methodology for predictive process modeling a final model needs to be trained using the available UWV datasets as described in the previous sector. Training is done on the train set. Recall that the test consists of a set of completed traces. Therefore, testing is done on the the test set where we randomly remove a certain number of events from the end of the trace to simulate that the process execution is not yet completed. A grid search is applied on the train set for a random forest classifier while using the grid of parameters introduced above. The best parameters from the grid search are used to train a final model on all data in the train set. The traces in the test set are processed into prefixes up to and for each milestone they contain, just like the preprocessing in the train set. Then one randomly chosen prefix for each trace in the test set is selected. Predictions are made on this preprocessed test set using the classifier trained on the train set. Consecutively, a number of scoring metrics is calculated for these predictions. The results are shown in Table 4.2.
From these results it becomes clear that the classifier performs quite good when applied on a separate dataset. This implies it is not overfitting on the data that is used to train the model, and it is capable of generalizing towards other data. According to the prediction 24.8% of the traces will have a negative outcome. The actual amount of traces with a negative outcome is 27.11% which is very close to the predicted outcome.

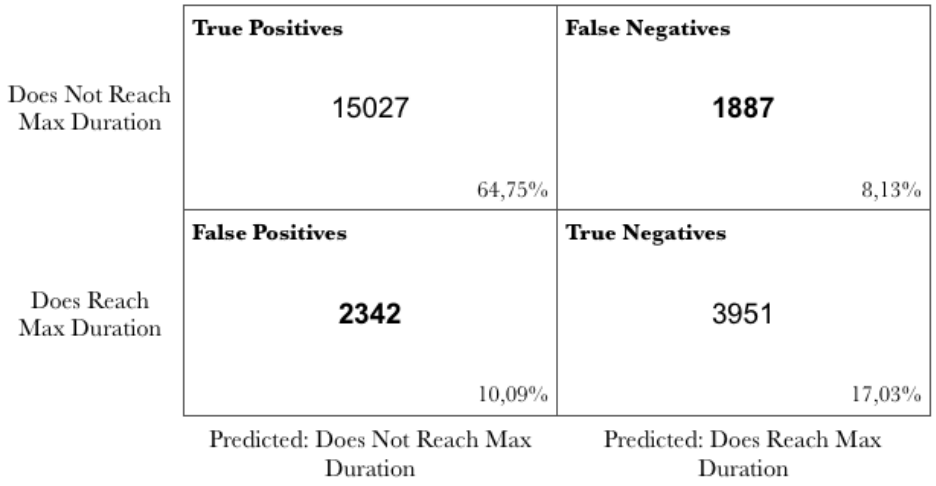

Figure 4.7: The confusion matrix for the predictions generated by the classifier.

Another way to assess the quality of the prediction model is to create a confusion matrix, as shown in Figure 4.7. The most important statistics here are the False Negatives and the False Positives. A false negative implies that a negative outcome is predicted, where the actual outcome is positive. For 8.13% of the traces a false negative is predicted. This is not a big problem in the case of UWV since some cases might be given extra attention where none is required. A false positive implies that a positive outcome is predicted where the outcome of the process is actually negative. This is problematic for UWV, since these cases might require extra attention but according to the prediction this is not necessary. In 10% of the cases a false positive is predicted. This indicates that the classifier is able to predict outcomes quite accurately, but definitely needs some improvement. The probability of a negative outcome can be used to determine how risky a case is. A solution could be to target cases which have a positive prediction and a not very high probability of a positive outcome as well. Another option is to create a set of classes based on the probability of either a positive or negative outcome. For example, three groups of predictions are created, a group with an evident positive prediction, a group with an evident negative prediction, and a group for which the prediction is doubtful because the probability of the predicted outcome is marginally bigger than the other option. The latter would be the case when the probability of negative outcome is 0.51. Using this approach, groups with a negative prediction or doubtful prediction can be labeled as risky.
The trained classifier is not perfect and needs some improvement. Features can be analyzed, the dataset can be improved and additional settings of the classifier can be tried. Given the time available for this master project, attempts to improve the classifier are not made.

Chapter 5

Recommendation System for Key Performance Indicator Improvement

The previous chapter has shown that process outcomes can be predicted. Process managers want to intervene in a process whenever a KPI prediction is negative. The second research goal of this study is to create a recommendation system that can boost process KPI's. This chapter introduces an extension of the methodology introduced in chapter 4 which can generate recommendations for running cases.

5.1 Concept

A classifier can predict process outcomes based on characteristics of a running case and the event execution of the case. The goal of recommendations is to put forward mitigation actions in order to prevent process outcomes to be negative. As shown in Chapter 2, a running case is represented in a trace. A trace is a sequence of events where each event represents an activity in the process. In case the KPI prediction of a running case is negative, we would like to know which activity to execute next in order to prevent a negative process outcome. A recommendation is an activity that, when executed next in a process, decreases a running process' risk. Since each of these activities is actually modeled as an event in the process and a classifier is available that can map sequences of events to an outcome value, the foundations of a recommendation system is already in place. When predicting process outcomes the classifier interprets a sequence of events to make predictions. When these sequences are concatenated with an activity, as if a new activity is executed in the process, a new prediction can be generated. If the predicted outcome value changes in favor of the KPI, it implies that the activity is likely to have a positive effect. By doing this for each possible activity, the predictions can be compared to identify the most effective ones in a certain case. Consecutively, the best activities can be suggested to process managers. This is the main concept of the recommendation system that is introduced in this chapter. Using the classifier to make predictions for each possible combination of activity and sequence can be challenging and not always necessary. From a business perspective, considering all activities for recommendation might not make sense at all. Some activities are not suitable to use as as recommendation, since they are automated, for example when executed by computers, or executed by actors a process manager has no direct control over. Additionally, some activities can only happen in a specific phase of a process so they cannot be recommended throughout the process but only at a specific moment. To cope with these challenges two distinctions are needed. From the set of all possible activities, the useful activities need to be determined first. These are the activities that can be used as a recommendation in general, as labeled by process managers or business owners. This set

usually is the same for all running cases. From this subset only a number of services are applicable in the state the running case is at, so for each case the applicable services need to be individually determined.
A recommendation system is created which is assessed on UWV's reintegration process. One of the most important types of activities within this process are the services offered by UWV. These services are generally designed to help customers finding a job. They are related to coaching, training, informing and testing a customer. These are the activities within the process that are possible to suggest to customers or process managers, i.e. considered as a recommendation. The recommendations are also tailored to a specific case, since one particular service could help for one customer but not for another.
In the next section a methodology for a recommendations system is introduced. This methodology extends the workflows that are introduced in Chapter 4. The methodology is implemented and evaluated using the UWV dataset later on in this chapter.

5.2 Design

The workflows introduced in Chapter 4 can be extended to incorporate the recommendation system as well. The preprocessing phase (see Figure 4.4) is not changed for the incorporation of a recommendation system. The same feature vectors are used here. Some changes have been made to the training and prediction phase of the workflow to incorporate the generation of recommendations as well. The prediction phase is renamed to *recommendation*. The adjusted workflows for the training and recommendation phase are shown in Figures 5.1 and 5.2 respectively.
In the original training workflow for prediction a classifier is trained using a set of selected features generated by an automated feature selection technique. Recall that features are stored in a prefix feature vector, which among other features, stores the number of times an activity has been executed, or the frequency of an unique event in the trace (see Definition 4.3). In feature selection the best features in the vector are selected and used to train the classifier. This classifier is then used to make predictions for running cases. In the recommendation system however, using only the best features to train the classifier is not sufficient. This classifier is namely also going to be used to assess the effectiveness of an activity for the process outcome, by making predictions on the trace of a running case concatenated with an activity. Therefore, the classifier must not be trained on the set of best features only, but on the union of this set with the set of activities suitable for recommendation. This total set of features can now be used to filter the preprocessed dataset of prefix feature vectors. Then, hyperparameter tuning can be applied before training the final classifier. In addition to the original workflow a transition system is created based on the original event log. The transition system is used to assess which activities are applicable in the state a running case is in. The transition system stores for each possible sequence in the original event log ($\mathcal{E}^*$) what the next possible activities are. So when making recommendations for a running case, the transition system is used to check which activities are common to happen after a sequence of events like the on in the running case. This overcomes the challenge to only use activities that make sense at the point in the process the running case is at. For the creation of the transition the total set of features is also needed since all the possible recommendations need to be accounted for. The workings of the transition system within the methodology is explained in more detail in the next subsections.

In the end of the training phase, the transition system is stored together with the classifier and the metadata.

The recommendation phase is an extension of the prediction phase introduced in the previous chapter. Running cases are preprocessed and consecutively a classifier is applied to make predictions on the running cases. In parallel, the transition system is used to identify the next possible activities for a running case. Then, each of the possible activities is concatenated with the trace to create *hypothetical prefixes*. This concatenation is processed in the prefix feature vector by

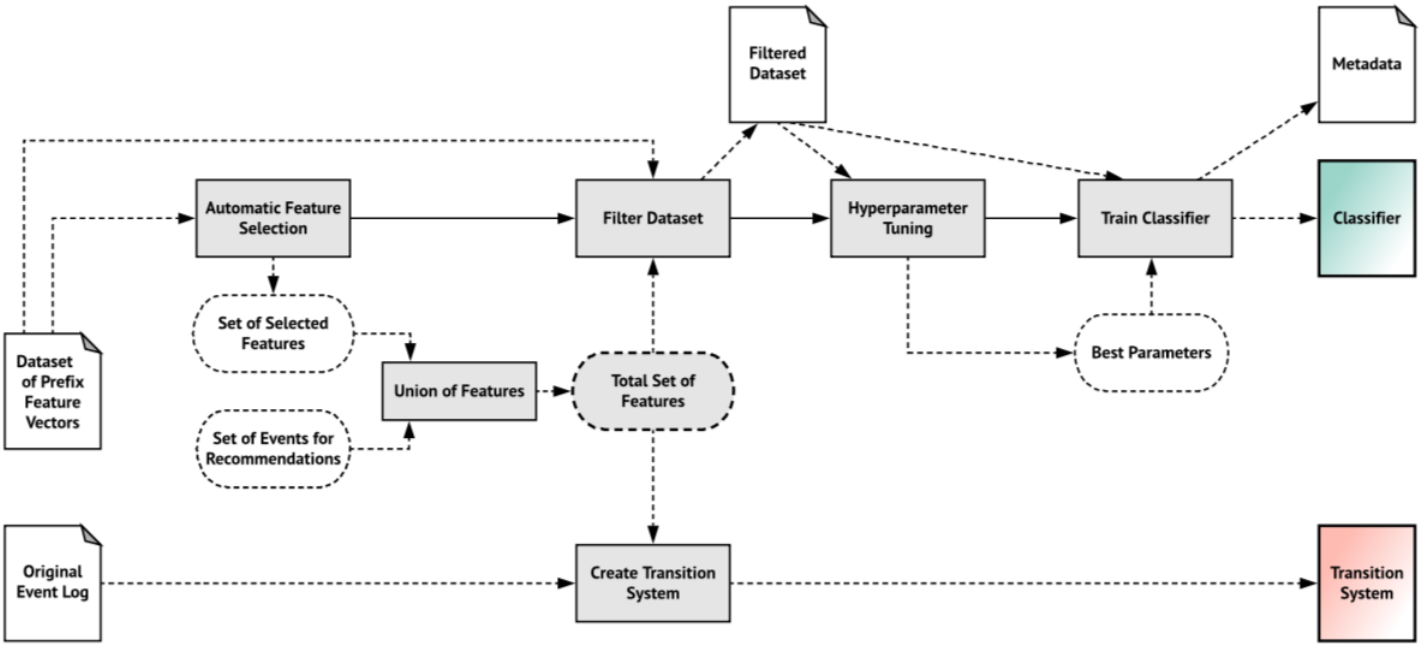

Figure 5.1: The workflow for the training phase in the methodology for creating prediction and recommendations, as designed in this research. Next to the classifier a transition system is also created.

increasing the value in the frequency feature of an activity by one, simulating an extra activity execution. The resulting slightly altered prefix feature vectors represent a running case in the hypothetical situation that an activity (the recommendation) has happened next. So for each running case we have a set of hypothetical prefixes based on the events that can happen next, as determined through the transition system. The hypothetical prefixes for each running case are combined with the ones for the other running cases to create a a large datatable of hypothetical prefix feature vectors. The classifier is now applied to make predictions on the hypothetical vectors. The predictions for these vectors are now compared to the original predictions to identify the best recommendations. In the last step of the workflow all predictions and recommendations for each running case are combined into one datatable. The set of possible activities for recommendations is needed to filter out activities that are suggested by the transition system, which are not suitable for recommendation.
In the next subsection the mechanisms for identifying possible next activities through the use of a transition system are discussed in more detail, followed by a detailed description of how the recommendations are eventually generated.

Identifying Next Activities

From all the possible activities in a process the activities that are possible to suggest for a certain running case need to be determined. For each trace a set of useful and meaningful activities need to be identified. Each running case is in a specific state prior to the generation of recommendations. Once the next possible states are known for a state, the activities that divide these states from the original last state of the trace are the activities that can be used for recommendations. The next possible states can be derived from historical data. The idea behind this is that when an activity y has been witnessed after a similar sequence of activities as the one in the running case, this activity is considered to be common practice and can therefore be executed as the next activity in the process. This can be illustrated by an example. Recall that an activity is represented by an event

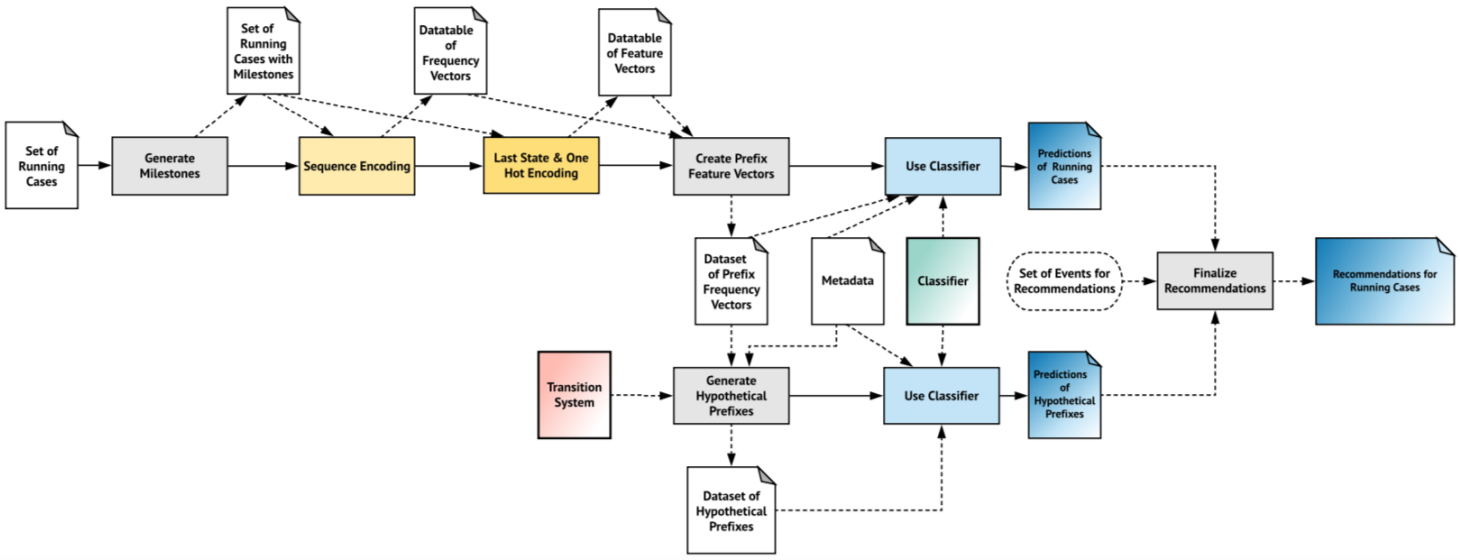

Figure 5.2: The workflow for the online phase in the methodology for creating predictions and recommendations, as designed in this research.

in a trace. Let $\mathcal{L}$ be an event log containing of traces σ_1 and σ_2. Let σ_1 be a complete trace with an event execution sequence of $\langle A, B, C, E\rangle$, and σ_2 be a complete trace with sequence $\langle A, B, D\rangle$. Consider a running case with a sequence of events $\langle A, B\rangle$ for which the best next activity needs to be suggested. Considering the two historical traces, the set of all events in the log initially is (A, B, C, D, E). When looking at traces σ_1 and σ_2, the possible next events can be narrowed down to only C and D and not A, B and E because the latter ones have not been witnessed after A and B. So when considering events for a recommendation, only A and B are taken into account when creating hypothetical prefixes. This is clearly an oversimplified example. The traces and prefixes concerned with the reintegration process at UWV are far more complicated. The number of unique activities that can happen in the process is far greater, resulting in a large number of possible event sequences. These sequences need to be stored, and for each of the stored sequences a set of next events must be stored as well. This can be problematic in complex processes, like in the case of UWV, since it requires a lot of computing power and computation time. Therefore, a technique is needed to simplify these sequences such that the storing of the sequences requires less computation time, as well as the identification of the next events.
A solution to this complexity problem is derived from the definition of a transition system as defined by van der Aalst et al. [18]. Where an event log is a set of traces, which stores the event execution for each individual case, a transition system is an abstraction of an event log. It does not hold detailed information for each trace in the log such as case attributes. Instead it is a process model from which each possible sequence of activities within a process can be derived which makes it very suitable for identifying possible next activities in a process. A transition system is a process model and it can be described by a triplet $TS = (S, A, T)$ where S is the set of states in a process, A is the set of activities and $T \subseteq S \times A \times S$ is the set of transitions. $A = \bigcup_{\sigma \in \mathcal{L}} \bigcup_{e \in \sigma} \lambda_A(e)$ is considered to be the set of all activities (events) in the event log $\mathcal{L}$ where the transition system is based on. A transition connects two states and is labeled with the name of an activity that connects the both states. When a process is in the first state and the activity happens, the process is considered to be in the second state. When creating a transition system from an event log, the most challenging aspect is to find a way to represent the set of states S. In order to determine the set of states, each position in a trace needs to be mapped onto a corresponding state [18], i.e. $\mathcal{E}^* \rightarrow S$. According to van der Aalst [18] a *state representation function* $l^{state}()$ is a function that, given some sequence $\sigma \in \mathcal{E}^*$ and a length $k = |\sigma|$ produces some state. A state is then the set of activities that have

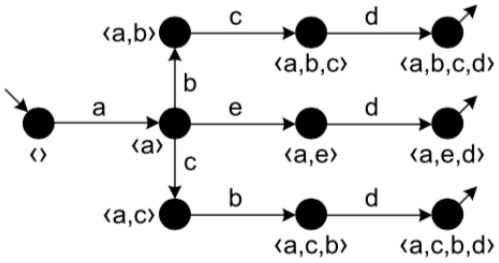

Figure 5.3: A transition system where each state represents the full history of events as a sequence, from [18].

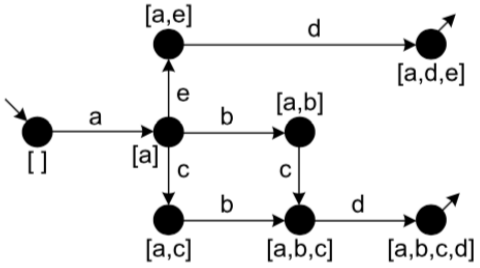

Figure 5.4: A transition system where each state represents the full history of events as a multiset, from [18].

occurred in the first k events.
Van der Aalst defines three types of state representation functions:

- Set
- Multiset
- Sequence

Examples of transition systems that use the sequence and multiset for state representation are shown in figures 5.3 and 5.4. These transition systems are created from the same set of traces.
The sequence abstraction represents a state as the sequence of events in the trace up to that point. In processes with a large variety in activities within the process such as is the case in the reintegration process at UWV, this results in a large number of unique sequences, thus a larger number of states that need to be stored in the transition system. The set abstraction is an abstraction of the sequence approach, but here the number of occurrences of each activity is lost. The multiset abstraction does keep this number of occurrences but ordering of the events in the sequence is lost, as would also be the case in the set approach. Using the multiset of events to represent a trace is similar to the encoding in the creation of frequency vectors in chapter 4. When using the multiset abstraction traces $[a, b, c]$ and $[a, c, b]$ in Figure 5.3 are considered to be equal. This is especially applicable in the reintegration process at UWV since a lot of events happen irregularly, repeatedly and not necessarily in a similar order each time. Using the multiset abstraction still captures the state of the process, but ignores the order. It is evident that a transition system build on the basis of the multiset abstraction can be used to identify the next possible activities for a running case. Whenever the state of a running case is known, the next possible states are also known. For example, in figure 5.4, after state $[a]$ the next possible states according to the transition system are $[a, b], [a, c]$ and $[a, e]$. So activities b,c and e are possible next activities and therefore taken into account when creating recommendations. So when a transition system is created for all the traces in the event log, the possible next activities can be identified, because they are basically just the activities that lead to the next states as stored in the transition system.

At the point where a next activity for a running case is needed, the prefix feature vectors have already been generated since they are also needed for prediction. In chapter 4 a prefix feature vector is defined as the combination of a frequency vector and a feature vector. It makes sense to use frequency vectors to represent the states in the transition system since frequency vectors from the prefix feature vectors for the running cases can be directly matched with states in the transition system to find the next activities. This is why a frequency vector is used to represent a state in the transition system.
A frequency vector for a sequence of events as used in the preprocessing of the methodology (see Definition 4.3) uses an ordering of elements and assign a frequency value to each possible event, even if the event is not present in a trace. It differs from a multiset abstraction of a (partial) trace, since events that are not in the trace are not included in the multiset. In the frequency vector the values for these missing activities would just be 0.

When creating frequency vectors for each state in the transition system an important aspect is that the frequencies of activities are in exactly the same order as in the prefix feature vectors used for training the classifier. Therefore a transition system needs to be created alongside the creation of the classifier such that both the classifier and the transition system use the same ordered list of activity labels to create frequency vectors, since the same dataset is used to train both models. Then, and only then, the transition system can be used to find the applicable events to use in the creation of hypothetical prefixes. The ordered list of events that is used to create the frequency vector in the preprocessing is the ordered list of all activity labels in the entire log. When creating the transition system we only need to consider the activities that are possible to use as a recommendation. In the case of UWV this would be all the activities in the process that can be suggested to customers and/or process managers. This set of activities is complemented with other encoded activities selected by the automatic feature selection. If we derive the ordering of the activities from the prefix feature vectors we can derive the ordered list of activities $A\prime$ for which a frequency vector for state representation can be generated. Let $A\prime$ be the ordered list of activity labels $(a_1, .., a_n)$ where $A \subseteq \bigcup_{\sigma \in \mathcal{L}} \bigcup_{e \in \sigma} \lambda_A(e)$. We can define the count of an event with activity $a \in A\prime$ in a sequence σ as $C_a(\sigma) = \Big|\{e \in \sigma : \lambda_A(e) = a\}\Big|$. The resulting state representation $l_{fv}^{state}(\sigma)$ would be the tuple $\big(C_{a1}(\sigma), .., C_{an}(\sigma)\big)$ where the ordering of $A\prime$ is followed.
The resulting transition can be defined as follows:

Definition 5.1. ***(Transition System)*** *Let $\mathcal{L}$ be an event log. A transition system is a triplet $TS = (S, A, T)$ based on $\mathcal{L}$ and l_{fv}^{state} with:*

- $S = \bigcup_{\sigma \in \mathcal{L}} \bigcup_{k=1}^{|\sigma|} l_{fv}^{state}\big(prefix(\sigma, k)\big)$ *is the state space.*
- $A = \bigcup_{\sigma \in \mathcal{L}} \bigcup_{e \in \sigma} \lambda_A(e)$ *is the set of all activity labels.*
- $T = \bigcup_{\sigma \in \mathcal{L}} \bigcup_{k=0}^{|\sigma|-1} \Big(l_{fv}^{state}\big(prefix(\sigma, k)\big), \sigma(k+1), l_{fv}^{state}\big(prefix(\sigma, k+1)\big)\Big)$ *where $\sigma(i)$ is the i-th event in σ, is the set of transitions.*

From the set of transitions T in the transition system the set of possible next events for a prefix can be derived. $\Big(l_{fv}^{state}\big(prefix(\sigma, k)\big)\Big)$ in $\Big(l_{fv}^{state}\big(prefix(\sigma, k)\big), \sigma(k+1), l_{fv}^{state}\big(prefix(\sigma, k+1)\big)\Big)$ is matched with the frequency vector of the prefix feature vector for a running case. Then, all combinations of this state with $\sigma(k+1)$ in T results isn a set of activities that can be used for recommendation for that prefix.
The next subsection discusses how this set of next activities is used to generate hypothetical prefixes, how predictions are made on these prefixes and how the recommendations are finalized.

Generation of Recommendations

Once a dataset of prefix feature vectors for running cases, a classifier and a transition system are available, recommendations can be generated. Generating recommendations for a running case σ works as follows: Given a transition system (S, A, T) and a running case σ:

1. Find $R = \{a \in A : \exists s \in S : (l_{fv}^{state}(\sigma), a, s) \in T\}$.
2. Encode $\sigma \oplus \{a\}$ for each a in R to create a set of hypothetical prefixes H.
3. Compute a prediction for each $h \in H$.
4. Compute a prediction for σ.
5. Compare the predictions for H with the prediction for σ and choose the best one(s). Activity $a \in A$ is the actual recommendation.

For Step 1, the set of transitions T in the created transition system the set of possible next activities for a prefix can be derived. The encoding of $\sigma \oplus \{a\}$ in Step 2 is derived by increasing the value of the count of activity a by 1 in the original prefix feature vector of the running case. The hypothetical prefix feature vector for a process generated from a prefix usedh length k represents the same process in the hypothetical position $k + 1$ as if a possible next event r has happened. The classifier can now make a prediction for each of the hypothetical prefix feature vectors to estimate what the effect of an event r would be on the process outcome, as in Step 3. In Step 4 the classifier is used to predict the running case itself. When looking at a negative outcome label, the predicted probabilities for the outcome for the original prefix σ and the hypothetical prefix $h \in H$ can be compared to determine a decrease in risk. Then the risk decreases for all the hypothetical prefixes can be compared to identify which recommendations to use, and to determine which recommendations is the best. The event that generates the highest decrease in risk is the best recommendation. The set of activities that are selected as suitable for recommendation can be used here to filter out activities that are not suitable. These activities are removed from the set of generated recommendations. This completes Step 5. Repeating steps one to five for each σ in the set of running cases creates recommendations for all of these traces.
The next section discusses the implementation of the recommendation system as introduced above while using the dataset related to the reintegration process at UWV.

5.3 Implementation

Removed for confidentiality reasons.

5.4 Evaluation using Historical Data

Removed for confidentiality reasons.

Chapter 6

The Koala System

The third research goal of this study is to incorporate the prediction and recommendation technologies into a graphical user interface (GUI). For convenience the GUI is named Koala. The Koala system should include all steps from preprocessing up to the generation of recommendations while being easy to use. This chapter discusses the functions incorporated in Koala, describes how it was created and shows it's design.

The creation of Koala started with an evaluation of the requirements. These requirements were then translated into a graphical design in the form of sketched mockups. Since the rest of the methodology is all written in Python code, a Python compatible GUI package is used to create the actual GUI from the mockups. For the creation of Koala the *PySide2*-package is used[1]. Pyside2 is a Python port for Qt, which is a C++ based GUI creation package that also has cross-platform portability. The software is also open-source and therefore reproducible. The GUI is created for and working in MacOS, but since the software is cross-platform, the widgets should work in Windows and Linux as well. This is not tested, since it is outside the scope of this research. The development of a prototype is in itself sufficient.

The interface consists of 3 main functions; *prediction model generation*, *(predictions and) recommendations generation* and *recommendation tuning*. These 3 functions are accessible through the home screen. The home screen is shown in figure 6.1. This screen is showed when the GUI is launched. From here one of the three main functions can be accessed.

[1] https://wiki.qt.io/Qt$_{for Python}$

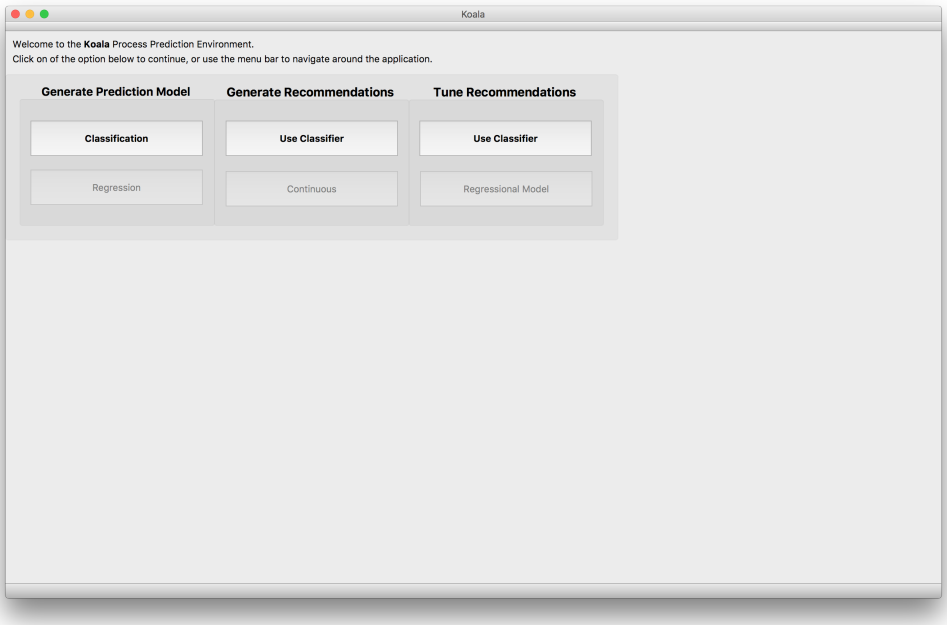

Figure 6.1: The home window of the Koala GUI.

6.1 Classification

When clicking the *Classification*-button on the home screen a new window is opened where a classification algorithm can be trained. The functions available in the classification section are:

- Loading a dataset in CSV format.
- Manual feature selection out of all events, prior to automated feature selection.
- Automated feature selection (Select best k features)
- Selection of events to be used for recommendations out of all events.
- Select scoring method to optimize for.
- Preprocess data for training.
- Run grid search over data.
- Select parameters to train final model with.
- Train and save final model and save metadata to disk.

The main window of the *classification*-section allows the user to upload a CSV file. This file should be a event log in the form of a datatable where each row represents an event record in an event log. In comparison, the widely used standard in process mining .XES cannot be directly used here. Upon opening a CSV file, the contents of the file are showed in the element to the right as shown in figure 6.2. Once a event log is loaded, the user can select which columns represent what in the datatable.

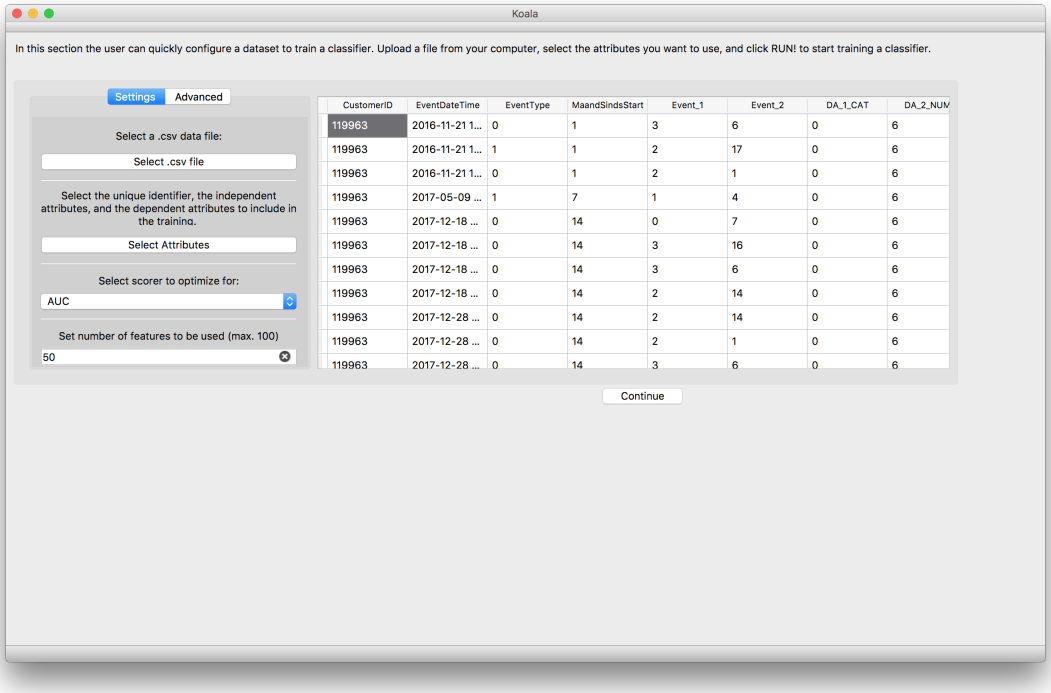

Figure 6.2: The main window in the training section, after a .csv dataset is uploaded. The dataset is visualized automatically.

By clicking the *Select Attributes*-button a dialog window opens where a number of basic attributes can be set. For example, the algorithms behind the GUI need to know which variable represents the *case id* or *timestamp* of the events. The dialog window is showed in figure 6.3. The main window also lets the user set the score to optimize the grid search for when hyperparameter tuning, such as AUC, Accuracy and Recall, as well as the number of features to select in the automatic feature selection.

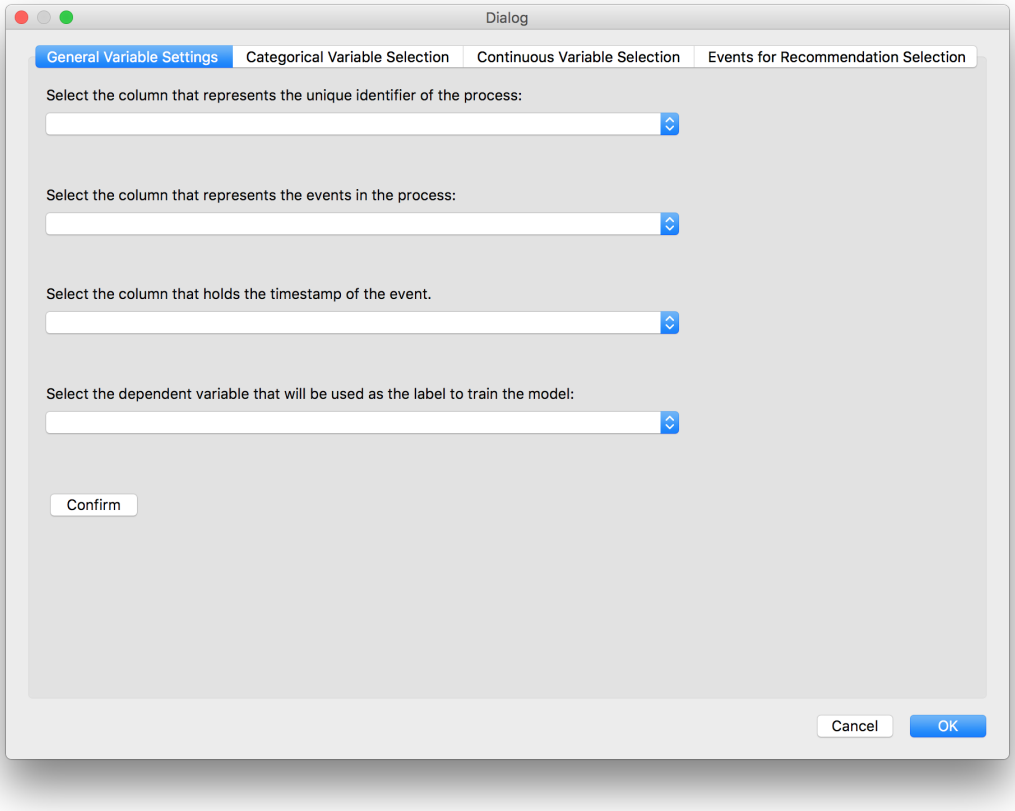

Figure 6.3: The dialog window for setting the main attributes.

The dialog window has multiple tabs, one to set the main attributes, one to set the categorical attributes, one to set the numerical attributes and one to select out of all events the ones the system takes into account when creating recommendations. A list of available attributes is given in each tab and the user can select the needed ones. When an attribute has been selected it is not available anymore in other tabs, making sure a column in the datatable is not used for more than one attribute type.
The general variables can be set in the first tab of the dialog window by clicking the dropdown bars. The available attributes are shown as the names of the columns. Each variable that is set must be unique and all variables must be set before confirming them. When this is not the case an error message is shown. Once confirmed, the other three tabs are unlocked and clickable. Here the other attributes can be selected.

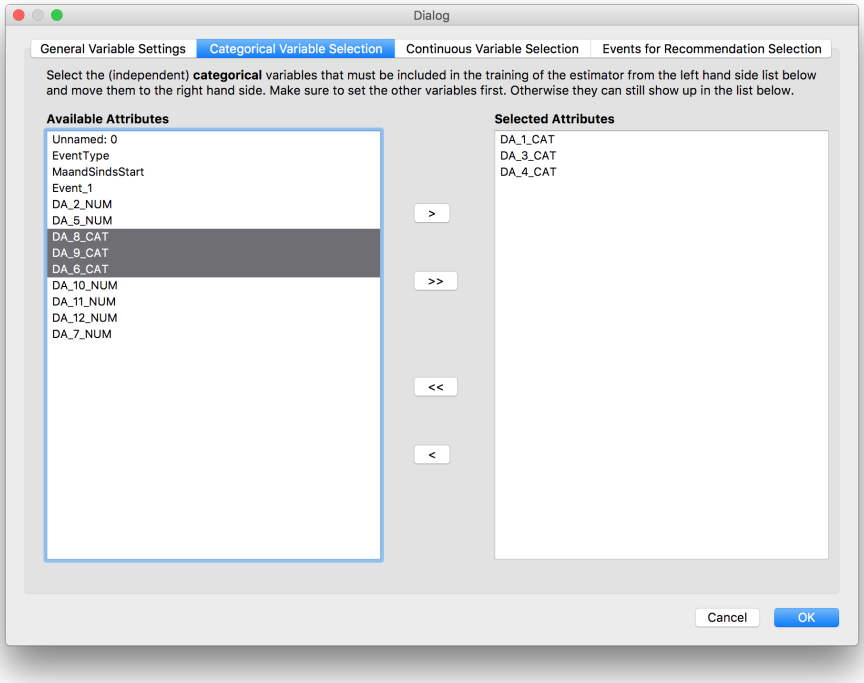

Figure 6.4: The dialog window to select the categorical features.

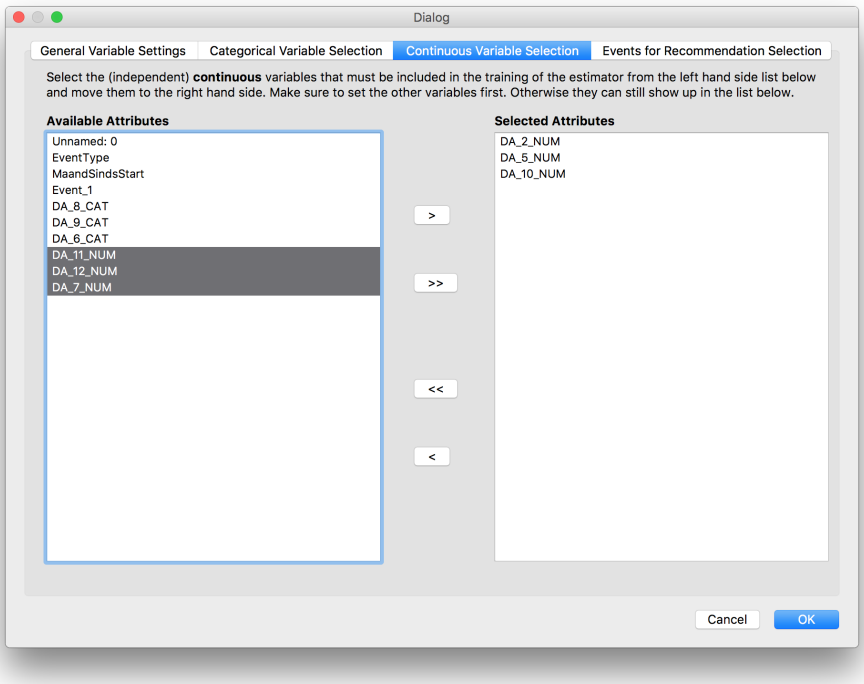

Figure 6.5: The dialog window to select the numerical (or continuous) features. note that the attributes that are selected in fig. 6.4 are no longer available.

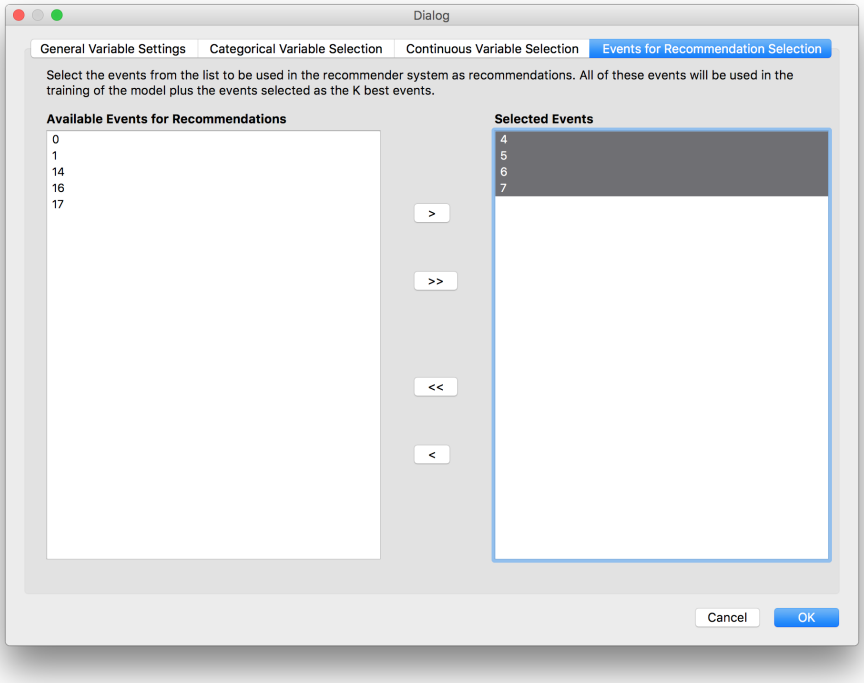

Figure 6.6: The dialog window to select the events to use as recommendations.

The events that are listed in the last tab of the dialog window are directly derived from the column that is selected as the column that represents the event name in the first tab(fig. 6.3).

When all tabs are set the *'OK'*-button saves all the settings and closes the dialog window, returning the user to the main *'classification'*-window(fig. 6.2). Clicking *'Continue'* in the main window starts the main algorithms behind the GUI. These algorithms executes the preprocessing and the grid search in the background. Once these steps are completed, the results of the grid search are showed in a scrollable table widget, as shown in figure 6.7.

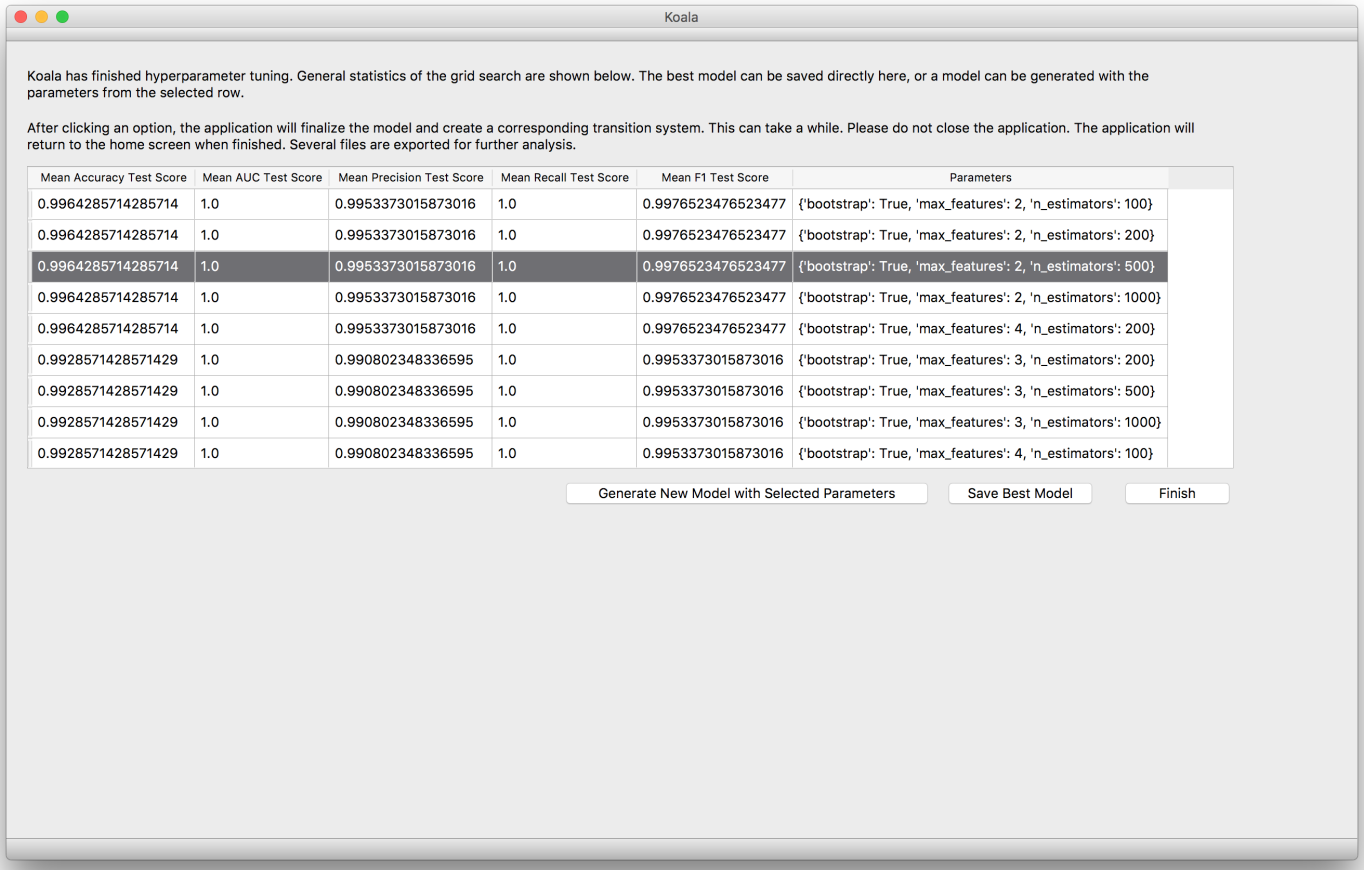

Figure 6.7: The results of the grid search on the uploaded dataset as showed in Koala. The proper model can be saved here.

Clicking the *'Save Best Model'*-button automatically trains a model using the parameters in the top row, which are the best parameters according to the grid search and the scoring that was selected earlier to optimize for. In the result table a row can also be selected if a classifier must be trained with the parameters in that row. Clicking the *'Generate Model with Selected Parameters'*-button trains a new model using these parameters. Metadata that is needed for prediction and recommendation is also generated followed by the generation of the transition system for the recommendations. The model, the metadata and the transition system are stored together. This file is needed by Koala for prediction and recommendation generation as described in the next section. The *'Finish'*-button returns the user to the *'Home'*-window of the GUI.

6.2 Prediction and Recommendation Generation

By clicking the *'Use Classifier'*-button in the main window the user is directed to the *'Generate Recommendation'*-section of the GUI as shown in figure 6.8. The functions available in the recommendation generation section are:

- Upload a classifier and a dataset of unfinished traces.
- Visualize metadata of uploaded model and uploaded dataset.
- Preprocess dataset into proper feature vector for the classifier.
- Generate predictions for dataset
- Generate recommendations for dataset
- Save a datatable consisting of the predictions and recommendations to disk.

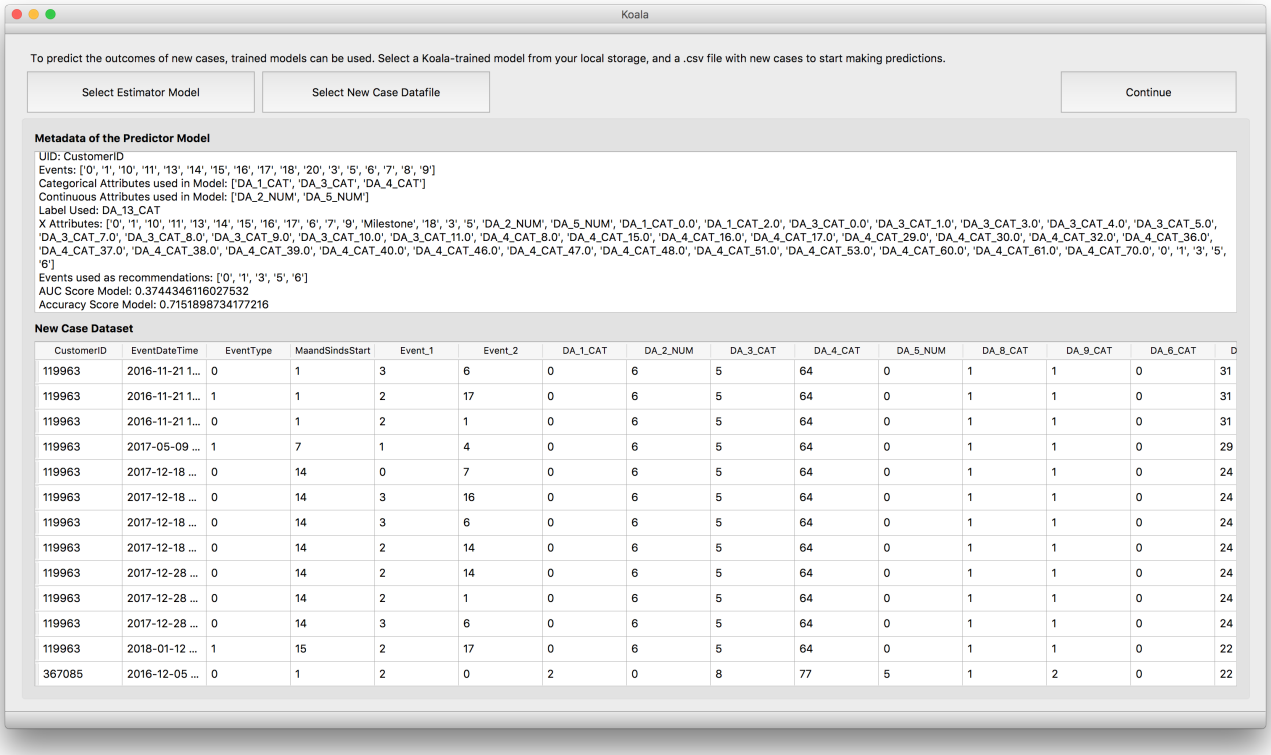

Figure 6.8: The main window within the *'Generate Recommendations'* part of the GUI. The model and dataset are already loaded here.

A dataset containing the event logs for incomplete traces and a Koala trained model file are loaded here. The metadata of the model is shown in the top widget where the contents of the dataset are shown in the bottom widget. This helps in identifying if the loaded dataset is compatible with the model with respect to the used variables.
Clicking *'Continue'* will launch the algorithms that create a datatable containing both predictions for each case and a set of recommendations for each case. In a next window, this datatable can be saved as a CSV file for analysis. Viewing and tuning this datatable is also possible in the GUI as described in the next section.

6.3 Recommendation Tuning

Once the recommendations are generated for a set of running cases, a user might want to analyze and filter them in the GUI as well. A third section is created that facilitates this. The reason that tuning is separate from the generation of recommendations is because a user might want to tune a set of recommendations multiple times or at later moments, so it must also be accessible directly. The functions available in the recommendation tuning section are:

- Upload a dataset with recommendations as generated by Koala.
- Filter datatable based on risk levels.
- Filter datatable based on recommendation effectiveness.
- Filter datatable based on realism of suggested recommendation. I.e. the proximity of the altered prefix (prefix concatenated with an event) with a prefix as stored in the transition system(based on historical data).
- Save tuned recommendation table to disk.

The *'Recommendation Tuning'*-section consists of one window where the recommendations file generated in the second section can be loaded into the GUI. Koala then shows this table. A set of tools is available to filter the datatable, as shown in figure 6.9.

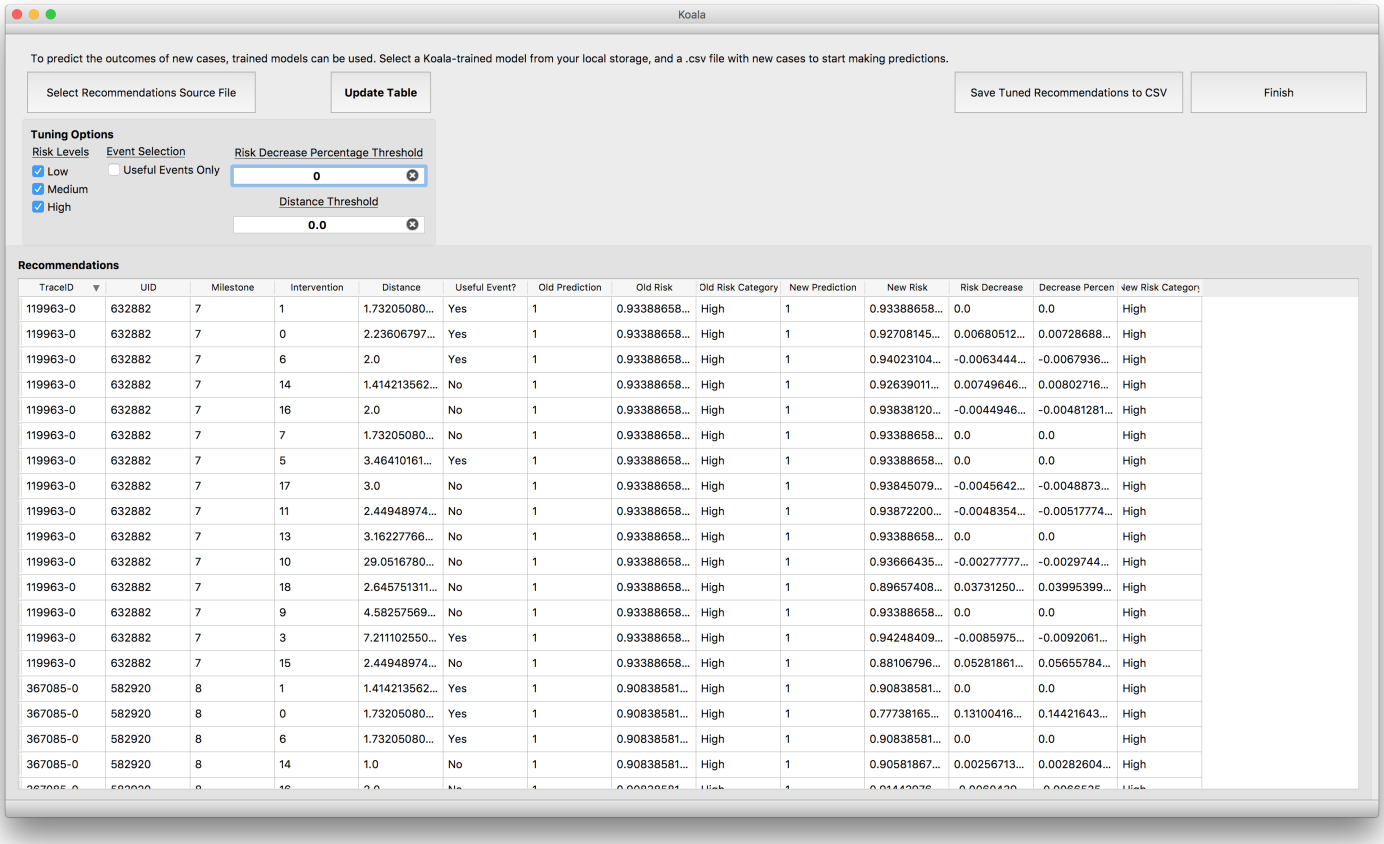

Figure 6.9: The screen where recommendations can be tuned. A recommendations source file as generated by Koala is already loaded here.

One of the options for tuning is to include high risk cases only. These cases are identified as having a probability of failure of 70% or higher. Medium risk cases have a 50-70% chance of failing. Cases with a positive prediction probability are considered as low risk cases. Unchecking one of the risk options in the GUI automatically refreshes the table but now without that option. Koala includes all events in the generation of recommendations. When setting up the classifier in the first section, a subset of all events is selected to use as recommendations. To filter the recommendations here the user can select to include only the events to be considered useful by checking the *'Useful Events Only'*-option. The table is refreshed automatically on checking and unchecking here as well. The user can also set two thresholds for filtering of the table. The first option is the predicted amount of risk decrease when a recommendation is applied, i.e. the effectiveness of the recommendation according to the system. This can be done by typing in a percentage in the *'Risk Decrease Percentage Threshold'*-form. The other threshold describes the euclidian distance of the altered prefix with a similar case in the transition system, as described in chapter 5. This describes the realism of using such a recommendation, based on if it has happened in similar cases before or not. Setting this threshold leaves out recommendations that are not very realistic from a business point of view as set by the user. After a threshold is set in either one of the boxes, the user clicks the *'Update Table'*-button which checks if the typed in number(s) are formatted correctly and then updates the table for the ones where the number if formatted correctly.
A filtered table can be saved to an additional CSV for further applications.

6.4 Improvements & Extensibility

The created GUI is an early version of a process prediction tool. Since it is an early version, the layout and scaling of the GUI is not yet very good. There is also some room for improvement in the navigation between the sections, for example direct links between the creation of the recommendations and the tuning could be established. Another improvement might be the options for interaction and filtering in the scrollable tables that are now part of the GUI. Functionalities could be added to these tableviews, e.g. sorting of columns by clicking on them. Tooltips can be added while hovering over something to give some additional information about how everything works.

The GUI can also be extended to incorporate regression predictions as well. The home window of Koala(fig. 6.1) already incorporates the buttons for this. Although a regression technique is tested at some point during this study, it has not matured into a working version for the GUI. Next to regression the GUI could also be extended with advanced settings for the training. A tab for this is already added to main train window as shown in figure 6.2. Within this tab settings for alternative classifiers, alternative hyperparameter and grid search settings could be placed in order to provide the user with more options to train and optimize a model. The GUI could be extended such that it can leave out certain events as well, in a similar way as how the categorical and continuous attributes are selected. This creates more flexibility in training a model. At this point the GUI only works with CSV files, but it could be extended to also be able to interpet XES files as seen in other process mining tools. In addition a data analysis component could be useful as well. This component should be able to visualize the data and the individual variables and show important statistics like average values. Perhaps a statistical distribution could be visualized for variables in order to look for outliers or strange patterns in the data that might affect the quality of the models. An univariate analysis, as introduced in the feature selection section of chapter 4, could even be added.

Chapter 7

Conclusions

This thesis introduced a framework for the prediction of process outcomes. This places this study within the academic field of predictive process monitoring, which uses machine learning techniques to create models that can predict the outcome of running processes in order to estimate performance. The framework is designed for all types of processes but is especially tailored to fit the core subject of the case study at UWV. This subject is the reintegration process, in which UWV helps unemployed people to get back to work while supporting them financially in the meantime. An important Key Performance Indicator of this process is its duration, which represents how long a person has been unemployed. This duration should be minimized as much as possible, and it should not surpass the max duration which is assigned to a case. A methodology has been created for the creation of a classifier that can accurately predict whether a case will have a favorable KPI outcome at the end of the process. In the case of the reintegration process this KPI is whether the maximum duration of a reintegration process is reached. Within the methodology introduced in this research, a number of workflows have been designed. First, a preprocessing workflow has been created that can automatically transform an event log into a set of feature vectors that are used to train a classifier. Data from the reintegration process at UWV is used to implement and test this workflow. This data is analyzed and cleaned first. Since the data has a lot of *null*-values, *null*-values in many cells in the dataset needed to be imputed, which is the replacement of *null*-values with another, more generic, value. This could result in a lot of rows having the same value for a certain column, which can cause overfitting towards the values the *null*-values are replaced with. Eventually, this was not the case, the classifier was tested on a totally separate dataset, and it still performed very well. Feature Engineering is applied to deal with the certain variables in the dataset, that would otherwise be less useful and meaninful. After data cleaning and feature engineering, a mechanism to generate prefixes from the event log is created and implemented in the workflow. For the generation of prefixes out of traces, a milestone approach is introduced to overcome some of the issues with other options for prefix generation such as the fixed-length, index or gaps abstractions. These other options had problems with the irregularity and lack of structure in the UWV event log, and would result in prefixes that do not make sense from a business point of view. Using milestones overcomes this problem. The next step in predictive process monitoring usually is to divide the prefixes into buckets. In this research the single bucket approach is used for convenience, which works fine regarding the quality of the predictions. Other bucketing methods have not been tried out, which is a drawback of this research. After bucketing, the prefixes are encoded into feature vectors. The sequence of events in a prefix is encoded using a frequency vector. This approach does not take into account the order of events, only the amount of each event in the prefix. Considering the irregularity and variation of traces in the data, other encoding might have resulted in a lot of unique feature vectors, which might have decreased the quality of the prediction model due to overfitting. Once again other techniques have not been tested in this study. For the encoding of attributes in the trace, last state encoding is used. The vectors for the event execution and attributes are combined into a final prefix feature vector for each prefix in the event log. A classifier is trained on this set of vectors and tested on a different dataset using the

random forest classifier, which parameters have been optimized for the dataset. Other algorithms have briefly been tested, but the random forest classifier proved to be a reasonable choice. The trained classifier proved to be capable of predicting process outcomes for the reintegration process at UWV, which fulfills Research Goal 1. The classifier can still be improved with respect the the amount of false positives that are predicted.

Next a recommendation system is created. The main mechanism of the recommendation system is to concatenate a trace of a running case with an activity and then use the classifier to identify whether the activity decreases the predicted risk. The concatenation is encoded by adding one event, representing the activity, to the frequency vector of the original prefix feature vector. This assumes that that event has hypothetically happened in that running case. If the resulting predicted risk is lower than the prediction for the original prefix feature vector the event can be use as a recommendation. The major problem with this approach is that not each event can be used to tweak a prefix, since some events are not suitable for recommendations because of their nature, and some events do not happen at each moment within a process. The latter implies that an event can only be used as a recommendation if it is has happened in other process instances before at a similar point in time within the process. Therefore, a set of traces needed to be identified which have a partial trace that is similar to the prefix of the running case. For these similar traces, events that happened after the partial trace can be used as a recommendation the running case. To identify these similar traces, a transition system is created on historical data to identify all the states that the process has been in. Similar states to the one of the running case can be found in the transition system such that the events that happened directly after that state can be identified. These are the events that make sense to be used as a recommendation, from a historical point of view. However, while creating a transition system for the reintegration process at UWV, the amount of different states proved to be huge, so finding an exact match between states in the transition system and the state of the running case caused a problem. To overcome this, also states that are quite similar to the state of the running case are taken into account. This similarity is assessed using the distance between two frequency vectors, which makes the claim that only activities are used that have happened before in a similar context, less strong. With the set of events (representing an activity in the process) that can be used as a recommendation, the original prefix feature vector of the running case is transformed into a set of hypothetical prefixes, one for each event in the set of selected events. Predictions are generated for these prefixes to identify which of the activities are most effective, and a set of ranked recommendations is created. With the introduction of this mechanism the recommendation sytem has been created, which fullfils Research Goal 2. However, the evaluation of the recommendation system using historical data is not perfect. The approach introduced in this thesis only used the historical data to identify if a suggested recommendation has worked in similar cases to the case the recommendation is created for. Conclusions for the effect of the recommendation on the actual running case cannot be drawn from this approach. Since the amount of time available for this study was limited, and the processes at UWV usually take quite some time, the recommendations could not be tested within process instances that are currently still running at UWV. Only if the recommendations are tested in a real life scenario, their actual effect on process outcome can be determined.

A GUI has been created in parallel with the development of the algorithms involved in creating a prediction and recommendation system for process outcomes. This system incorporates both the prediction and recommendation system and is usable with different datasets. Therefore Research Goal 3 has been fulfilled. The system is only a prototype, and it still has it flaws and imperfections. In the future Koala can be optimized with respect to its design and its algorithms, and new functions can be added, like the option to also predict continuous process outcomes through regression.

7.1 Suggestions for UWV

In the evaluation of the recommendation using historical data it became clear that in processes where the recommended activity was executed the process overall performed better. We recommend that UWV investigates the possibilities of implementing and testing the recommendation system within the reintegration process. The amount of data that is stored by UWV for the reintegration process is large. The full potential of the data can be unlocked if the workflows are further fine tuned to fit the data better. If implementation of the system proves to be challenging, at least the methodology for prediction of process outcomes should be used to assess the risks of customers within the process. Work coaches and other employees involved with the reintegration process can use these prediction to focus their attention on the customers that actually need help reintegrating. With respect to the greated GUI, UWV should investigate the option to build and implement a similar system that can generate predictions and recommendations for processes in an automated manner. Such a system should be easy to use by different types of employees but leave room for more advanced functionalities that can be used by, for example, business analysts.

7.2 Future Work

This research extends the work in the field of predictive process monitoring by adding a system for the generation of recommendations. The methodology in this research is only a proof-of-concept and it should therefore be tested using different datasets which originate from different types of processes. It should also be tested within a set of running processes, to see if the recommendations actually work in practice. The methodology in this research predicts categorical KPI outcomes through classification. It can be extended to implement regression as well. Regression has been experimented with in the beginning of the project and the results were promising. Extending the recommendation system to work with regression models as well can therefore be valuable. The created Koala system can be transformed into an open-source tool for researchers and process managers to experiment with. The mechanisms of the system could also be added as a functionality to (open-source) process mining tools such as PROM and RapidMiner.

Bibliography

[1] UWV - About Us. https://www.uwv.nl/overuwv/english/about-us-executive-board-organization/detail/about-us. Accessed: 17-10-18. 1

[2] Wil M. P. Van Der Aalst. Data scientist: The engineer of the future. *Enterprise Interoperability VI*, page 1326, 2014. 5

[3] David Hand, Heikki Mannila and Padhraic Smyth. *Principles of Data Mining*. MIT press, Cambridge, MA, 2001. 7, 8

[4] Massimiliano De Leoni, Wil M.P. Van der Aalst, and Marcus Dees. A general process mining framework for correlating, predicting and clustering dynamic behavior based on event logs. *Information Systems*, 56:235257, 2016. 14, 15, 20, 22

[5] Chiara Di Francescomarino, Marlon Dumas, Fabrizio Maria Maggi, and Irene Teinemaa. Clustering-based predictive process monitoring. *IEEE Transactions on Services Computing*, 2016. 14, 15

[6] Marlon Dumas, Macello La Rosa, Jan Mendling, and Hajo A. Reijers. *Fundamentals of business process management*. Springer, 2013. 7

[7] Manuel Fernández-Delgado, Eva Cernadas, Senén Barro, and Dinani Amorim. Do we need hundreds of classifiers to solve real world classification problems? *The Journal of Machine Learning Research*, 15(1):3133–3181, 2014. 15

[8] Chiara Di Francescomarino, Marlon Dumas, Fabrizio Maria Maggi, and Irene Teinemaa. Clustering-based predictive process monitoring. *IEEE Transactions on Services Computing*, page 11, 2017. 14, 20

[9] Geetika T Lakshmanan, Songyun Duan, Paul T Keyser, Francisco Curbera, and Rania Khalaf. Predictive analytics for semi-structured case oriented business processes. In *International Conference on Business Process Management*, pages 640–651. Springer, 2010. 14, 15

[10] Anna Leontjeva, Raffaele Conforti, Chiara Di Francescomarino, Marlon Dumas, and Fabrizio Maria Maggi. Complex symbolic sequence encodings for predictive monitoring of business processes. In *International Conference on Business Process Management*, pages 297–313. Springer, 2015. 14, 15

[11] Yanchi Liu, Zhongmou Li, Hui Xiong, Xuedong Gao, and Junjie Wu. Understanding of internal clustering validation measures. *2010 IEEE International Conference on Data Mining*, 2010. 9

[12] Fabrizio Maria Maggi, Chiara Di Francescomarino, Marlon Dumas, and Chiara Ghidini. Predictive monitoring of business processes. In *International Conference on Advanced Information Systems Engineering*, pages 457–472. Springer, 2014. 14, 15, 16

[13] Marcus Dees, Massimiliano de Leoni and Felix Mannhardt. Enhancing process models to improve business performance: A methodology and case studies. In *On the Move to Meaningful Internet Systems. OTM 2017 Conferences*, pages 232–251. Springer International Publishing, 2017. 7

[14] Andreas C. Mller and Sarah Guido. *Introduction to machine learning with Python: a guide for data scientists*. OReilly, 2017. 9, 22, 23, 24

[15] Arik Senderovich, Chiara Di Francescomarino, Chiara Ghidini, Kerwin Jorbina, and Fabrizio Maria Maggi. Intra and inter-case features in predictive process monitoring: A tale of two dimensions. *Lecture Notes in Computer Science Business Process Management*, page 306323, 2017. 13

[16] Niek Tax, Ilya Verenich, Marcello La Rosa, and Marlon Dumas. Predictive business process monitoring with lstm neural networks. *Advanced Information Systems Engineering Lecture Notes in Computer Science*, page 477492, 2017. 13

[17] Teinemaa, I., Dumas, M., La Rosa, M., Maggi, F. M. Outcome-Oriented Predictive Process Monitoring: Review and Benchmark. 2017. iiiiii, ixix, ixix, 6, 7, 8, 9, 13, 14, 15, 16, 17, 20, 23, 24

[18] Wil M.P. Van der Aalst. *Process Mining: Data Science in Action*. Berlin: Springer-Verlag, 2011. ixix, ixix, ixix, 5, 8, 9, 32, 33

[19] Wil M.P. Van der Aalst. *Process Mining: Discovery, Conformance and Enhancement of Business Processes*. Berlin: Springer-Verlag, 2011. 5

[20] Ilya Verenich, Marlon Dumas, Marcello La Rosa, Fabrizio Maria Maggi, and Chiara Di Francescomarino. Complex symbolic sequence clustering and multiple classifiers for predictive process monitoring. In *International Conference on Business Process Management*, pages 218–229. Springer, 2015. 14, 15

[21] Wil M.P. van der Aalst, Arya Adriansyah, Ana Karla Alves de Medeiros, Franco Arcieri, Thomas Baier, Tobias Blickle, Jagadeesh Chandra Bose, Peter van den Brand, Ronald Brandtjen, Joos Buijs, Andrea Burattin, Josep Carmona, Malu Castellanos, Jan Claes, Jonathan Cook, Nicola Costantini, Francisco Curbera, Ernesto Damiani, Massimiliano de Le- oni, Pavlos Delias, Boudewijn F. van Dongen, Marlon Dumas, Schahram Dustdar, Dirk Fah- land, Diogo R. Ferreira, Walid Gaaloul, Frank van Geffen, Sukriti Goel, Christian Gu nther, Antonella Guzzo, Paul Harmon, Arthur ter Hofstede, John Hoogland, Jon Espen Ingvaldsen, Koki Kato, Rudolf Kuhn, Akhil Kumar, Marcello La Rosa, Fabrizio Maggi, Donato Malerba, Ronny S. Mans, Alberto Manuel, Martin McCreesh, Paola Mello, Jan Mendling, Marco Mont- ali, Hamid R. Motahari-Nezhad, Michael zur Muehlen, Jorge Munoz-Gama, Luigi Pontieri, Joel Ribeiro, Anne Rozinat, Hugo Seguel P erez, Ricardo Seguel P erez, Marcos Sepu lveda, Jim Sinur, Pnina Soffer, Minseok Song, Alessandro Sperduti, Giovanni Stilo, Casper Stoel, Keith Swenson, Maurizio Talamo, Wei Tan, Chris Turner, Jan Vanthienen, George Varvaressos, Eric Verbeek, Marc Verdonk, Roberto Vigo, Jianmin Wang, Barbara Weber, Matthias Weidlich, Ton Weijters, Lijie Wen, Michael Westergaard, and Moe Wynn. Process mining manifesto. *Business Process Management Workshops*, 2012. 6