Humboldt-Universität zu Berlin
Mathematisch-Naturwissenschaftliche Fakultät
Institut für Informatik

Feature extraction from event logs for predictive monitoring of business processes

Masterarbeit

zur Erlangung des akademischen Grades

Master of Science (M. Sc.)

eingereicht von: Florian Borchert
geboren am: 10.06.1990
geboren in: Neubrandenburg

Gutachter/innen: Prof. Dr. Matthias Weidlich
Prof. Dr. Marius Kloft

eingereicht am: ..... verteidigt am: .....

Abstract

Many organizations record information about their IT-driven processes in some form of event logs, be it for auditing or debugging purposes or simply by accident. Obtaining knowledge from such event logs is the goal of *process mining*. Process mining techniques can be used to discover process models from event logs or relate existing process models and observed behaviour in various ways. While we usually envision process models to carry some executable semantics, like Petri net systems or BPMN models, we can also think of purely statistical models or a combination of both. Statistical models are particularly suited for predictive tasks such as early classification of a running case or anomaly detection. In the context of business process management, these activities are denoted as *predictive monitoring*.

However, most standard machine learning and data mining algorithms cannot be directly applied to complex event logs. Some of the most widespread algorithms, like Support Vector Machines or decision tree learners, require the data to be mapped into a vector space first. To this end, meaningful features need to be extracted from event logs.

Now, it is not obvious what good features in the domain of business processes are: events describe observed behaviour and are typically equipped with information about execution times and quite often also resources and other attributes. Business processes usually show a certain degree of structure in terms of subprocesses and workflow patterns, which is reflected in the logs. Furthermore, processes may allow for a parallel execution of activities, so that the order in which events are recorded can have different interpretations. Treating the traces in an event log simply as symbolic sequences of activities neglects all of these potentially useful perspectives.

In this master's thesis, we investigate feature extraction techniques for event log traces. We provide an overview over the field of predictive monitoring and analyze existing trace profiles proposed in the literature. Based on this, we apply the results obtained in recent research on process discovery to find meaningful abstractions over related subsequences of events.

We assess different feature sets by evaluating their predictive power in a supervised classification setting as well as a semi-supervised outlier detection setting. For this purpose, we use two datasets from public administration, describing complex processes in EU agricultural subsidy management. Our particular goal in this domain is to predict negative outcomes, for instance, additional work due to legal claims or corrections necessary after initial payment decisions.

Zusammenfassung

Viele Organisationen protokollieren Informationen über ihre IT-gestützten Prozesse in Form von Event Logs, sei es aus Auditierungsgründen, zum Debugging oder schlicht unbeabsichtigt. Nützliches Wissen aus den Event Logs von Geschäftsprozessen zu gewinnen ist das Ziel von *Process Mining*. Process-Mining-Techniken können eingesetzt werden, um Prozessmodelle aus Event Logs zu erzeugen oder bestehende Prozessmodelle und das beobachtete Verhalten auf verschiedene Weise in Verbindung zu bringen. Während bei *Prozessmodellen* typischerweise an Modelle mit einer Semantik zur Ausführung ihrer Elemente gedacht wird, wie Petrinetze oder BPMN-Modelle, sind auch rein statistische Modelle oder eine Kombination aus beidem möglich. Statistische Modelle eignen sich insbesondere für Aufgaben wie die frühzeitige Klassifizierung von Prozessinstanzen oder die Erkennung von Anomalien. Im Kontext des Geschäftsprozessmanagements werden diese Aufgaben als *Predictive Monitoring* bezeichnet.

Allerdings sind die meisten Standardalgorithmen des maschinellen Lernens und Data Mining nicht einfach auf komplexe Event Logs anwendbar. Für einige der meistverbreiteten Lerngalgorithmen, wie Support Vector Machines oder Decision Trees, müssen die Daten zunächst in einen Vektorraum überführt werden. Dafür müssen aussagekräftige Merkmale (Features) aus den Event Logs extrahiert werden.

Nun ist es nicht offensichtlich, was gute Features für Logs von Geschäftsprozessen ausmacht: Events in einem Log beschreiben beobachtetes Verhalten and sind typischerweise mit weiteren Informationen über Ausführungszeit und oft auch Ressourcen und mit anderen Attributen ausgestattet. Geschäftsprozesse weisen üblicherweise einen gewissen Grad an Strukturiertheit in Form von Subprozessen und Workflow-Patterns auf. Außerdem können Prozesse die parallele Ausführung von Aktivitäten erlauben, so dass nicht klar ist, in welcher Reihenfolge die aufgezeichneten Events zu interpretieren sind. Die Traces in einem Event Log einfach als symbolische Sequenzen von Aktivitäten zu behandeln, vernachlässigt all diese möglicherweise sinnvollen Perspektiven.

In dieser Masterarbeit untersuchen wir Methoden der Feature-Extraktion aus Event Logs. Wir beschreiben und analysieren verschiedene Vorschläge aus der Literatur. Darauf aufbauend wenden wir die Resultate der jüngeren Forschung zur *Process Discovery* an, um sinnvolle Abstraktionen über Teilsequenzen von Events zu finden.

Wir beurteilen verschiedene Arten von Features, indem wir ihre Vorhersagekraft sowohl in einem überwachten Klassifizierungsverfahren als auch einem halb-überwachten Ausreißererkennungsverfahren evaluieren. Zu diesem Zweck verwenden wir zwei Datensätze der öffentlichen Verwaltung, die komplizierte Prozesse in der Bearbeitung von Anträgen auf EU-Agrarfördermittel beschreiben. Unser Ziel ist hier, Muster in den Abläufen zu erkennen, die mit negativen Folgen assoziiert sind. Dazu zählt z. B. zusätzliche Nacharbeit, die durch Klagen oder notwendige Korrekturen von Amtsseite entsteht.

Contents

List of Figures

List of Tables

1 Introduction

Faced with the ever growing amount of data that we treasure up, we need to think of intelligent ways to exploit the knowledge it may embody. A particularly rich kind of data is recorded in the form of *event logs* by all kinds of software systems. Such logs may be readily available in an explicit representation or buried somewhere in the system, but it is hard to think of an information system whose usage does not leave any traces. Depending on the quality of an event log, its analysis may yield valuable insights about the processes the corresponding system is meant to implement or support. Knowledge extraction from event logs from a business process perspective is the goal of *process mining*.

We can phrase this as a learning task: Given *example executions* of a business process, captured in an event log, can we *learn* a model of this process that generalizes well to unseen process instances?

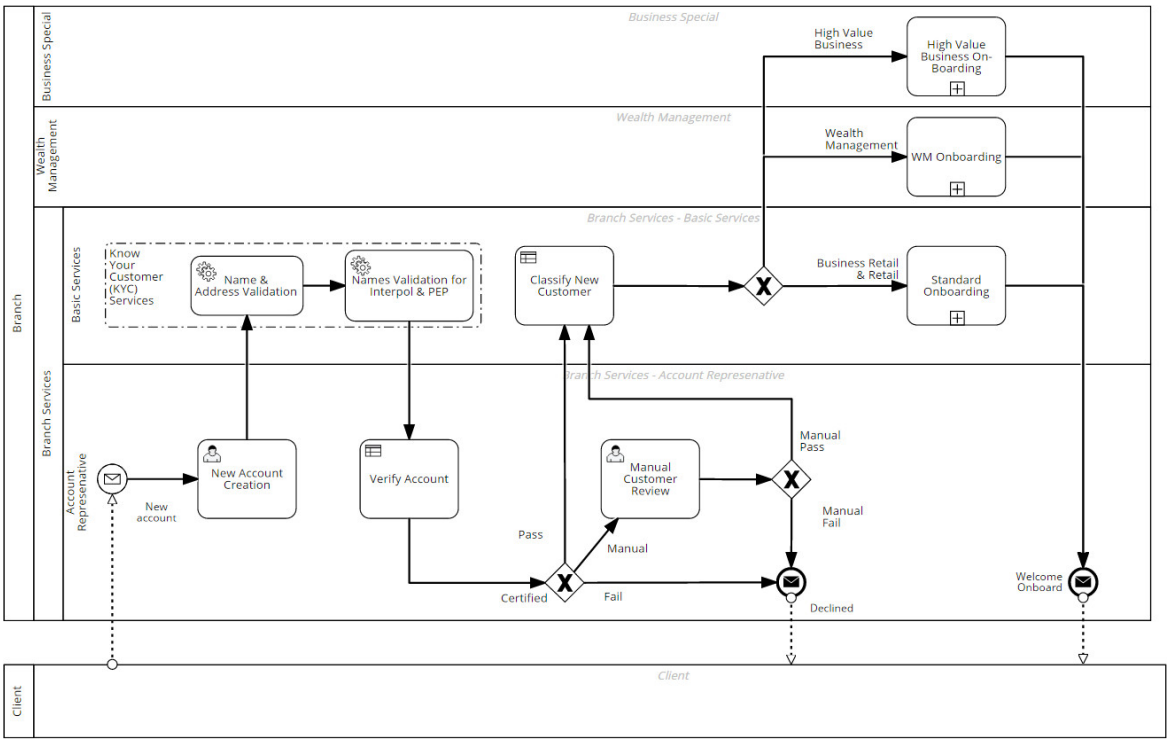
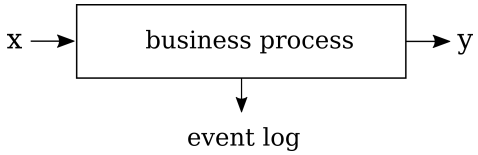


Figure 1.1: A BPMN workflow model enriched with organizational information represented by horizontal lanes, from [95]

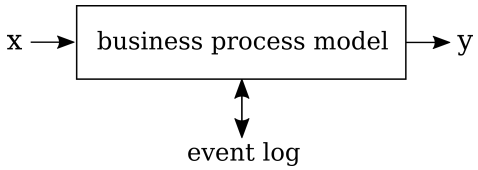
A model like this may take various forms. For instance, BPMN (Business Process Model and Notation) has been widely used in practice for process modelling. An example of a such a model is shown in fig. 1.1. And indeed, it is possible to discover BPMN models and other workflow models (such as Petri net systems) from event logs [29].

Discovery of (executable) process models is arguably the most impressive and intuitive flavour of process mining. While such an executable process model can be very useful, it may not be a necessary prerequisite to answer certain questions about the process. Consider the activity of *predictive monitoring* of business processes, by which we mean the ability to make predictions for running process instances. If predictive accuracy is our primary interest, a statistical model of the relation between the ongoing case and the response variable may serve this purpose.

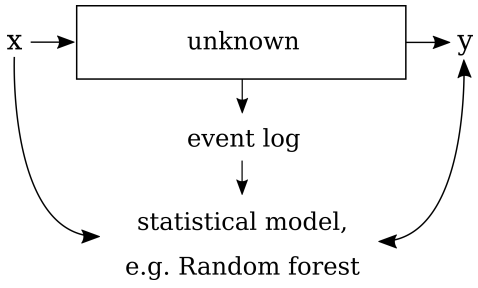
To illustrate this point, we adopt an argument presented by Leo Breiman [20][1]. We assume some input x and output y and that both are related by a business process, whose execution can be observed through an event log.



We could try to understand exactly how the business process "works", i.e., how it associates inputs and outputs. We could achieve this by constructing an executable, integrated business process model that fits the observed behaviour:



This usually requires strong assumptions about the nature of the business process, but may be worthwile if these assumptions are justified. But we could also treat the business process as a black box and use some statistical model to associate x, the execution history of the running process, and y:



[1] Breiman actually contrasted *data modelling* and *algorithmic modelling*, arguing that data models are often a poor emulation of nature, but we saw parallels to the use of business process models.

where the statistical model can itself be a more or less complex black box, for instance, if it is represented by an artificial neural network or an ensemble of simpler models.

In this thesis, we follow the second approach. In particular, we examine how standard machine learning algorithms can be used to learn predictive models for ongoing cases. Usually, this requires mapping the input data and trace of events to a representation suitable for the particular algorithm, i.e., *features* need to be constructed that capture the interesting aspects of the observed behaviour. Since feature extraction is a highly domain specific task, we direct our attention to representations that incorporate the typical properties of business processes, which include the presence of hierarchical substructures and concurrency as well as varying granularity levels of logged events.

To illustrate and evaluate our findings, we use real life event logs from the public administration in the domain of EU agricultural subsidy management collected over the period of multiple years.

We are aware that using event logs the way we suggest here may be considered problematic. Typically, there are people involved in business processes and information in event logs may therefore be very sensitive. Learning models of (human) behaviour raises all kinds of issues regarding data privacy and surveillance. Therefore, we want to emphasize right at the beginning that our example datasets have been properly anonymized and we do not see any potential for abuse in our particular application.

1.1 Organization

The thesis is structured as follows:

- In chapter 2, we introduce our notion of event logs along with a brief overview of the field of process mining, then narrow the scope to predictive monitoring of business processes.
- In chapter 3, we discuss related work, including a summary of trace representations and various types of predictions that have been attempted using event logs.
- In chapter 4, we provide a detailed discussion about features that can be extracted from process cases in order to feed them into machine learning algorithms.
- In chapter 5, we briefly introduce basic machine learning concepts, algorithms and performance measures.
- In chapter 6, we present the event logs we have used in our experiments.
- In chapter 7, we explain the experimental setup and results for different binary classification and outlier detection scenarios.
- In chapter 8, we summarize our findings and point to interesting future work.

1.2 Contributions

We compare different types of trace representations, respectively feature sets, in supervised and semi-supervised machine learning settings with respect to their predictive power on real life data sets. We propose an approach to derive aggregations of attributes on subsets of related events and apply it to incorporate execution time information on different levels of granularity. To make predictions, we employ binary as well as one-class classification algorithms, and assess the suitability of the latter in cases where only or almost only instances of one class are available for training.

2 Learning from event logs

This chapter introduces basic concepts necessary for the remainder of this thesis. To begin with, we provide a definition for event logs that serves as the basis for all subsequent analysis. We then give a brief overview of the field of process mining in general as well as the goal of predictive monitoring of business processes and the challenges that arise when analyzing event logs.

2.1 Event logs

Table. 2.1 shows an example of what we will refer to by the term *event log*. The log consists of multiple *cases*, each corresponding to a process *instance*. We assume that cases can be easily distinguished in the log (e.g., by a case ID), otherwise we would need to correlate events to cases first [41].

Each case carries a number of *case attributes* as well as a *trace* of events. We formalize this notion of events and event logs by closely following the definitions in [1, p. 100ff.]:

Definition 1 (Event). *Let $\mathcal{E}$ be the set of all possible* events *(the event universe). An event $e \in \mathcal{E}$ is characterized by attributes from the set of event attributes EA. For an attribute name $a \in EA$ we denote by $\#_a(e)$ the corresponding attribute value of e. Further, $\#_a(e) = \perp$ if attribute a is missing in e.*

It is assumed that every event has at least the following attributes:

$\#_{act.}(e) \in \mathcal{A}$	...	*the activity corresponding to e (where $\mathcal{A}$ is the set of possible activities)*
$\#_{time}(e) \in \mathcal{T}$	...	*the timestamp of e (from a time domain $\mathcal{T}$ which is totally ordered)*

Other event attributes that might be encountered (but not required) include:

$\#_{trans}(e)$	...	the *transaction type* of e if its activity has a lifecycle (e.g., start, completion or suspension of an activity)
$\#_{resource}(e)$	...	a *resource* associated to e, e.g., the user who executed the activity
$\#_{cost}(e)$	...	the *cost* associated to executing the activity

Case ID	Customer ID	Activity	Resource	Cost	Timestamp
1	2	A	Pete	50	2010/12/30 11:02:00
		F	Sue	400	2010/12/31 10:06:00
		C	Mike	100	2011/01/05 15:12:00
		D	Sara	200	2011/01/06 11:18:00
		H	Pete	200	2011/01/07 14:24:00
2	2	A	Mike	400	2010/12/30 11:32:00
		C	Mike	100	2010/12/30 12:12:00
		B	Sean	200	2010/12/30 14:16:00
		D	Sara	200	2011/01/05 11:22:00
		G	Ellen	50	2011/01/08 12:05:00
3	1	A	Pete	100	2010/12/30 14:32:00
		B	Mike	400	2010/12/30 15:06:00
		C	Ellen	200	2010/12/30 16:34:00
		D	Sara	200	2011/01/06 09:18:00
		E	Sara	50	2011/01/06 12:18:00
		F	Sean	400	2011/01/06 13:06:00
		C	Pete	100	2011/01/08 11:43:00
		D	Sara	200	2011/01/09 09:55:00
		G	Ellen	200	2011/01/15 10:45:00
4	3	A	Pete	50	2011/01/06 15:02:00
		C	Mike	400	2011/01/07 12:06:00
		F	Sean	100	2011/01/08 14:43:00
		D	Sara	200	2011/01/09 12:02:00
		H	Ellen	200	2011/01/12 15:44:00
5	4	A	Ellen	50	2011/01/06 09:02:00
		B	Mike	400	2011/01/07 10:16:00
		C	Pete	100	2011/01/08 11:22:00
		D	Sara	200	2011/01/10 13:28:00
		E	Sara	200	2011/01/11 16:18:00
		C	Ellen	100	2011/01/14 14:33:00
		B	Mike	400	2011/01/16 15:50:00
		D	Sara	200	2011/01/19 11:18:00
		E	Sara	200	2011/01/20 12:48:00
		B	Sue	400	2011/01/21 09:06:00
		C	Pete	100	2011/01/21 11:34:00
		D	Sara	200	2011/01/23 13:12:00
		H	Mike	200	2011/01/24 14:56:00
6	8	A	Mike	50	2011/01/08 15:02:00
		B	Ellen	100	2011/01/08 16:06:00
		C	Mike	400	2011/01/09 16:22:00
		D	Sara	200	2011/01/09 16:52:00
		G	Mike	200	2011/01/16 11:47:00

Table 2.1: Example event log with 6 cases, adapted from [1, p. 13]

Definition 2 (Case, trace). *Let $\mathcal{C}$ be the set of all possible* cases *(the case universe). Let CA be the set of case attribute names. Equivalently to event attributes, for an attribute $a \in CA$ and case $c \in \mathcal{C}$ the corresponding attribute value of c is denoted by $\#_a(c)$. Again, $\#_a(c) = \perp$ if the attribute is missing.*
We assume the existence of at least one case attribute called trace*, with $\#_{trace}(c) \in \mathcal{E}^*$ being the corresponding sequence of events. We refer to the trace attribute by the shorthand $\hat{c} = \#_{trace}(c)$.*
By A^* we denote the set of finite sequences over the set A, where a sequence is a mapping $\mathbb{N} \to A$. $a \oplus a'$ is the concatenation of two sequences. $hd(a)$ is the first and $lt(a)$ the last element of a sequence (or $\perp$ for empty sequences, which we denote by ϵ). The ordering of a trace is assumed to be consistent with the time attribute of its events, i.e., $\forall c \in \mathcal{C}\ \forall 1 \leq i < j \leq |\hat{c}| : \#_{time}(\hat{c}(i)) < \#_{time}(\hat{c}(j))$. This implies that we do not allow identical timestamps in the same trace.
We also define a relation over traces w.r.t. to their time attributes. Let $t_1, t_2 \in \mathcal{E}^*$ be traces, which might be empty, then:
$t_1 < t_2 \Leftrightarrow \forall 1 \leq i \leq |t_1| \forall 1 \leq j \leq |t_2| : \#_{time}(t_1(i)) < \#_{time}(t_2(i))$.

Definition 3 (Event log). *An* event log *$L \in \mathcal{C}^n$ is an n-dimensional vector of cases. The ordering of cases is usually of no interest and simply a mathematical convenience.*

These definitions summarize our assumptions about the contents of an event log. In the literature it is sometimes assumed that specific changes in data items, like document contents, are present as event attributes, i.e., can be tracked along the process execution. In contrast, we focus only on activities and timestamps. We also do not impose any constraints on trace completeness, i.e., an event log may contain ongoing cases.

Applying these definitions to the example log from fig. 2.1 and denoting by c_i the case c with $\#_{caseID}(c) = i$, we see that:

- $\hat{c}_1 < \hat{c}_6$
- $\hat{c}_1 \not< \hat{c}_4$
- $\#_{act.}(\hat{c}_1(1)) = \#_{act.}(\hat{c}_4(1)) = \text{A}$
- $\#_{time}(\hat{c}_1(4)) = 2011/01/06\ 11{:}18{:}00$
- $\#_{resource}(\hat{c}_1(2)) = \text{Sue}$
- $\#_{cost}(\hat{c}_1(2)) = 400$
- $\#_{caseID}(c_1) = 1$
- $\#_{customerID}(c_1) = 2$

Note that the latter four attributes are optional according to our definition. Technically no explicit *ID* attribute is required if the identity of a case can be uniquely determined from the log, even though such an attribute is usually convenient.

2.2 Process mining

Process mining is an umbrella term for various techniques that are used to extract knowledge about business processes from event logs. Analysis of data from actual process executions should reveal information about the process *as it is* actually implemented as opposed to a potentially idealized conception about *how it should be* implemented.

The applications of process mining are often simply categorized into *discovery*, *conformance checking* and *enhancement*. These categories are somewhat limited and mostly centered around executable process models. Therefore, we adopt the "refined process mining framework" by van der Aalst [1, p. 242] as depicted in fig. 2.1. Here the activities are categorized into *cartography*, *auditing* and *navigation*, which we will briefly explain in the following to provide some context for our own work.

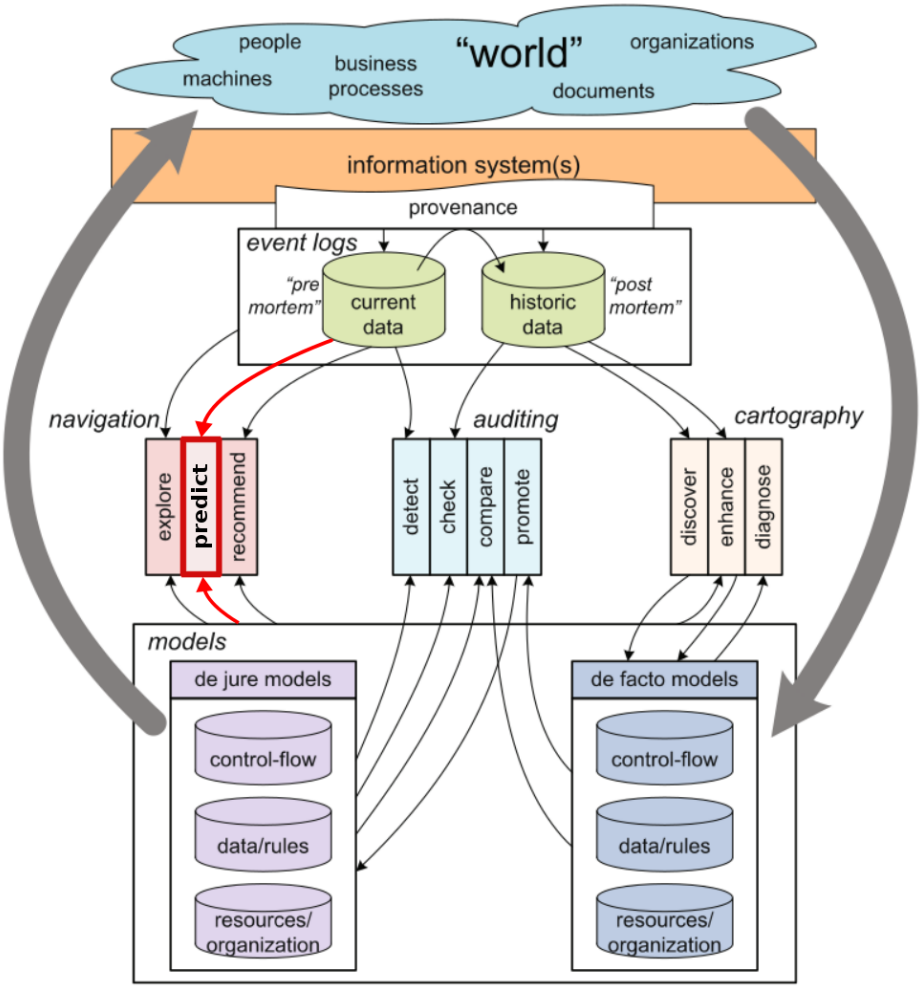


Figure 2.1: Refined process mining framework by van der Aalst [1, p. 242]. The main goal of our work, framed in red, is to provide predictions for currently running ("pre mortem") cases

2.2.1 Cartography

Cartography encompasses the extraction and improvement of (interpretable) *de facto* process models from historic event logs.

Starting with nothing but an event log, process **discovery** aims to find a process model given example executions. Most discovery algorithms deal with the workflow perspective, i.e., events are projected on their $\#_{act.}$ attributes. Availability of such a workflow model is often assumed as the basis for further process mining activities, therefore we go into a little more detail.

There is a variety of workflow mining algorithms, the simplest one being the α-algorithm [5]. This algorithm mines a Petri net system based on the *directly-follows* relation of events in the log. However, the α-algorithm is mainly of academic interest and has several shortcomings which have been adressed by its successors, some examples of which are listed below:

- the *fuzzy miner* [47] abstracts from irrelevant edges and aggregates activities into clusters in the dependency graph which can be used as a basis for other discovery algorithms
- *genetic* process discovery algorithms [2] employ an evolutionary approach to find the "best" process model out of many candidates
- *region-based* approaches construct Petri net systems by adding places that are feasible according to *language-based regions* present in the log. This can be expressed as an integer linear programming problem [94, 100]
- the *Heuristic Miner* [93] constructs a *Causal net* instead of a Petri net system based on several (local and global) dependency measures
- the *Inductive Miner* and its extensions [58, 60, 59, 61] construct sound block-structured workflow nets via the intermediate representation of a process tree

There are many more discovery algorithms and their applicability depends on the required output in terms of representation language (Causal net, Petri net system, BPMN model, etc.) and quality criteria. Criteria typically used to judge a mined workflow model are fitness, simplicity, precision and generalization [24]. Other perspectives beside workflow can also be of interest: for example, projecting the events on their $\#_{resource}$ attribute, techniques have been proposed to discover interactions, e.g., in the form of a *social network* [4].

If a process model already exists, the results can be used for its **enhancement**. Given a workflow model it could be extended with information present in the log. In the case of organizational information, this could result in a model such as the one in fig. 1.1.

Diagnosis refers to classical model-based analysis, e.g., checking the soundness of a process model. This activity is in general independent of event logs, even though it can be used to evaluate process models that have been mined from these logs.

2.2.2 Auditing

Auditing assumes the existence of *de jure* models, i.e., models that are normative and currently assumed to be correct. These models can be **compared** to discovered *de facto* models, in order to detect deviations from reality.

The results of this comparison can serve as an input for improvement of de jure models by **promoting** parts of the de facto models.

Normative models can also form the basis to check conformance of process executions. This can be done either in an online (**detect**) or offline (**check**) setting. In the online case, measures can be taken in order to enforce the prescribed execution of the case. Detection can be directed towards many properties, such as workflow constraints (e.g., adherence of a two-man-rule) or service level agreements. Conformance measures are often either based on log replay [82, 69] or log / model alignments [8] and therefore assume high quality process models as well as a tight connection between models and event logs.

2.2.3 Navigation

Navigation techniques are targeted at running process instances and can potentially influence their future.

Exploration of running cases is a manual activity which can be supported by visualizations based on process models or comparison to historical cases.

Predictions about running cases are supposed to uncover some yet unknown property. This might mean an attempt to forecast the future, but could also be some characteristic in the present state of the process that should be automatically detected.

These predictions can then be used to make **recommendations**, e.g., which activity should be executed next in order to achieve some goal.

Note that the latter two activities stand out, in that they do not necessarily require a human readible process model. In the case of cartography, providing such a model is the goal in the first place. For auditing, a normative model has to be provided. However, in order to make predictions, this is not necessary as long as a meaningful prediction target can be determined. This stands in contrast to the L* life-cycle model for a process mining project described in the process mining manifesto [6], where "operational support" is just the last stage after an integrated process model has been created. We have already made this point in the introduction in a more abstract fashion.

2.3 Predictive monitoring

By predictive monitoring, we aim to provide operational support for running process instances. One can envision various kinds of predictions (which we briefly discuss in chapter 3), but we focus on the *classification* of running cases, which is a quite general framework. This leads us to the following problem formulation.

If $\mathcal{Y}$ is the finite set of possible case labels, we are looking for a function (or hypothesis):

$$h : \mathcal{C} \to \mathcal{Y}$$

In the case of binary classification there are only two labels $\mathcal{Y} = \{-1, +1\}$ and we will confine ourselves to this simple case in this thesis, since most results can be extended to the multi-class case. Such a binary label could capture any boolean statement about the case, for instance:

- occurence or non-occurence of a particular event at some point in the future
- achievement or non-achievement of business goals, such as time or cost constraints
- a category such as "normal" or "anomalous"

In particular, we try to automatically induce a good hypothesis in a *supervised machine learning* setting, i.e., we are given a training event log $L_{tr} = (c_1, \dots c_n)$ of size n as well as the corresponding labels $Y_{tr} = (y_1, \dots, y_n)$.

A hypothesis is considered good if it has low expected classification error over the distribution of all possible cases. Since we usually do not have access to this distribution, the error is estimated on a hold out test set L_{te} instead, whose labels Y_{te} are unknown to the classifier. In theory, it is usually assumed that all samples are drawn i.i.d. from the *same* distribution, but in practice this assumption is frequently violated, e.g., in the presence of concept drift (see section 2.5.3).

If we want to present a case as an example to a learning algorithm, we have to encode its attributes in a form suitable to this particular algorithm. The case attributes except the trace can usually be encoded as a real-valued vector, e.g., by using one-hot encoding for nominal attributes and a direct encoding of numerical attributes. In contrast, the *trace* attribute is not a vector but a *sequence* of events and thus requires special treatment. Following Xing et. al [99], we distinguish three approaches to sequence classification:

- the feature-based approach
- the distance-based approach
- the model-based approach

which we will explain in the following.

2.3.1 Sequence classification

Feature-based approach

When a sequence is converted into a feature vector, many conventional machine learning algorithms, such as decision tree learners or logistic regression, can be applied to it. We call the corresponding vector space the *input space* $\mathcal{X}$. In the case of event log traces, the raw input data has to be converted to this space by a *feature map*:

$$\psi_{trace} : \mathcal{E}^* \to \mathcal{X}$$

There are countless ways to extract features from a sequence, that are highly domain specific. In text mining, k-grams of words have been successfully used to capture local language effects in the context of a word. An element in the feature vector can then encode the presence or absence of a k-gram or the number of occurences. Another approach would be to consider only *frequent* patterns in a sequence. Both approaches have in common that they aggregate information on the level of subsequences, i.e., local properties. In contrast, features can also represent global properties of the sequence, such as its length or duration.

Distance-based approach

Other classification algorithms rely on a distance function. A distance measure for traces would be a metric of the form:

$$d : \mathcal{E}^* \times \mathcal{E}^* \to [0, \infty)$$

Well-known distance measures for simple symbolic sequences are Edit oder Hamming distance. Other distance measures are more domain specific and employ, for example, sequence alignments. In the case of time series data, techniques like dynamic time warping can be used [12]. Of course, distance-based methods can also be applied if features have been extracted from instances by computing the cosine or Euclidean distance between two feature vectors. A popular and very simple distance-based classification algorithm is k nearest neighbours (see, for example [52, p. 14]).

Somewhat related are *kernel* methods that implicitly map instances to a high (or even infinite) dimensional feature space. They are applicable when the learning algorithm only requires the computation of inner products between instances. Then, instead of operating in the high-dimensional space, the algorithm only needs to compute a (positive definite) kernel function for any two instances. Specific kernel functions have been derived for structured data such as symbolic sequences or even graphs [45]. The most famous kernel based learning algorithm is probably the support vector machine (SVM). Again, if a feature representation is available, standard kernels like the Gaussian or polynomial kernel can be applied.

In our application, is not obvious how to come up with a distance measure that captures all aspects (i.e., attributes) of events in a trace. The same applies to kernels

over traces. It might, however, be possible to combine distance measures for different aspects by an ensemble of nearest neighbour classifiers or different kernels via multiple kernel learning.

Model-based approach

Graphical models, that explicitly capture the sequential nature of the data, are a more direct approach to sequence classification. The simplest example of such models are Markov chains. They can be extended to Hidden Markov models (HMM) [78], where the underlying stochastic process is not observable but instead outputs emissions with a certain probability. A generalization are even more flexible probabilistic graphical models, such as Dynamic Baysesian Networks [71].

The probability of an unseen observation sequence given the model can be used for classification. If we wanted to incorporate the time information associated to events, these models can be extended to a continuous time domain, e.g., as continuous-time Markov chains [10].

It is unclear if the Markov condition is a reasonable assumption in the presence of concurrency. Higher-order Markov models may somewhat alleviate this, but then again the right order must be chosen and more training data is required to choose the parameters of such a higher-order model. Additionally, in the case of HMMs, the number of hidden states needs to be chosen.

Artificial neural networks, especially recurrent neural networks (RNN) have also been applied to sequence classification [54]. An extension that might be more suitable for the possibly long time lags between important events in traces are long short term memory (LSTM) networks [53].

While artifial neural networks have shown remarkable results in many applications, choosing the right architecture in a new domain is challenging. Moreover, training a neural network is usually costly in terms of the amount of required training data and computation time. In addition, lacking a good baseline, it is hard to justify such a complex solution over a simple one.

The category of model-based classification also includes approaches that are based on explicit workflow models, such as Petri net systems. We can think of measures (such as fitness), that allow to classify traces into the ones that are matched by the model and the ones that are not, or more sophisticated time-based or probabilistic extensions of workflow models.

Model-based approaches are particularly interesting when they are *generative*, e.g., model a probability distribution of input and output variables and can thus be used to sample data from the input distribution. In contrast, discriminative models only provide us with the posterior distribution or even a mere discriminant function without probabilities [13, p. 43]. In fact, the way we formulated the classification problem, the latter would be sufficient.

2.3.2 Problem formulation

Trading off the different options, we decided to focus on feature-based classification at the first attempt. We consider this to be the easiest way to incorporate business process specific aspects into the learning task. Also, we would like to have a baseline for our specific problems that allows us to evaluate more complex approaches in future work. Existing implementations of distance- and model-based approaches are briefly described in the chapter 3.

Feature-based classification has another advantage: it allows us to treat a trace and the other case attributes uniformly (for both feature vectors can simply be concatenated). Equivalently to the trace feature map ψ_{trace} we call the corresponding feature map for the other case features ψ_{case}.

We can now state our problem more precisely:

Problem statement. *Given a training event log with n cases $L_{tr} \in \mathcal{C}^n$ and corresponding labels $Y_{tr} \in \mathcal{Y}^n$, find:*

1. *a feature map $\psi : \mathcal{C} \to \mathcal{X}$, $c \mapsto (\psi_{case}(c), \psi_{trace}(\hat{c}))$*
2. *a classifier $g : \mathcal{X} \to \mathcal{Y}$*

such that the hypothesis $h : \mathcal{C} \to \mathcal{Y}$, $c \mapsto g(\psi(c))$ has low classification error on unseen cases.

Since the encoding of case features ψ_{case} should be straightforward and standard induction algorithms exist to find g given ψ, the main task remains to find a suitable mapping ψ_{trace}, that captures important aspects of the trace. We will refer to the resulting feature vector as a *trace profile* and present some suggestions for such profiles from the literature in the next chapter, along with related work on predictive monitoring. Before, we present tools and standards for process mining and point to some typical problems that frequently arise when dealing with event logs. Later, we will see how they become manifest in the context of predictive monitoring and how they can be tackled.

2.4 Tools and standards

2.4.1 XES

XES is a standard for event logs that defines a meta model as well as an XML based representation. OpenXES [48] is the Java based reference implementation of this standard. The meta model is depicted in fig. 2.2.

We see that the structure closely resembles our event log definition, simplifying the mapping of theoretical concepts to concrete implementations. As a minor difference, the XES has no notion of a *case* but all case attributes are attached to a *trace* object. From a conceptual point of view, this does not make a difference.

The XES standard allows for a convenient exchange of event logs between different tools, such as ProM and Disco.

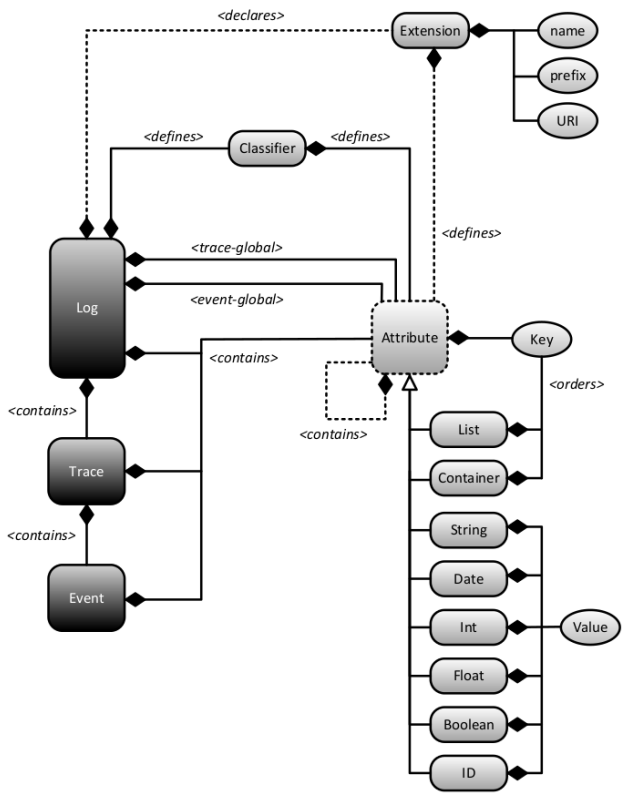


Figure 2.2: XES meta model, from [49]. Classifiers are functions determining the identity of a model element, e.g., if a case is uniquely identified by a combination of customer ID and start date. Extensions define the semantics of model elements.

2.4.2 ProM

ProM [76] is an extensible open source framework for process mining. When mentioning the usage of ProM in the remainer of this thesis, we always refer to the current version 6.6, which is based on XES as opposed to versions before 6.x which employed the older MXML standard. At the time of writing, more than 150 plug-ins are available for ProM, including implementations of all discovery algorithms mentioned in section 2.2.1. A large number of the research results presented in chapter 3 have also led to the implementation of ProM plugins.

2.4.3 Disco

In comparison to ProM, the proprietary tool *Disco* [42] provides a cleaner user interface and better performance but a more limited set of features. For us, Disco has proven useful when performing initial explorative analyses of event logs, including filtering parts of it and obtaining descriptive statistics.

2.5 Challenges

2.5.1 Concurrency

Business processes may allow for the concurrent execution of activities and many possible interleavings of those activities may be observed in the log. Consider the very simple process in fig. 2.3.

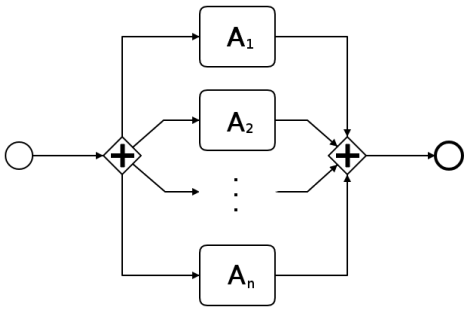


Figure 2.3: A process allowing for the parallel execution of n activities

There are n activities that may be executed in any order, resulting in $n!$ possible interleavings. Even for relatively small values of n one can hardly expect to observe all possible combinations. The situation becomes even worse if instead of single activities the parallel branches consist of composite subprocesses, possibly involving loop constructs. Thus, any process mining approach usually has to adopt weaker notions of completeness. For instance, many process discovery algorithms are based on the directly-follows relation. In the example, completeness w.r.t to the directly-follows relation would require only $n(n-1) \in \mathcal{O}(n^2)$ observations.

2.5.2 Incompleteness and noise

Even with weaker notions of completeness, an example log may still not be complete because the number of possible observations is too large (in the presence of concurrency) or even infinite (in the presence of loops). Events could also be missing due to logging errors.

The opposite problem is *noise*, i.e., the occurence of too much behaviour in the log. Again, this may be due to logging errors such as duplicate log entries or wrong timestamps. Another type of noise are events that are irrelevant for the analysis, either because they are extremely rare or extremely frequent like periodically scheduled events. The impact of noise depends on the use case. When human readible process models shall be discovered, too much behaviour can distract the reader from the essential information. In contrast, when detecting deviations or making predictions, infrequent behaviour may be very relevant.

2.5.3 Concept drift

Business processes may change while they are being analyzed for various reasons. This phenomenon is known as *concept drift* and can manifest in different ways, visualized in fig. 2.4:

(a) Sudden drift: the current process is substituted at one moment by another process

(b) Gradual drift: the current process is substituted but both processes exist in parallel for a certain time

(c) Recurring drift: different processes appear in intervals, e.g., due to seasonal effects

(d) Incremental drift: the current process is substituted in small steps, e.g., due to incremental reengineering

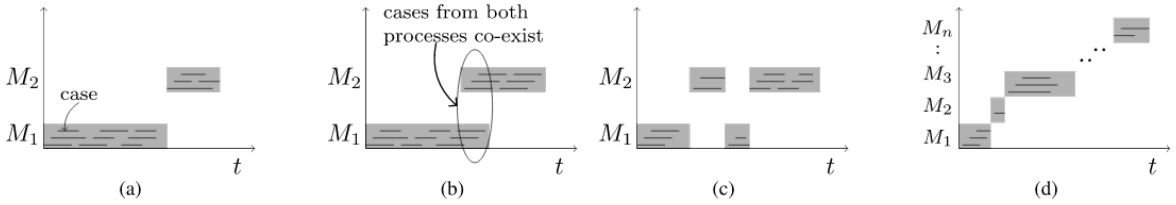


Figure 2.4: Types of concept drift, from [18]

The authors of [18] propose a method to detect concept drift by computing statistically significant differences in "populations" of cases. Concept drift can have a major impact on the generalizability of process mining results to new data. We try to explicitly address this when presenting our experimental results in chapter 7.

3 Related work

We begin with an overview of different trace profiles proposed in the literature. We then discuss previous work on predictive monitoring similar to ours, i.e., implementations of feature-based case classification. Finally, we present examples of other kinds of predictions for business processes apart from case classification.

3.1 Trace profiles

Table 3.1 shows an outline of trace profiles that have been used in various applications. Some of them have been derived in the context of trace clustering, where the goal is to partition an event log into similar cases, often with the goal to discover more meaningful process models for each cluster. The clustering results can therefore be evaluated by the quality of the resulting process model.

The authors of [87] suggest counting various types of event attributes. Note that in our definition of event logs, the activity and originator profiles from [87] are just special instantiations of the event attribute profile. In the case of metric attributes, counting is only possible if the values have been properly discretized. Otherwise, other means of aggregation are necessary, which the authors indicate in the case of timestamps in the performance profile. In [15], single activity- and k-gram-based profiles are discussed in the context of trace clustering. For the former, the authors criticize the loss of context information and execution order. For the latter, they observe that the number of potential features can become very large depending on the value of k and that the right k is hard to choose in practice. Instead, the authors derive distance measures based on Levenshtein (or edit) distance. In [50], a survey of generic outlier detection techniques for temporal data is presented. Among other things, the authors propose breaking time series into fixed length windows. Such approaches have also be applied to event logs, for example by [18]. In [90], different trace clustering techniques are compared, most of which employ abstract trace representations as feature vectors.

3.2 Feature-based case classification

The authors of [83] identify decision points in a Petri net model and use decision trees to model the probability for each choice at these points as a function of case attributes.

An approach to predict binary labels for running cases using decision trees has been proposed by [66]. The labels are business goals expressed as Linear temporal logic formulas over complete traces. The implementation proceeds in two steps: for a running case, a set of complete traces with similar prefixes is determined. From these

Source	Application	Trace profiles	Description
[87]	Trace clustering with standard distance measures	Activity profile (Bag-of-activities)	Number of events that are equivalent according to $\#_{act.}$
		Originator profile	Number of events that are equivalent according to $\#_{resource}$
		Transition profile	Number of bigrams of activity names
		Event attributes	Number of events that are equivalent according to any event attribute
		Performance	Trace size, case duration, minimum / maximum / mean / median time differences between events
[14, 17]	Trace clustering with Euclidean distance	Sequence patterns	Tandem arrays, repeats of activities, alphabet features
[63]	Supervised sequence classification	Frequent subsequences	Sequences should not only be frequent, but also distinctive for at least one class, i.e., kind of supervised feature selection
[44]	Outlier-aware clustering	S-patterns	Frequent activity subsequences with fork / join semantics
[88]	Prediction of overtime risk	Workload	Average workload per resource
[62]	Feature-based classification with Random Forests	Boolean encoding	Set-of-activities
		Frequency-based encoding	Bag-of-activities
		Simple index encoding	Exact sequence of activities, e.g., the nth feature corresponds the the activity that occurs at the nth index in the sequence of activities
		Index latest payload encoding	Simple index encoding + data attributes attached to the last event in this sequence (can be mimicked by case attributes)
		Index-based encoding	Encoding of other event attributes besides activities at their corresponding indexes
		HMM-based encoding	Hidden Markov models over index-based encodings, relative likelihoods of belonging to the positive or negative class are used as features
[91]	Prediction of next event	Path representations	Emphasis on the encoding of parallelism, requires a business process model and relatively strong assumptions to identify which activities belong to parallel paths in this model
[18]	Concept drift detection	Window count	Bag of subsequences in which pairs of activities are followed within windows size l
		J measure	Cross entropy of probabilities that pairs of activities are followed within window of size l

Table 3.1: Examples of trace profiles from the literature

cases, a data snapshot is produced and a decision tree is learned from the data of the labelled, completed traces. Similar prefixes are determined by edit distance of activities and a user defined similarity threshold. Hence, this approach combines distance- and feature-based classification.

The authors of [62] encode workflow and data as described in table 3.1 and use Random Forests for classification. Both [66] and [62] compare the predicitive performance for different lengths of prefixes, i.e., evaluate the impact of earliness of prediction. These ideas are combined in [92], who first cluster similar traces and then use complex symbolic sequence encodings of traces for binary classification. In [32] the issue of choosing hyperparameters in this setup is addressed. An extension to these approaches is proposed by [89], where event attributes containing unstructured text data are encoded using text models. These features are then used along the index-based encodings proposed by [62].

In [16], the sequence and alphabet features described in [14] are used to predict faulty behaviour for an X-ray machine. Beside predictive performance, the authors aim to find signature patterns that are indicative for faulty behaviour. For this reason, they use decision trees and association rules to find simple, interpretable predictive models.

In [67], artifical neural networks are used for case classification. The authors employ features handcrafted for their application, such as "the execution times of the services that are executed up to the [prediction]checkpoint". They compare this machine learning approach with constraint satisfaction and rule-based quality of service aggregation techniques and suggest to combine these approaches for maximum performance.

3.3 Other prediction targets

Even though we limited the scope of this thesis to feature-based case classification, we want to briefly describe other types of predictions that can be made for running cases.

3.3.1 Next event

Given a case with the partial sequence of events $\langle e_1, \ldots, e_t \rangle$ the goal of some approaches is to predict e_{t+1} instead of a label for the whole sequence. This is an instance of the time series prediction problem [33].

The authors of [56] model the probability of executing the next task as a Markov chain. They assume that document contents are assembled along the process and the probability of executing a task depends at any point only on the current state given by the accumulated document content. The transition probabilities are learned with decision trees over these document contents. In order to determine which transitions are possible, the approach relies on existing process discovery techniques.

The work of [22] and [38] is inspired by natural language processing techniques. The former work employs probabilistic finite automata (PFA) of activities to model transition probabilities, while the latter use recurrent neural networks over sequences of activities and sequences of activity-resource pairs.

In [85], recommendations for next activities are made on the basis of some target function. Historical traces are filtered w.r.t. the "support" of executing the particular next activity. The expected target value is computed as the weighted average over the set of completed, supporting traces.

3.3.2 Time

Another interesting question is how long a running process instance will take to finish. The authors of [3] use transition systems of state abstractions. Running instances are mapped to a state and the expected remaining execution time is computed by averaging the remaining execution time of completed cases when they were in the same state. This approach is extended by [43], where cases are clustered using predictive clustering trees before deriving the state transition system. Another extension was proposed by [25], who use more robust state abstractions based on frequent sequences and use nested regression models to predict the remaining time.

A more sophisticated approach to remaining time prediction based on stochastic Petri nets is proposed by [81]. In [80], the authors use time series Petri nets to model the execution time of single activities. This approach is more flexible and allows, for example, the inclusion of seasonal effects.

Deadline transgressions are predicted in [72] using risk indicators based on statistical outlier detection for several timing and workflow aspects of a trace. Another approach to temporal outlier detection using Bayesian networks of activity durations and timestamps is proposed by [79].

3.3.3 Risk

Prediction of deadline transgressions can be seen of a special instance of predicting some sort of risk measure.

In [39], the authors express process models and security breach models by a set of rules and classify traces according to a probabilistic mapping to these models.

The prediction of different risk measures is the goal in [28]. Examples for such risks are time overrun, cost overrun or reputation loss. The authors use regression trees of case attributes and execution history, which includes the bag-of-activities of the trace and last executor of each task. Additionally, the authors try to find an optimal global work item distribution that minimizes the global risk. This problem is formulated as a Mixed-Integer Linear Programming problem. In [30], the authors propose an approach to propagate risks detected on a process instance level to other instances based on a similarity measure. Similar instances are determined using the trace profiles proposed by [87].

Overall process risk is also estimated by [73], using data enriched process models of desirable behaviour and the degree of deviation as an indicator for riskiness.

4 Feature extraction

In this chapter, we present a general framework for feature extraction from traces based on simple mathematical functions and show how existing trace profiles fit in that framework. As a novel instantiation of this framework, we derive a trace profile based on the execution times of high-level structures in a trace using process trees.

As a running example, we use the event log from table 2.1. Again, we refer by c_i to the case c with $\#_{caseID}(c) = i$.

4.1 Decomposition of trace profiles

Recall that our goal is to find a feature map:

$$\psi_{trace} : \mathcal{E}^* \to \mathcal{X}$$

For the remainder of this chapter, assume that $\mathcal{X}$ is a subset of $(\mathbb{R} \cup \{\perp\})^n$, i.e., the feature map constructs real valued feature vectors where entries could be missing. We refer to this set by the shorthand $\hat{\mathbb{R}} = \mathbb{R} \cup \{\perp\}$. How to deal with missing values strongly depends on the learning algorithm and will be dicussed in chapter 5.

We noticed that the ith component of such a feature vector can in many cases be understood as a composition $\psi_{trace}(t)_i = \Gamma_i([\pi_i(s) \mid s \in \sigma_i(t)])$ of the following functions:

- a selection function $\sigma_i : \mathcal{E}^* \to P(\mathcal{E}^*)$
- a projection function $\pi_i : \mathcal{E}^* \to \hat{\mathbb{R}}$
- an aggregation function $\Gamma_i : B(\hat{\mathbb{R}}) \to \hat{\mathbb{R}}$

where $[\pi_i(s) \mid s \in \sigma_i(t)]$ is a multiset of real numbers and $B(\hat{\mathbb{R}})$ is the set of all multisets over $\hat{\mathbb{R}}$. For a multiset s, we denote by $s(x)$ the number of times x occurs in s.

σ_i implicitly defines an equivalence relation over sequences of events, i.e., all sequences contained in the image of σ_i are considered equivalent in that they are aggregated to feature i.

Note that this formulation comprises all possible mappings $\mathcal{E}^* \to \hat{\mathbb{R}}$, e.g., by setting $\sigma(t) = \{t\}$, π to the desired mapping and $\Gamma_i(s) = max(s)$ or any other function which selects the single member of the bag. Thus, we do not lose generality yet can analyze existing trace profiles more precisely.

4.1.1 Applications

Sets and bags of activities

Let $\mathcal{A} = \{a_1, a_2, \ldots, a_n\}$ be the set of activity names. The *set-of-activities* profile indicates if a_i occurs in a trace or not.

$$\begin{aligned} \sigma_i(t) &= \{\langle t(j)\rangle \mid 1 \leq j \leq |t| \wedge \#_{act.}(t(j)) = a_i\} \\ \pi_i(e) &= 1 \\ \Gamma_i(s) &= \begin{cases} 1 & \text{if } |s| > 0 \\ 0 & \text{otherwise} \end{cases} \end{aligned} \tag{4.1}$$

Example: Let $(a_1, a_2, \ldots, a_8) = (A, B, \ldots, H)$. For illustrational purposes, we simultaneously compute the whole feature vector $(\psi_{trace}(\cdot)_1, \ldots, \psi_{trace}(\cdot)_8)$

$$\hat{c}_3 \xrightarrow{\sigma} \begin{pmatrix} \{\langle \hat{c}_3(1)\rangle\} \\ \{\langle \hat{c}_3(2)\rangle\} \\ \{\langle \hat{c}_3(3)\rangle, \langle \hat{c}_3(7)\rangle\} \\ \{\langle \hat{c}_3(4)\rangle, \langle \hat{c}_3(8)\rangle\} \\ \{\langle \hat{c}_3(5)\rangle\} \\ \{\langle \hat{c}_3(6)\rangle\} \\ \{\langle \hat{c}_3(9)\rangle\} \\ \{\} \end{pmatrix} \xrightarrow{\pi} \begin{pmatrix} [1] \\ [1] \\ [1^2] \\ [1^2] \\ [1] \\ [1] \\ [1] \\ [] \end{pmatrix} \xrightarrow{\Gamma} \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 0 \end{pmatrix}$$

The *bag-of-activities* profile indicates the number of times a_i occurs in a trace, with σ_i and π_i being the same as above and:

$$\Gamma_i(s) = s(1) \tag{4.2}$$

Example: Let $(a_1, a_2, \ldots, a_8) = (A, B, \ldots, H)$, then the bag-of-activities feature vector is constructed like this:

$$\hat{c}_3 \xrightarrow{\sigma} \begin{pmatrix} \{\langle \hat{c}_3(1)\rangle\} \\ \{\langle \hat{c}_3(2)\rangle\} \\ \{\langle \hat{c}_3(3)\rangle, \langle \hat{c}_3(7)\rangle\} \\ \{\langle \hat{c}_3(4)\rangle, \langle \hat{c}_3(8)\rangle\} \\ \{\langle \hat{c}_3(5)\rangle\} \\ \{\langle \hat{c}_3(6)\rangle\} \\ \{\langle \hat{c}_3(9)\rangle\} \\ \{\} \end{pmatrix} \xrightarrow{\pi} \begin{pmatrix} [1] \\ [1] \\ [1^2] \\ [1^2] \\ [1] \\ [1] \\ [1] \\ [] \end{pmatrix} \xrightarrow{\Gamma} \begin{pmatrix} 1 \\ 1 \\ 2 \\ 2 \\ 1 \\ 1 \\ 1 \\ 0 \end{pmatrix}$$

We have already pointed out that these kinds of "event attribute" profiles in the terminology of [87] differ only in the type of attribute the events are projected onto.

For example, we can obtain the *originator profile* by setting:

$$\sigma_i(t) = \{\langle t(j)\rangle \mid 1 \leq j \leq |t| \wedge \#_{resource}(t(j)) = r_i\} \tag{4.3}$$

where $r_i \in \mathcal{R}$ is an element of the finite set of resources.

Bag of bigrams

When using bigrams, instead of single events we count the number of occurences of pairs of events. Let $\mathcal{A} \times \mathcal{A} = \{b_1, b_2, \ldots, b_n\} = \{(a_1, a_1), (a_1, a_2), \ldots (a_m, a_m)\}$ be the set of all pairs of activities, then:

$$\begin{aligned} \sigma_i(t) &= \{\langle t(j), t(j+1)\rangle \mid 1 \leq j < |t| \wedge (\#_{act.}(t(j)), \#_{act.}(t(j+1))) = b_i\} \\ \pi_i(e) &= 1 \\ \Gamma_i(s) &= s(1) \end{aligned} \tag{4.4}$$

Example: Let $b_i = (C, D)$

$$\begin{array}{lllllll} \hat{c}_1 & \xrightarrow{\sigma_i} & \{\langle \hat{c}_1(3), \hat{c}_1(4)\rangle\} & \xrightarrow{\pi_i} & [1] & \xrightarrow{\Gamma_i} & 1 \\ \hat{c}_2 & \xrightarrow{\sigma_i} & \{\} & \xrightarrow{\pi_i} & [] & \xrightarrow{\Gamma_i} & 0 \\ \hat{c}_3 & \xrightarrow{\sigma_i} & \{\langle \hat{c}_3(3), \hat{c}_3(4)\rangle, \langle \hat{c}_3(7), \hat{c}_3(8)\rangle\} & \xrightarrow{\pi_i} & [1^2] & \xrightarrow{\Gamma_i} & 2 \end{array}$$

The same logic applies to all kinds of n-grams, which only differ in the subset construction procedure.

Performance profiles

In order to derive a performance profile based on timestamps in the spirit of [87], the timestamps of pairs of activities need to be aggregated. Consider for instance the *maximum time difference* between pairs of activites. Let again $\mathcal{A} \times \mathcal{A} = \{b_1, b_2, \ldots, b_n\}$ be the set of all pairs of activities and $\delta : \mathcal{T} \times \mathcal{T} \to \mathbb{R}$ a function to compute differences in the time domain, e.g., the number of seconds or minutes between two timestamps. We can then define the following profile:

$$\begin{aligned} \sigma_i(t) &= \{\langle t(j), t(k)\rangle \mid 1 \leq j < k \leq |t| \wedge (\#_{act.}(t(j)), \#_{act.}(t(k))) = b_i\} \\ \pi_i(e) &= \delta(\#_{time}(hd(e)), \#_{time}(lt(e))) \\ \Gamma_i(s) &= max'(s) \end{aligned} \tag{4.5}$$

where the max' function needs to be defined also for empty multisets:

$$max'(s) = \begin{cases} max(s) & \text{if } |s| > 0 \\ \perp & \text{otherwise} \end{cases} \tag{4.6}$$

Example: Let $b_i = (B, C)$ and $\delta(\cdot, \cdot)$ the difference in *days*, then:

$$\begin{array}{lllllll} \hat{c}_3 & \xrightarrow{\sigma_i} & \{\langle \hat{c}_3(2), \hat{c}_3(3)\rangle, \langle \hat{c}_3(2), \hat{c}_3(7)\rangle\} & \xrightarrow{\pi_i} & [0,9] & \xrightarrow{\Gamma_i} & 9 \\ \hat{c}_2 & \xrightarrow{\sigma_i} & \{\} & \xrightarrow{\pi_i} & [] & \xrightarrow{\Gamma_i} & \perp \end{array}$$

We could also be interested in the executions of a single activity, for instance, the *last time* an activity was exectued. Here, we need an additional function $\vartheta : \mathcal{T} \to \mathbb{R}$ that converts single timestamps into real numbers, e.g., seconds since the epoch (Unix time). For activity a_i, this would correspond to:

$$\begin{aligned} \sigma_i(t) &= \{t(j) \mid 1 \leq j \leq |t| \wedge \#_{act.}(t(j)) = a_i\} \\ \pi_i(e) &= \vartheta(\#_{time}(tl(e))) \\ \Gamma_i(s) &= max'(s) \end{aligned} \tag{4.7}$$

Index-based encodings

The *simple index* encoding proposed by [62] can be realized by a one-hot encoding. Let i be the index and a_j an activity from $\mathcal{A}$, then:

$$\begin{aligned} \sigma_{j+(i-1)|\mathcal{A}|}(t) &= \{t(i) \mid \#_{act.}(t(i)) = a_j\} \\ \pi_{j+(i-1)|\mathcal{A}|}(e) &= 1 \\ \Gamma_{j+(i-1)|\mathcal{A}|}(s) &= s(1) \end{aligned} \tag{4.8}$$

The same could be achieved by projecting the events onto other attributes, as the authors have proposed in the *indexed-based encoding*.

Discretization

Discretization of numeric attributes can also be achieved by the selection function. Let v be some attribute with domain $\mathbb{R}$ and $d : \mathbb{R} \to \mathbb{N}$ a discretization function that maps each value to one of n bins, then for the ith bin:

$$\sigma_i(t) = \{t(j) \mid 1 \leq j \leq |t| \wedge d(\#_v(e)) = i\} \tag{4.9}$$

Discretization in the time domain can be realized by constructing *time windows*. Let $s \in \mathcal{T}$ be the starting point for the windowing, w be the window size and b the offset by which the window is moved.

We can construct the set of all subsequences within the ith time window $[s+ib, s+ib+w)$ by setting:

$$\begin{aligned}\sigma_i(t) = \{\langle t(s_1)\ldots,t(s_m)\rangle \mid\ & 1 \leq s_1 < \ldots < s_m \leq |t| \\ & \wedge \forall j : t(s_{j+1}) = t(s_j + 1) \\ & \wedge \#_{time}(t(s_1)) \geq s + ib \\ & \wedge (t(s_1 - 1) = \perp \vee \#_{time}(t(s_1 - 1)) < s + ib) \\ & \wedge \#_{time}(t(s_m)) < s + ib + w \\ & \wedge (t(s_m + 1) = \perp \vee \#_{time}(t(s_m + 1)) \geq s + ib + w)\}\end{aligned} \tag{4.10}$$

4.1.2 Subsequence construction

We see that we can construct various trace profiles by combining different functions σ, π and Γ. While π and Γ are somewhat generic, it is not obvious what the right scope of σ is, i.e., which subsequences should be considered equivalent.

In our definition of event logs, we only assumed timestamps and activity names to be present as event attributes. We already presented the time window approach as a means to provide a subset mapping by timestamps.

Let us now take a closer look at equivalence classes over activities. In the examples, we have seen that equivalence can be determined on the level of single activities (4.1) or n-grams of activities (4.4). When constructing bigrams, we considered subsequences of adjacent events $t(i), t(i+1)$ to be equivalent if they corresponded to the same ordered pair of activities $(\#_{act.}(t(i)), \#_{act.}(t(i+1))) = (a_1, a_2)$.

But why should this be a meaningful abstraction in the domain of business processes? For instance, if a_1 and a_2 can be executed concurrently, it may not be of interest whether a_1 was executed before or after a_2 and we could instead consider *unordered* pairs $\{a_1, a_2\}$ for determining equivalence.

Then again, why should we look only at adjacent events? In example 4.5 the time differences of any two events corresponding to a pair of activites are computed, provided that they occurred somewhere in the trace in the correct order. Here we could also consider *unordered* pairs of activites, but if two activities can be executed concurrently, does it even make sense to compute time differences between their execution?

We see that when constructing features based on activities, we are faced with two problems:

1. The number of possible equivalence relations is extremely large. If m is an upper bound on the length of a trace and n the number of activities, there are $(n+1)^m$ potential, not necessarily contiguous, activity sequences. Out of these, we can construct $2^{((n+1)^m)}$ subsets.

2. The class of meaningful projections and aggregations depends on the semantics of the equivalence relation.

4.2 Tree-based trace profiles

To solve the aforementioned problems, we propose a heuristic that allows to construct meaningful sets of subsequences along with semantics for their interpretation.

The algorithm proceeds in the following steps:

1. Mine a *process tree* from all available traces
2. For each node v_i in the tree, define $\sigma_i(t)$ as the set of all subsequences in t that are *matched* by node i

In the following, we define what a process tree is and what it means for a sequence of events to be matched by one of its nodes.

4.2.1 Process trees

Process trees are rooted trees whose leaves correspond to single *activities* and all other nodes are *operators*.

We formally define process trees closely following [59]. Let $\bigotimes$ be the set of tree operators, $\mathcal{A}$ the set of activities and $\tau \notin \mathcal{A}$ the *silent activity*. We assume that there are no duplicate activities in the tree (except for the silent activity).

We recursively define:

- $a \in \mathcal{A} \cup \{\tau\}$ is a process tree
- if $M_1, \dots M_n$ with $n > 0$ are process trees and $\otimes \in \bigotimes$ is an operator, then $\otimes(M_1, \dots M_n)$ is a process tree

The *language* $\mathcal{L}$ defined by a process tree M are the sequences of activities that are accepted by the tree. The operators have corresponding language join functions $\otimes_l$ that describe how the languages of its children are to be combined, thus defining the semantics of the operator. This leads us to the following identities:

$$\begin{aligned} \mathcal{L}(a) &= \{\langle a \rangle\} \text{ for } a \in \mathcal{A} \\ \mathcal{L}(\tau) &= \{\epsilon\} \\ \mathcal{L}(\otimes(M_1, \dots, M_n)) &= \otimes_l(\mathcal{L}(M_1), \dots, \mathcal{L}(M_n)) \end{aligned}$$

The standard operators defined by [59] are introduced below:

$\rightarrow$ is the *sequential execution* operator with

$$\rightarrow_l (L_1, \dots, L_n) = \{t_1 \oplus t_2 \oplus \dots \oplus t_n \mid \forall i \in 1 \dots n : t_i \in L_i\}$$

$\times$ is the *exclusive choice* operator with

$$\times_l (L_1, \ldots, L_n) = \bigcup_{1 \leq i \leq n} L_i$$

$\circlearrowleft$ is the *structured loop* operator with

$$\circlearrowleft_l (L_1, \ldots, L_n) = \{t_1 \oplus t'_1 \oplus t_2 \oplus t'_2 \oplus \ldots \oplus t_m \mid \forall i : t_i \in L_1 \wedge t'_i \in \bigcup_{2 \leq j \leq n} L_j\}$$

$\wedge$ is the *parallel execution* operator with

$$\wedge_l (L_1, \ldots, L_n) = \{t \mid t \in \{t_1, \ldots, t_n\}_{\simeq} \wedge \forall i : t_i \in L_i\}$$

where $\{t_1, \ldots, t_n\}_{\simeq}$ is the set of all interleavings of traces $t_1 \ldots t_n$. We refer to [59] for the exact definition, because it is a bit cumbersome to express in predicate logic.

Fig. 4.1 shows an example of a process tree whose language includes all traces in example log 2.1.

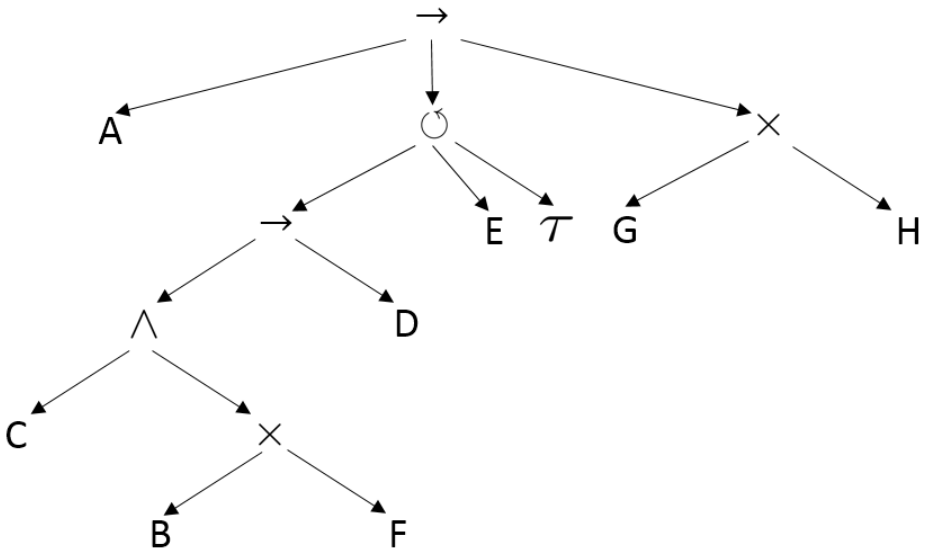


Figure 4.1: A process tree whose language includes all the traces from the example log in table 2.1

A process tree can be converted to a sound, block-structured workflow net. However, we are not interested in this workflow net but instead use the tree structure to find matching subsequences of events.

There are several algorithms to construct a process tree given an event log, for example the evolutionary tree miner [23] or the inductive miner and its variants [58, 60, 59, 61]. We briefly describe the latter at the end of this chapter.

Add link to implementation (footnote)

4.2.2 Matching algorithm

In listing 4.1 we present an algorithm that constructs sets of event sequences that are matched by tree nodes.[1]

For simplicity, we assume the existence of the following utility functions:

MinePROCESSTREE(L, θ)
: Returns a process tree mined from log L with parameters θ.

GETNODES(P)
: Returns all nodes of the process tree P.

GETROOT(P)
: Returns the root node of the process tree P.

GETCHILDREN(v, P)
: Returns all child nodes of node v in process tree P.

ISLEAF(v, P)
: Returns *true* if v is a leaf node in process tree P, $false$ otherwise.

GETACTIVITY(v, P)
: Returns the activity $a \in \mathcal{A} \cup \{\tau\}$ of a leaf node v in process tree P.

GETOP(v, P)
: Returns the operator $\otimes \in \bigotimes$ of an inner node v in process tree P.

The algorithm takes as parameters an event log L and optionally parameters for the mining algorithm θ or an existing process tree P. An existing process tree may be provided if it was learned from a training event log L_{tr} and shall be applied to the test log L_{te}. Reusing the same tree is important, because all cases have to be mapped to the same feature space.

When a process tree has been supplied or mined from the log, the sequences of events matching the nodes of the tree are determined for each trace. The algorithm recursively combines all subsequences matched by the children of a node. The way these subsequences are combined are specific to the operators and we will discuss the combination routine for each of them in the following. Note that the algorithm handles only the standard operators as defined in section 4.2.1, but could be easily extended to custom operators.

The combination procedures would be trivial if the process tree perfectly fitted all traces, but this can neither be assumed for the traces in the training log and even less for unseen cases. Therefore, we discuss our considerations regarding these cases as well.

[1] An implementation of the algorithm in the Scala programming language is avaiable at **TODO**

Listing 4.1 Subsequence construction from process trees

```
function ConstructFeatures(L, θ, P)
    if P = ⊥ then                                   ▷ no existing tree provided
        P ← MineProcessTree(L, θ)                   ▷ e.g., with the inductive miner
    end if
    for t ← L do
        for v_i ← GetNodes(P) do
            σ_i(t) = {}                             ▷ initialize empty sets of subsequences
        end for
        v ← GetRoot(P)
        Match(v, t, P)
    end for
    return (σ, P)
end function

procedure Match(v_i, t, P)
    if IsLeaf(v_i, P) then
        if GetActivity(v_i, P) = τ then
            σ_i(t) = {ϵ}
        else
            σ_i(t) = {t(j) | 1 ≤ j ≤ |t| ∧ GetActivity(v_i, P) = #_act.(t(j))}
        end if
    else
        ⟨u_{j_1}, ..., u_{j_m}⟩ ← GetChildren(v_i, P)
        for j ← j_1 ... j_m do
            Match(u_j, t, P)
        end for
        ⊗ ← GetOp(v_i, P)
        if ⊗ = × then
            σ_i(t) = combineXor(σ_{j_1}(t), ..., σ_{j_m}(t)))
        else if ⊗ = ↺ then
            σ_i(t) = combineLoop(σ_{j_1}(t), ..., σ_{j_m}(t))
        else if ⊗ = → then
            σ_i(t) = combineSequence(σ_{j_1}(t), ..., σ_{j_m}(t)))
        else if ⊗ = ∧ then
            σ_i(t) = combineAnd(σ_{j_1}(t), ..., σ_{j_m}(t)))
        end if
    end if
end procedure
```

Listing 4.2 Combining matches in an exclusive choice node

function CombineXor($\sigma'_1, \ldots, \sigma'_m$)
  return $\bigcup_{1 \leq i \leq m} \sigma'_i$
end function

Listing 4.2 shows how subsequences are combined under the $\times$ operator. This is very easy: the subsequences matched by a $\times$ node are simply all subsequences that are matched by any of its children.

Listing 4.3 Combining matches in a structured loop node

function CombineLoop($\sigma'_1, \ldots, \sigma'_m$)
  return $\{t_1 \oplus t_2 \mid t_1 \in \sigma'_1 \wedge t_2 \in \bigcup_{2 \leq i \leq m} \sigma'_i \cup \{\tau\}$
  $\wedge \nexists t' \in \bigcup_{2 \leq i \leq m} \sigma'_i : |t'| > 0 \wedge t' > t_1 \wedge t' < t_2\}$
end function

The loop scenario in listing 4.3 is more tricky. One could think that a $\circlearrowleft$ node should match all of its repetitions, but if there was another $\circlearrowleft$ somewhere up in the tree, we cannot know where to stop matching in the current iteration.

Instead, the algorithm matches each single iteration of the loop seperately and combines the iterations at upper nodes. Additionally, this allows us to count the number of loop repetitions.

Consider the example tree $\circlearrowleft (\circlearrowleft (A, B), C)$ and the simplified trace (the numbers in the subscript denote the index of the corresponding events):

$$t = \langle A_1, B_2, A_3, C_4, A_5, C_6, A_7, B_8, A_9, D_{10} \rangle$$

The inner node $\circlearrowleft (A, B)$ matches $\{\langle A_1, B_2 \rangle, \langle A_3 \rangle, \langle A_5 \rangle, \langle A_7, B_8 \rangle, \langle A_9 \rangle\}$, the outer node combines these matches into $\{\langle A_1, B_2 \rangle, \langle A_3, C_4 \rangle, \langle A_5, C_6 \rangle, \langle A_7, B_8 \rangle, \langle A_9 \rangle\}$.

The combination of subsequences under the $\rightarrow$ operator (shown in listing 4.4) needs to be understood in terms of possible loop repetitions.

The rather complicated looking last three parts of the conjunction ensure that the sequence matches all repetitions of its children as long as they do not interfere with the other parts of the sequence:

$$\begin{aligned} &\forall i, j, i', j' : i' = i + 1 \wedge j' = 1 \vee i = i' \wedge j' = j + 1 \\ &\quad \Rightarrow t_{i,j} < t_{i',j'} \wedge \nexists t' \in \sigma'_i \cup \sigma'_{i'} : |t'| > 0 \wedge t' > t_{i,j} \wedge t' < t_{i',j'} \end{aligned}$$

ensures that the items of the sequence are in the right order and that inserting

Listing 4.4 Combining matches in a sequential execution node

function CombineSequence($\sigma'_1, \ldots, \sigma'_m$)
return $\{t_{1,1} \oplus t_{1,2} \oplus \ldots \oplus t_{2,1} \oplus \ldots \oplus t_{m,n} \mid$

$$\begin{aligned}
&\forall i, j : t_{i,j} \in \sigma'_i \\
\wedge &\forall i, j, i', j' : i' = i + 1 \wedge j' = 1 \vee i = i' \wedge j' = j + 1 \\
&\quad \Rightarrow t_{i,j} < t_{i',j'} \wedge \nexists t' \in \sigma'_i \cup \sigma'_{i'} : |t'| > 0 \wedge t' > t_{i,j} \wedge t' < t_{i',j'} \\
\wedge &\nexists t'_1 \in \sigma'_1 : t'_1 < t_{1,1} \wedge \forall t' \in \bigcup_{2 \leq j \leq m} \sigma'_j : |t'| = 0 \vee t' < t'_1 \vee t' > t_{m,n} \\
\wedge &\nexists t'_m \in \sigma'_m : t'_m > t_{m,n} \wedge \forall t' \in \bigcup_{1 \leq j < m} \sigma'_j : |t'| = 0 \vee t' < t_{1,1} \vee t' > t'_m \}
\end{aligned}$$

end function

another valid subsequence is not possible.

$$\nexists t'_1 \in \sigma'_1 : t'_1 < t_{1,1} \wedge \forall t' \in \bigcup_{2 \leq j \leq m} \sigma'_j : |t'| = 0 \vee t' < t'_1 \vee t' > t_{m,n}$$

ensures that no subsequence can be appended to the left and

$$\nexists t'_m \in \sigma'_m : t'_m > t_{m,n} \wedge \forall t' \in \bigcup_{1 \leq j < m} \sigma'_j : |t'| = 0 \vee t' < t_{1,1} \vee t' > t'_m$$

ensures that no subsequence can be appended to the right.

As an example, consider the tree $\rightarrow (\circlearrowleft (\circlearrowleft (A, B), C), D)$ and trace t as before.The sequential operator would concatenate all matches of its left child and match the entire sequence.

Note that this implies that the algorithm might also match some subsequences that are not exactly part of the tree's language. For instance, the sequence $\langle A, A, B \rangle$ is matched by the tree $\rightarrow (A, B)$. For our purposes, this is acceptable, since we are not interested in exact alignment of the model and the trace but use the model as an heuristic to find related events.

This is even more of an issue when dealing with parallel executions (listing 4.5). Consider the tree $\wedge(A, \times(B, \tau))$ and trace $\langle A_1, B_2, A_3 \rangle$. Whether we match $\{\langle A_1, B_2 \rangle, \langle A_3 \rangle\}$ or $\{\langle A_1 \rangle, \langle B_2, A_3 \rangle\}$ is arbitrary.

We propose a greedy approach that collects subsequences in ascending time order and records a match as soon as at least one subsequence from each child is found. This may not yield perfect results, as we can see for the tree $\wedge(A, B, \times(C, \tau))$ and the sequence $\langle A_1, B_2, C_3, A_4, B_5, C_6 \rangle$. The algorithm would return $\{\langle A_1, B_2 \rangle, \langle C_3, A_4, B_5 \rangle, \langle C_6 \rangle\}$, which does not exactly conform to the operator's language, but again, we consider this acceptable for the purpose of feature extraction. A more sensible approach could be based on trace alignments.

Listing 4.5 Combining matches in a parallel execution node

function CombineAnd($\sigma'_1, \ldots, \sigma'_m$)
  $\sigma'' \leftarrow \{\}$
  $\langle t_1, \ldots, t_n \rangle \leftarrow$ sort $\bigcup_{1 \leq i \leq m} \sigma'_i \setminus \{\epsilon\}$ ascending by $t \mapsto \#_{time}(lt(t))$
  $currset \leftarrow \{\epsilon\}$
  for $t \leftarrow \langle t_1, \ldots, t_n \rangle$ **do**
    $currset \leftarrow currset \cup \{t\}$
    if $\forall i : \sigma'_i \cap currset \neq \varnothing \wedge \exists i : \sigma'_i \cap currset \neq \{\epsilon\}$ **then**
      $\sigma'' \leftarrow \sigma'' \cup currset$
      for $i \leftarrow 1 \ldots m$ **do**
        $\sigma'_i \leftarrow \sigma'_i \setminus currset$
      end for
      $currset \leftarrow \{\epsilon\}$
    end if
  end for
  return $\sigma'' \cup (currset \setminus \{\epsilon\})$
end function

4.2.3 Example: temporal profile

To illustrate how the algorithm works, we show how a profile based on timestamps can be constructed by this approach. Let $\delta(\cdot, \cdot)$ denote the difference in whole, rounded up *days* between two timestamps and:

$$\pi_i(e) = \delta(\#_{time}(hd(e)), \#_{time}(tl(e))$$
$$\Gamma_i(s) = mean'(s)$$

We construct σ for the trace shown in table 4.1 and the process tree from fig. 4.1. The corresponding trace profile is depicted in table 4.2. This example illustrates how the algorithm not only computes the duration of the whole trace, but also durations for possibly interesting subprocesses.

We see that computing time differences for nodes containing single activities or $\times$ nodes over single activities is obviously useless. But since we can leverage the semantics of these nodes when constructing the trace profile, we just do not have to include these items in the feature vector in the first place. Hence, the final feature vector would contain only the blue shaded values. This is an advantage over generic profiles based on bigrams or frequent patterns.

Interestingly, we cannot derive a duration for E_5 on its own but we see its effect when relating it to the substructure it is part of, resulting in the differences between item 12 and 13.

Case ID	Row ID	Activity	Resource	Cost	Timestamp
7	1	A	Ellen	50	2011/01/06 09:02:00
	2	B	Mike	400	2011/01/07 10:16:00
	3	C	Pete	100	2011/01/08 11:22:00
	4	D	Sara	200	2011/01/10 13:28:00
	5	E	Sara	200	2011/01/11 16:18:00
	6	C	Ellen	100	2011/01/14 14:33:00
	7	B	Mike	400	2011/01/16 15:50:00
	8	D	Sara	200	2011/01/19 11:18:00
	9	F	Sue	400	2011/01/21 09:06:00
	10	C	Pete	100	2011/01/21 11:34:00
	11	D	Sara	200	2011/01/23 13:12:00
	12	H	Mike	200	2011/01/24 14:56:00

Table 4.1: Another example trace similar to the event log in table 2.1. The row ID was added for notational convenience.

4.2.4 Inductive Miner

We have not yet described how a process tree is actually created from an event log. We therefore briefly describe the Inductive Miner (IM) [59], which we have used as the process tree mining algorithm in our experiments (chapter 7).

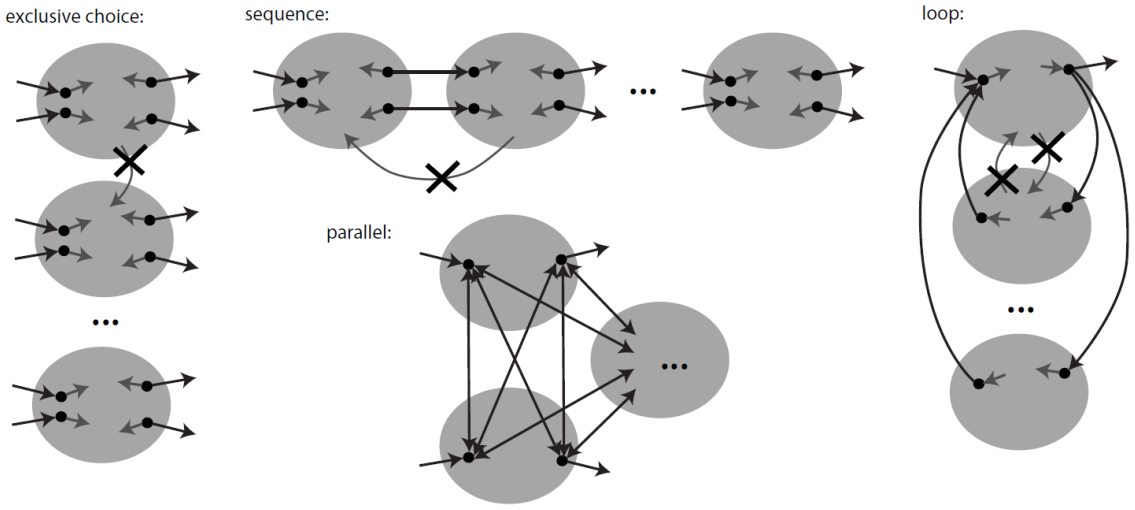


Figure 4.2: Cuts of directly-follows graph corresponding to tree operators, from [59]

The algorithm considers the directly follows graph G of activities. The vertices of G are single activities. An edge (a, b) in the graph is present, if b directly follows a in some trace in the event log. The set of nodes is recursively split into subsets $\Sigma_1 \dots \Sigma_n$ according to the semantics of the tree operators defined above. The cuts in the directly follows graph corresponding to tree operators are sketched in fig. 4.2. The log is split into sub logs according to $\Sigma_1 \dots \Sigma_n$ and the algorithm recurses on

i	v_i	$\sigma_i(t)$	$p = [\pi_i(s) \mid s \in \sigma_i(t)]$	$\Gamma_i(p)$
1	A	$\{\langle A_1 \rangle\}$	$[0]$	0
2	B	$\{\langle B_2 \rangle, \langle B_7 \rangle\}$	$[0^2]$	0
3	C	$\{\langle C_3 \rangle, \langle C_6 \rangle, \langle C_{10} \rangle\}$	$[0^3]$	0
4	D	$\{\langle D_4 \rangle, \langle D_8 \rangle, \langle D_{11} \rangle\}$	$[0^3]$	0
5	E	$\{\langle E_5 \rangle\}$	$[0]$	0
6	F	$\{\langle F_9 \rangle\}$	$[0]$	0
7	G	$\{\}$	$[]$	$\perp$
8	H	$\{\langle H_{12} \rangle\}$	$[0]$	0
9	τ	$\{\epsilon\}$	$[\perp]$	$\perp$
10	$\times(B, F)$	$\{\langle B_2 \rangle, \langle B_7 \rangle, \langle F_9 \rangle\}$	$[0^3]$	0
11	$\wedge(C, \times(B, F))$	$\{\langle B_2, C_3 \rangle, \langle C_6, B_7 \rangle, \langle F_9, C_{10} \rangle\}$	$[1, 2, 0]$	1
12	$\rightarrow(\wedge(C, \times(B, F)), D)$	$\{\langle B_2, C_3, D_4 \rangle, \langle C_6, B_7, D_8 \rangle, \langle F_9, C_{10}, D_{11} \rangle\}$	$[3, 5, 2]$	3.33
13	$\circlearrowleft(\rightarrow(\wedge(C, \times(B, F)), D), E, \tau)$	$\{\langle B_2, C_3, D_4, E_5 \rangle, \langle C_6, B_7, D_8 \rangle, \langle F_9, C_{10}, D_{11} \rangle\}$	$[4, 5, 2]$	3.67
14	$\times(G, H)$	$\{\langle H_{12} \rangle\}$	$[0]$	0
15	$\rightarrow(A, \circlearrowleft(\rightarrow(\wedge(C, \times(B, F)), D), E, \tau), \times(G, H))$	$\{\langle A_1, B_2, C_3, D_4, E_5, C_6, B_7, D_8, F_9, C_{10}, D_{11}, H_{12} \rangle\}$	$[18]$	18

Table 4.2: Temporal profile of trace in table 4.1

the sub logs, resulting in a tree structure of splits. We refer to [59] for a detailed formal description of the cut operators. Which cut is applicable for a subgraph can be detected using standard graph algorithms for finding connected components. If no cut is applicable, the algorithm returns a flower model $\circlearrowleft(\tau, a_1, \ldots, a_n)$ that allows for any combination of activities $a_1 \ldots a_n$

In [60], the authors extend the algorithm (IMi) to be more robust to noise by filtering edges in a way similar to the Heuristic Miner [93]. The filtering threshold needs to be provided by the user. Once a cut has been detected based on the filtered graph, infrequent behaviour not fitting the cut semantics is filtered from the log. Filtering may also result in empty traces ϵ being left after splitting the log, which is dealt with based on their frequency. If ϵ is frequent enough, a τ node is discovered in the process tree.

The initial variant of the algorithm splits the log into parts corresponding to $\Sigma_1 \ldots \Sigma_n$. Thus, the time complexity depends on the size of the log. In [58], the authors proposed a variant that splits only the directly follows graph instead of sub logs. Thus, by requiring only a single pass over the log to construct the directly follows graph, the time complexity depends only on the number of activities instead of the log size. The authors have successfully applied the algorithm to logs containing hundreds of millions of traces.

As the algorithm depends on the directly-follows graph, incompleteness of the log w.r.t the directly-follows relation can cause suboptimal cuts. To address this problem, an extension (IMin) is proposed in [61], which estimates probabilities for behavioural relations between activities and selects the cut with the highest probability. This optimization problem causes the algorithm to have runtime possibly exponential in the number of activities.

4.3 Conclusion

We argue that using the Inductive Miner together with our feature construction approach has the following benefits:

- Process tree nodes corresponding to features have clear semantics and represent hierarchical structures of subprocesses
- IM has an explicit notion of concurrency
- IMi is able to deal with noise by setting an appropriate threshold for infrequent behaviour
- IMin is potentially able to handle incomplete behaviour (even though its runtime can render it practically infeasible)

The algorithm we proposed for matching the tree nodes to subsequences of events is certainly not ideal, as we have seen for the handling of parallel execution. We can think

of a more sophisticated approach based on alignments of a trace and the workflow net corresponding to the process tree [7]. However, the search space to construct such an alignment may be very large due to the presence of silent transitions introduced by the Inductive Miner. It is also not clear if a more correct matching would result in better features. We compare trace profiles, including profiles constructed with our approach, in different experimental settings in chapter 7.

5 Statistical classification

This chapter describes the classification algorithms we use throughout our experiments, as well as our model selection procedure. We also introduce the performance measures we apply to asses our experimental results. We explain these general machine learning concepts for the sake of completeness, but they are mostly independent of our domain.

5.1 Algorithms

5.1.1 Trees and forests

Decision Trees

Decision tree learning is a simple yet powerful technique that can be used for classification and regression [52, p. 305ff]. Decision trees partition the input space by recursive splits on the input variables, thus modelling non-linear decision functions. An exampe is shown in fig. 5.1.

There are numerous algorithms to learn a decision tree given a labelled training set, that differ mainly in their choice of splitting criterion and pruning strategy. Widely used variants are C4.5 [77], using information gain as the splitting criterion and a pruning strategy based on confidence intervals, and CART [21], which uses cost-complexity pruning.

Random Forests

Decision trees are simple, interpretable models but may suffer from high variance, which is why they are often used in ensembles that combine the predicitions of multiple trees to provide more stable estimates.

Random forests [19] are such an ensemble method. For each tree in the forest, a new training set is created by the *bootstrap* method, i.e., by sampling elements with replacement from the original training set. The predictions of all trees are aggregated, therefore this procedure is known as *bagging*, standing for bootstrap aggregating. In order to reduce the correlation between trees, a random subset of input variables is chosen for each splitting decision during the growing procedure. The trees are grown without pruning. The authors of [19] also propose a method to determine the importance of an input variable by permuting its values in the training set and compute the out-of-bag-error when fed back into the forest.

Random forests are known to work well in a wide range of applications [40]. Practical advantages are that they can handle missing values well and are robust against outliers

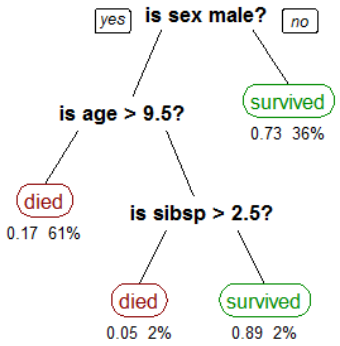


Figure 5.1: Classification tree for predicting survival of Titanic passengers, from [96]. "sibsp" is the number of the passenger's spouses or siblings

and scale-invariant w.r.t the input variables. They are also relatively fast to train: the average case complexity for the learning procedure is $\Theta(mk\tilde{n}\, log^2\, \tilde{n})$, where m is the number of trees, k the number of randomly selected features and $\tilde{n} = 0.632n$ (n being the sample size and 0.632 because bootstrapping draws, on average, 63.2% of unique samples [64, p. 96]). Another advantage of Random Forests is that they output probability estimates $P(Y|X)$, providing a measure for classification confidence and a criterion for ranking.

5.1.2 One-class SVM

In cases where only little or no training data for a particular class is available, the classification task can sometimes be phrased as an *outlier detection* problem [26]: given a dataset, we are looking for a function that distinguishes typical from atypical instances. An algorithm that can be applied in this setting is the one-class support vector machine (OC-SVM) [84], which we briefly describe in this section.

The basic idea is to map the data into a Hilbert space $\mathcal{H}$ via a feature map $\phi : \mathcal{X} \to \mathcal{H}$ and then find a maximum-margin hyperplane that seperates the data from the origin in that space, such that most data lies on one side of the hyperplane. If $w \in \mathcal{H}$ and $\rho \in \mathbb{R}$ are the parameters of this hyperplane, the corresponding decision function can be written as:

$$f(\mathbf{x}) = \text{sgn}(\langle w, \phi(\mathbf{x})\rangle - \rho)$$

This function should return $+1$ for most of the data and -1 for the rest. The attractiveness of this approach – and support vector machines in general – is based on the fact that a linear separation in $\mathcal{H}$ can correspond to a complex non-linear decision boundary in $\mathcal{X}$.

To make computations in $\mathcal{H}$ possible, it is assumed that inner products in $\mathcal{H}$ $\langle \phi(\mathbf{x}), \phi(\mathbf{x}')\rangle$ can be computed by some kernel function $K : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ in the input

space for any two instances $\mathbf{x}, \mathbf{x}' \in \mathcal{X}$, which is known as the kernel "trick".

A frequently used kernel is the Gaussian (or RBF) kernel defined over $\mathbb{R}^d$ [68, p. 94]:

$$K(\mathbf{x}, \mathbf{x}') = e^{-\gamma \|\mathbf{x} - \mathbf{x}'\|^2}$$

where $\gamma > 0$ is a free parameter governing the kernel *width*.

The parameters for the decision function are chosen such that they solve the following (primal) minimization problem:

$$\begin{aligned} \min_{w \in \mathcal{H}, \boldsymbol{\xi} \in \mathbb{R}^n, \rho \in \mathbb{R}} \quad & \frac{1}{2}\|w\|^2 + \frac{1}{\nu n} \sum_{i=1}^{n} \xi_i - \rho \\ \text{s.t.} \quad & \langle w, \phi(\mathbf{x}_i) \rangle \geq \rho - \xi_i, \quad \xi_i \geq 0 \end{aligned}$$

where the ξ_i's are slack variables which are non-zero for those instances which fall on the "wrong" side of the hyperplane. We refer to those instances as outliers. $\|w\|^2$ is a regularization term that encourages simple decision boundaries.

The authors of [84] show that the free parameter ν is an upper bound on the fraction of outliers as well as a lower bound on the fraction of support vectors and thus controls the trade-off between minimizing $\|w\|^2$ and ξ.

Since we usually cannot compute ϕ directly, instead of the primal optimization problem the corresponding dual problem is solved, which depends on the training instances only through the inner product $\langle \phi(\cdot), \phi(\cdot) \rangle$ that can be replaced by the kernel function:

$$\begin{aligned} \min_{\boldsymbol{\alpha} \in \mathbb{R}^n} \quad & \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j K(\mathbf{x}_i, \mathbf{x}_j) \\ \text{s.t.} \quad & 0 \leq \alpha_i \leq \frac{1}{\nu n}, \quad \sum_{i=1}^{n} \alpha_i = 1 \end{aligned}$$

The decision function then becomes:

$$f(\mathbf{x}) = \operatorname{sgn}\left(\sum_{i=1}^{n} \alpha_i K(\mathbf{x}_i, \mathbf{x}) - \rho \right)$$

where $\rho = \sum_{j=1}^{n} \alpha_j K(\mathbf{x}_j, \mathbf{x}_i)$ for any i for which $\alpha_i > 0$, i.e., any support vector $\mathbf{x}_i$.

We refer to [84] for further details on how to derive these formulas and their theoretical justification. The main insight is that in order to apply the OC-SVM to find a non-linear decision boundary, we need to supply a kernel function K as well as the desired fraction of outliers ν.

A downside of kernel SVMs is the time complexity: calculating the kernel matrix takes at least $\mathcal{O}(n^2)$, which not even accounts for the number of features and the time it takes to solve the minimization problem.

5.2 Preprocessing

5.2.1 Feature scaling

Many learning algorithms benefit from preprocessing the feature values if they are on very different scales. For instance, when using SVMs with RBF kernel and having a feature with range $[-0.1, 0.1]$ and another with $[-1000, 1000]$, the second one will dominate the squared norm in the kernel function. Therefore, it is often useful to rescale the features to a common range.

Two simple ways to do achieve this are min-max-scaling:

$$x_i' = \frac{x_i - min(x_i)}{max(x_i) - min(x_i)}$$

and standardization, i.e., transforming the values such that they have zero mean und unit variance:

$$x_i' = \frac{x_i - \bar{x_i}}{\sigma_i}$$

Sometimes, it may also be beneficial to apply a log transformation to the feature values.

In any case, the merit of feature scaling strongly depends on the learning algorithm. For instance, tree-based algorithms do not require such transformations because splits are determined by scanning through the values of single features. Therefore, any monotonic transformation will not change the structure of a tree.

5.2.2 Missing values

When feature values are missing - for instance, when constructing them using the approach we presented in the last chapter - measures need to be taken that again strongly depend on the learning algorithm. If there are very few instances with missing values, possibly due to errors made while acquiring the data, one could simply ignore these instances. However, if missing values are frequent, this is a prohibitive waste of training data and ignores the fact that a missing value might carry some meaning.

Tree-based learners usually deal very well with missing values. An instance whose feature value is missing for a decision node can be split into pieces and considered as a partial instance for each subtree [97, p. 194ff]. Alternatively, missing values can be replaced (imputation), e.g., by a unique value such as an extremely large number that would never occur in real data.

This does not work for SVMs though, as they are sensitive to the scale of features. Here, a simple solution is to replace the missing values with the mean or median of the values which are present. More sophisticated schemes exist that regard possible correlations between features [52, p. 333].

5.3 Performance measures

Several performance measures can be used to assess the performance of a classification algorithm and they are usually based on the confusion matrix, depicted in table 5.1. The elements of this matrix can be related in various ways, some of which we discuss below.

		Predicted class	
		Positive (+1)	Negative (-1)
Actual class	Positive (+1)	True Positives (TP)	False Negatives (FN)
	Negative (-1)	False Positives (FP)	True Negatives (TN)

Table 5.1: Confusion matrix

5.3.1 Accuracy

The accuracy is the fraction of correctly classified instances from all instances, i.e., it does not distinguish between positive and negative instances.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

This measure can be misleading in the case of a skewed class distribution. If there are only very few positive instances, say 1%, a trivial classifier which always predicts "negative" would achieve 99% accuracy. We therefore resort to other performance measures and describe them below.

5.3.2 Area under the curve

If the learning algorithm outputs scores, such as probabilities, the predictions can be sorted into a ranked list. In cases where this ranking is of primary interest, its quality is typically measured in terms of the *receiver operating characteristic* (ROC) curve. The ROC curve is obtained by plotting the FP rate against the TP rate for varying discrimination thresholds, i.e., the score upon which a prediction is considered positive (see fig. 5.2). The FP rate is defined as $FPR = FP/(FP + TN)$. Similarly, the TP rate is defined as $TPR = TP/(TP + FN)$.

By stepwise incrementation of the discrimination threshold, more and more instances are labeled as positive, and we would expect both TP rate und FP rate to grow. The *area* under this curve (AUC) coincides with the average bipartite ranking performance of a classifier h on a test sample [68, p. 224f]. If we denote this sample as U, which

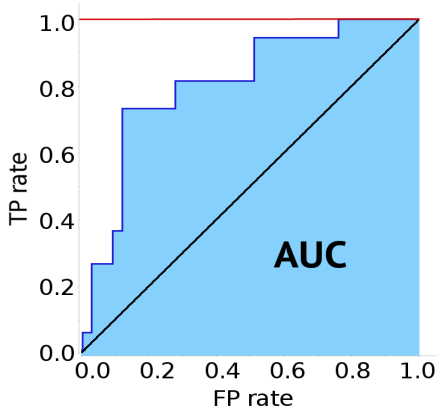


Figure 5.2: Area under the curve. The diagonal black line corresponds to a random guess with an AUC of 0.5. The horizontal red line corresponds to a perfect classifier with AUC 1.0.

has m positive points $z'_1, \ldots, z'_m$ and n negative points $z_1, \ldots, z_n$, then:

$$AUC(h, U) = \frac{1}{mn} \sum_{i=1}^{m} \sum_{j=1}^{n} \mathbb{1}_{h(z'_i) \geq h(z_j)}$$

where $\mathbb{1}$ is the indicator function. In contrast to accuracy, a trivial classifier with a constant distribution of labels, e.g., one that always predicts the majority class, has an expected AUC value of 0.5 no matter how skewed the classes are.

5.3.3 Precision and recall

Precision and recall are two measures regularly applied in information retrieval applications but can also be used to evaluate classification results. Recall is just another name for the TP rate, whereas precision is the fraction of true positives from all instances flagged as positive, that is $P = TP/(TP + FP)$.

Since it is sometimes convenient to have a single real number to compare the performance of classifiers, precision and recall can be combined by calculating their harmonic mean [97, p. 175]. This so called F-measure is defined as:

$$F = \frac{2 \cdot P \cdot R}{P + R}$$

The F-measure is more sensitive to skew than AUC. To see this, consider two samples with a class distribution of positive to negative instances of 1%/99% and 50%/50%, respectively. A classifier that always predicts "positive" would achieve an F-score of $2 \cdot 0.01 \cdot 1/(0.01 + 1) = 0.02$ on the first sample, whereas on the second sample it would be $2 \cdot 0.5 \cdot 1/(0.5 + 1) = 0.67$. In contrast to AUC, F-measure can also be applied with classifiers that do not output scores or probabilities, such as SVMs[1].

[1] This is not entirely true, since SVM classification outputs *can* be turned into probabilities by a method called Platt scaling [74]. However, this requires an additional logistic transformation of the SVM scores, whose parameters have to be learned in addition to the SVM parameters.

5.4 Model selection

When we want to estimate the performance of a hypothesis, we need a way to do this without looking at the hold-out test data. For this purpose, we could either keep aside a part of the training data as a validation set or perform K-fold cross validation [52, p. 241ff.]. When training data is scarce, cross validation makes better use of it. Such an estimate can be necessary especially if the learning algorithm has hyperparameters that need to be chosen. Examples are the noise parameter of the Inductive Miner algorithm that we use for feature construction, the SVM parameter ν and the kernel width γ for the RBF kernel. The simplest approach for choosing these values is to iterate through many different combinations of hyperparameters (grid search) and choose the combination with best predictive performance in K-fold cross validation. We then use this set of hyperparameters to train the model on the complete training set and apply it to the test set.

Let $\boldsymbol{\alpha}$ be the set of hyperparameter combinations and $\kappa : \{1 \dots n\} \to \{1 \dots K\}$ a function that assigns each training sample to one of K folds. The grid search procedure works as described in listing 5.1.

Listing 5.1 Grid search for hyperparameter optimization

function GridSearch($\boldsymbol{\alpha}, X_{tr}, Y_{tr}$)
  for $\alpha \leftarrow \boldsymbol{\alpha}$ **do**
    for $k \leftarrow 1 \dots K$ **do**
      $\hat{h}_\alpha^{-k} \leftarrow$ train model with parameters α on (X_{tr}, Y_{tr})
        with all (x_i, y_i) for which $\kappa(i) = k$ removed
      for $x_i \leftarrow X_{tr}, \kappa(i) = k$ **do**
        $y_i' \leftarrow \hat{h}_\alpha^{-k}(x_i)$
      end for
    end for
    $P(\alpha) \leftarrow$ compute performance by comparing all y_i' and real labels y_i from Y_{tr}
  end for
  return $\arg\max_\alpha P(\alpha)$
end function

Exhaustively scanning through all combinations can be computationally expensive, especially when the number of hyperparameters is large. However, as the combinations are independent of each other, the search procedure can be easily parallelized across multiple CPUs or machines.

The lack of hyperparameters that need to be chosen[2] is another practical advantage of Random Forests.

[2] ... apart from the number of trees, where often more is just better

6 Datasets

In this chapter, we introduce the event logs that have been used for the experimental evaluation. We start with a short introduction of the domain, the software system under consideration and the event logs we extracted from it. We provide some descriptive statistics of the characteristics of the event logs that allow to better judge the experimental results.

6.1 Domain

The European Union spends a large fraction of its budget on the Common Agricultural Policy (CAP). In 2014, the expenditures for the CAP amounted to around €58 billion or 40% of the EU budget [37]. Out of this, €41.7 billion [36] were *direct payments* that are mainly aimed to provide a basic income for farmers decoupled from production. The rest of the budget is spent for market related expenditures and rural development.

The processes that govern the distribution of these funds are subject to complex regulations captured in EU and national law. The member states are required to operate an Integrated Administration and Control System (IACS), which includes IT systems to support the complex processes of subsidy distribution.

We had access to the Java Enterprise system **profil c/s**[1], which is used in 9 out of 16 German federal states at the level of the federal ministries of agriculture and local departments. The system supports various kinds of administrative processes, but we focus on the yearly allocation of direct payments, starting with the application and, if all goes well, finishing with the authorization of a payment.

Most activities that are performed in this software are recorded explicitly, i.e., event logs are "first-class citizens" and of relatively high quality. We extracted the logs for the period of 2009-2013 in two federal states, namely *Saxony-Anhalt* (referred to by *ST*) and *Schleswig-Holstein* (referred to by *SH*), each log containing many thousands of individual cases.

[1] **profil** is an abbreviation for the German name **Pro**gramm zur **F**ördermittelverwaltung **i**n der **L**andwirtschaft, which we would roughly translate with "Program for managing subsidies in agriculture". **c/s** stands for Client/Server.

Case attributes		Document + Activity	User	Timestamp		succ.
concept:name	20009001269	FP200 + DokumentEingegangen	Import	2009/03/25	00:00:00	true
applicant	19610230044	FP200 + DokumentGueltig	9	2009/04/21	00:00:00	true
year	false	Flächen + Initialisieren	Dummy	2009/05/01	05:47:55	true
dep	3	Flächen + BearbeitungBeginnen	Dummy	2009/05/01	05:47:56	true
electronic	true	Flächen + ErsterfassungBeenden	Dummy	2009/05/01	05:47:56	true
fp	200	Flächen + BearbeitungBeginnen	Dummy	2009/05/01	05:58:08	true
rejected	false	Flächen + ErsterfassungBeenden	Dummy	2009/05/01	05:58:09	true
numparcels	4	Flächen + BearbeitungBeenden	Dummy	2009/06/17	11:28:38	true
area	18.74	VWK RFA + Initialisieren	Dummy	2009/06/19	20:26:02	true
cc	0	VWK RFA + DurchfuehrungVermerken	Dummy	2009/06/19	20:26:02	true
cutting0	0	ZFK + Initialisieren	Dummy	2009/06/19	20:26:03	true
payment_granted0	3942.33	ZFK + BearbeitungBeginnen	Dummy	2009/06/19	20:26:03	true
payment_actual0	3942.33	ZFK + BearbeitungBeenden	Dummy	2009/06/19	20:26:03	true
cutting1	-	VWK Fl + DurchfuehrungVermerken	Dummy	2009/09/17	10:09:49	true
payment_granted1	-	VWK Fl + BearbeitungBeenden	Dummy	2009/09/17	10:09:49	true
payment_actual1	-	VWK Fl + BearbeitungBeenden	Dummy	2009/10/20	07:24:06	true
cutting2	-	FP200 + Initialisieren	Dummy	2009/10/26	05:37:29	true
payment_granted2	-	FP200 + BearbeitungBeginnen	Dummy	2009/10/27	08:52:34	true
payment_actual2	-	FP200 + BerechnenUndPruefen	Dummy	2009/10/27	08:52:36	true
		FP200 + BearbeitungBeenden	Dummy	2009/10/27	08:52:36	true
		FP200 + BearbeitungBeginnen	Dummy	2009/11/02	07:47:53	true
		FP200 + BerechnenUndPruefen	Dummy	2009/11/02	07:47:54	true
		FP200 + BearbeitungBeenden	Dummy	2009/11/02	07:47:54	true
		FP200 + BearbeitungBeginnen	Dummy	2009/11/13	07:43:34	true
		FP200 + BerechnenUndPruefen	Dummy	2009/11/13	07:43:35	true
		FP200 + BearbeitungBeenden	Dummy	2009/11/13	07:43:36	true
		FP200 + BearbeitungBeginnen	Dummy	2009/11/13	09:00:50	true
		FP200 + BerechnenUndPruefen	Dummy	2009/11/13	09:00:51	true
		FP200 + BearbeitungBeenden	Dummy	2009/11/13	09:00:51	true
		FP200 + AntragBewilligen	Dummy	2009/11/13	10:23:01	true
		FP200 + AntragZahlungAnweisen	Dummy	2009/12/01	09:07:36	true
		FP200 + Nutzungsaktualisierung	Dummy	2010/01/18	19:24:28	true

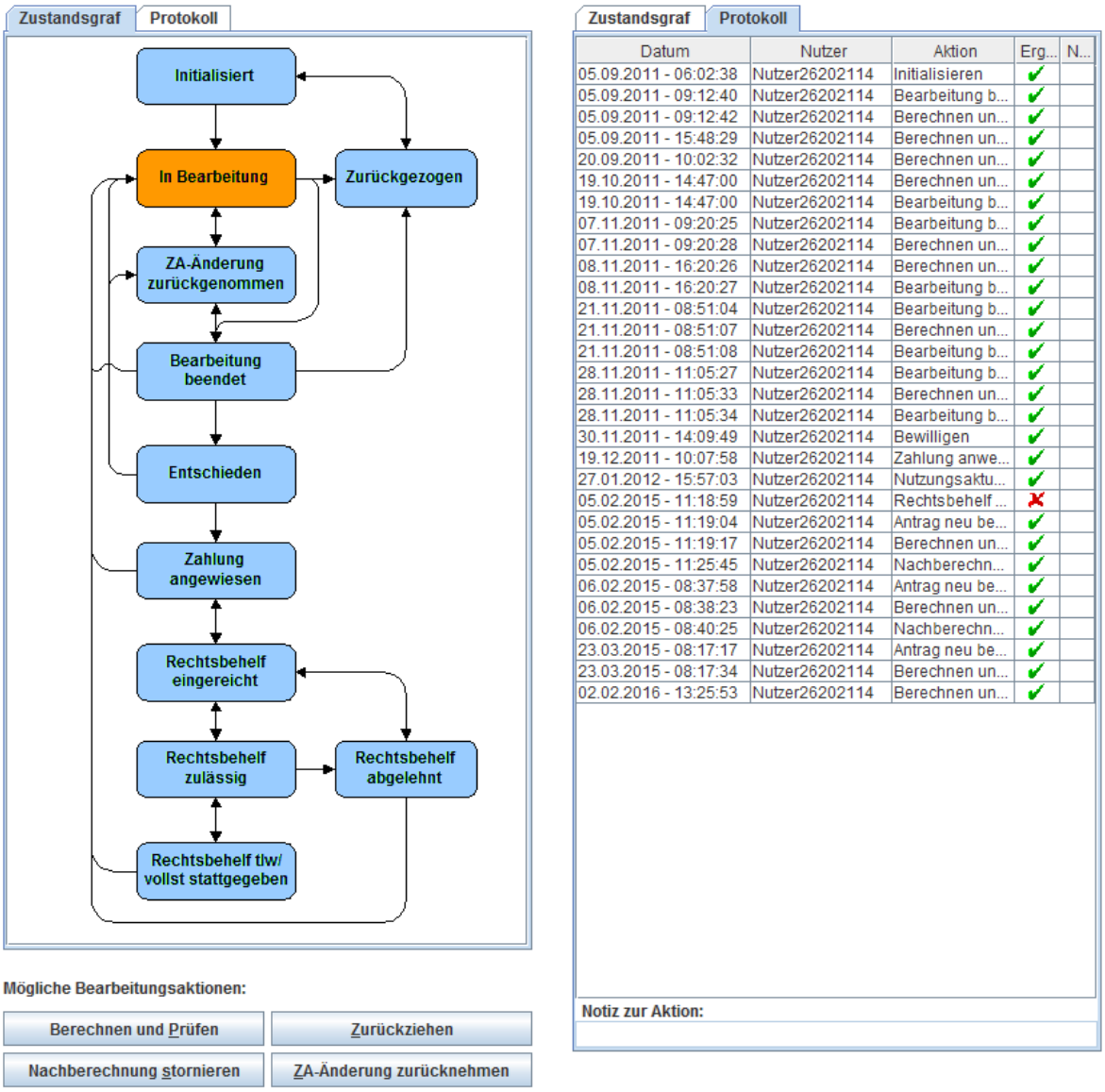
Table 6.1: Example trace from *SH* ending regularly

6.2 Data extraction and preprocessing

The workflows in **profil c/s** can be understood in terms of documents, where each document has a state that allows for certain actions. These actions can be executed manually through document specific tools or, in many cases, automatically scheduled. There is no overall mechanism, such as a workflow engine, governing the execution of these activities. However, certain actions can only be taken if requirements are fulfilled, e.g., related documents are in a final state. Most actions are reversible, i.e., documents can usually be transferred into a non-final state. We show an example of the state graph of an "FP200" document (the main document) in fig. 6.1 (a). The current state is marked, the possible actions are indicated by arrows. Note that while this document is in this particular state, other actions in other documents may be necessary before the next action can be performed. Events are recorded on the level of documents, cf. fig. 6.1 (b).

A case consists of all events relating to all documents for one applicant in one year. Table 6.1 and 6.2 show two examples of cases from *SH*. The second example shows the possible sequence of actions in different documents.

All events are recorded in one large database table with foreign keys relating to the



(a) State graph with possible next actions below (b) Protocol

Figure 6.1: State graph and event protocol for the main document “FP200”

documents, which are themselves related to years and individual applicants.

Since it would have been tedious to resolve all the relations in the database, we instead wrote a script that leveraged the existing object-relational mapping approach, read the whole object graph corresponding to one applicant and year and converted it to XES with the OpenXES Library. The start of the first example case serialized to XES is depicted in appendix A.

As the only preprocessing step, all occurences of the activity “ZFK aktualisieren" were removed. This activity was executed daily at the same time and amounted for about 50% of the total log size.

Note that the events do *not* contain information about changing data items. This implies, that once a case has finished, we only have access to its final document state. We also do not know which user performed an activity, since most user names have been anonimyzed and replaced by the same dummy user.

6.3 Statistics & insights

We performed an explorative analysis of the data using Disco and ProM to get a first intuition of its characterstics. Table 6.3 shows a general view of the data for one year (2011). The number of cases is about the same for every other year.

We see that in *SH*, there are about 4 times as many cases as in *ST* but divided among fewer departments. We also see that the variability in terms of exact activitiy sequences is much higher in *ST* than in *SH*. We ascribe this to the fact that the departments in *SH* make heavier use of automation, resulting in a significant number of cases which are never manually touched and therefore exhibit a very regular workflow.

Fig. 6.2 shows the distribution of cases per variant. Since the distribution is extremely concentrated at the most populous variants, we also plotted the data with a log transformation of both axes. The straight line in these plots indicates that the variants follow some kind of power law distribution.

We observed that the length of cases is subject to large variablity, the longest cases taking > 5 years, which means that they are still active at the time of this writing. The minimum and median case durations are smaller than one year.

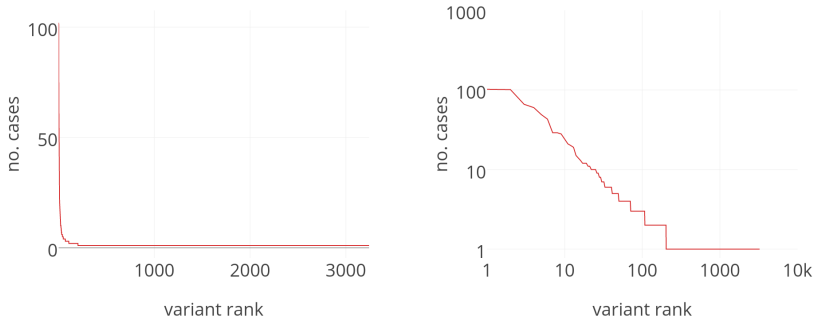
To visualize the time distribution of cases, we created dotted charts [86] for all the logs from *ST* in fig. 6.3. The overall picture for *SH* is very similar. The first plot shows clearly how the data arrives in yearly batches. The second plot shows a more detailed perspective. Nearly all cases start at around May of each year[2] and finish by the end of the year. However, some cases take much longer, because the departments need to make corrections after the initial payment has be authorized.

This is the kind of behaviour we will focus our predictions on, i.e., we want to know if a case is going to cause additional work at an early point – in the best case at a point were there is still a possibility for intervention. We will elaborate on these conditions later when we introduce the exact experimental design.

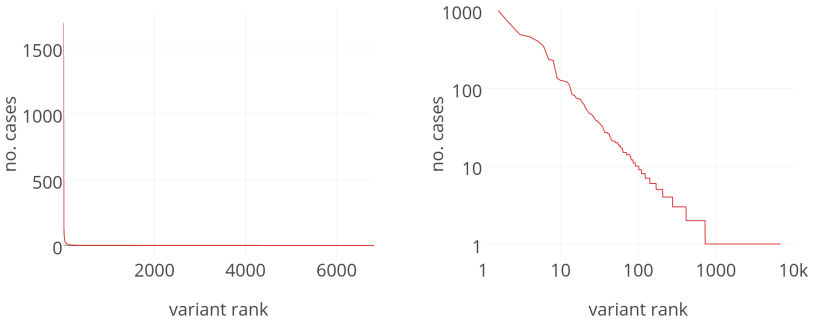
[2]This is due to the fact that the deadline for handing in applications is always the 15th of May.

Case attributes		Document + Activity	User	Timestamp		succ.
concept:name	20009004911	Flächen + Initialisieren	Dummy	2009/05/07	05:15:51	true
applicant	19540980018	Flächen + BearbeitungBeginnen	Dummy	2009/05/07	05:15:52	true
year	2009	Flächen + ErsterfassungBeenden	Dummy	2009/05/07	05:15:53	true
dep	6	Flächen + BearbeitungBeginnen	Dummy	2009/05/07	05:19:06	true
electronic	true	Flächen + ErsterfassungBeenden	Dummy	2009/05/07	05:19:08	true
fp	200	FP200 + DokumentEingegangen	Import	2009/05/10	00:00:00	true
numparcels	42	FP200 + DokumentGueltig	37	2009/05/13	00:00:00	true
rejected	false	VWK RFA + Initialisieren	Dummy	2009/06/19	21:20:04	true
area	105.2601	VWK RFA + DurchfuehrungVermerken	Dummy	2009/06/19	21:20:04	true
cc	15	ZFK + Initialisieren	Dummy	2009/06/19	21:20:09	true
cutting0	0	ZFK + BearbeitungBeginnen	Dummy	2009/06/19	21:20:09	true
payment_granted0	32811.27	ZFK + BearbeitungBeenden	Dummy	2009/06/19	21:20:09	true
payment_actual0	32811.27	VWK RFA + BearbeitungBeginnen	Dummy	2009/07/01	18:38:21	true
cutting1	0	VWK RFA + BearbeitungBeginnen	Dummy	2009/07/02	18:59:15	true
payment_granted1	38601.5	VWK RFA + BearbeitungBeginnen	Dummy	2009/07/10	17:40:16	true
payment_actual1	5790.23	VWK RFA + BearbeitungBeginnen	Dummy	2009/07/16	20:02:37	true
cutting2	-	VWK RFA + BearbeitungBeginnen	Dummy	2009/07/17	11:40:33	true
payment_granted2	-	VWK RFA + BearbeitungBeginnen	Dummy	2009/08/13	08:17:28	true
payment_actual2	-	VWK RFA + BearbeitungBeginnen	Dummy	2009/10/09	15:07:43	true
		Flächen + BearbeitungBeenden	Dummy	2009/10/14	13:40:49	true
		Flächen + ErsterfassungBeenden	Dummy	2009/10/14	13:53:28	true
		Flächen + Speichern	Dummy	2009/10/14	13:54:30	true
		Flächen + BearbeitungBeenden	Dummy	2009/10/14	13:54:34	false
		Flächen + Pruefen	Dummy	2009/10/14	13:54:46	true
		Flächen + Speichern	Dummy	2009/10/14	13:55:48	true
		Flächen + Pruefen	Dummy	2009/10/14	13:55:51	true
		Flächen + BearbeitungBeenden	Dummy	2009/10/14	13:55:54	true
		FP200 + Initialisieren	Dummy	2009/10/26	06:01:37	true
		FP200 + BearbeitungBeginnen	Dummy	2009/11/11	17:17:38	true
		ZFK + Speichern	Dummy	2009/11/12	14:29:30	true
		VWK Fl + DurchfuehrungVermerken	Dummy	2009/11/13	12:54:54	true
		VWK Fl + BearbeitungBeenden	Dummy	2009/11/15	12:54:54	true
		FP200 + BerechnenUndPruefen	Dummy	2009/11/15	17:17:39	true
		FP200 + BearbeitungBeenden	Dummy	2009/11/15	17:17:40	true
		FP200 + AntragBewilligen	Dummy	2009/11/18	15:54:24	true
		FP200 + AntragZahlungAnweisen	Dummy	2009/12/01	08:56:07	true
		FP200 + Nutzungsaktualisierung	Dummy	2010/01/18	19:09:30	true
		FP200 + AntragWidersprechen	Dummy	2010/10/29	14:21:09	true
		FP200 + Speichern	Dummy	2010/10/29	14:21:19	true
		FP200 +WiderspruchZulassen	Dummy	2010/10/29	14:21:22	true
		FP200 +WiderspruchStattgeben	Dummy	2010/10/29	14:21:26	true
		FP200 + AntragNeuBearbeiten	Dummy	2010/10/29	14:21:30	true
		FP200 + BerechnenUndPruefen	Dummy	2010/10/29	14:21:34	true
		FP200 + ZAAenderungZurueck	Dummy	2010/10/29	14:23:29	true
		FP200 + NachbBearbeitungBeenden	Dummy	2010/10/29	14:23:54	true
		FP200 + AntragBewilligen	Dummy	2010/10/29	14:24:43	true
		FP200 + Speichern	Dummy	2010/10/29	14:32:38	true
		FP200 + AntragZahlungAnweisen	Dummy	2010/11/12	06:48:33	true
		FP200 + Nutzungsaktualisierung	Dummy	2010/11/18	10:33:46	false
		FP200 + Nutzungsaktualisierung	Dummy	2010/11/18	10:34:28	true

Table 6.2: Example trace from *SH* involving a legal complaint resulting in a second payment (indicated by the marked activities)



(a) Cases per variant *ST* (b) Cases per variant *ST* (log-log)

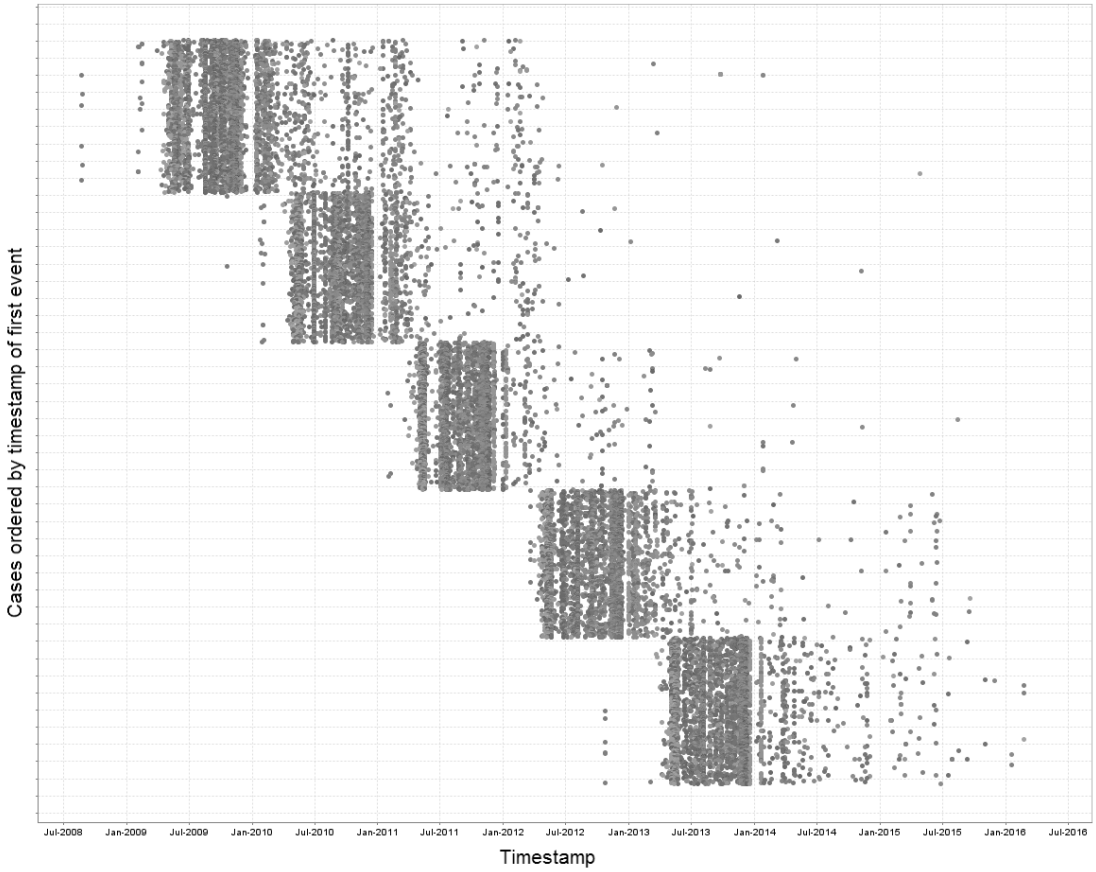


(c) Cases per variant *SH* (d) Cases per variant *SH* (log-log)

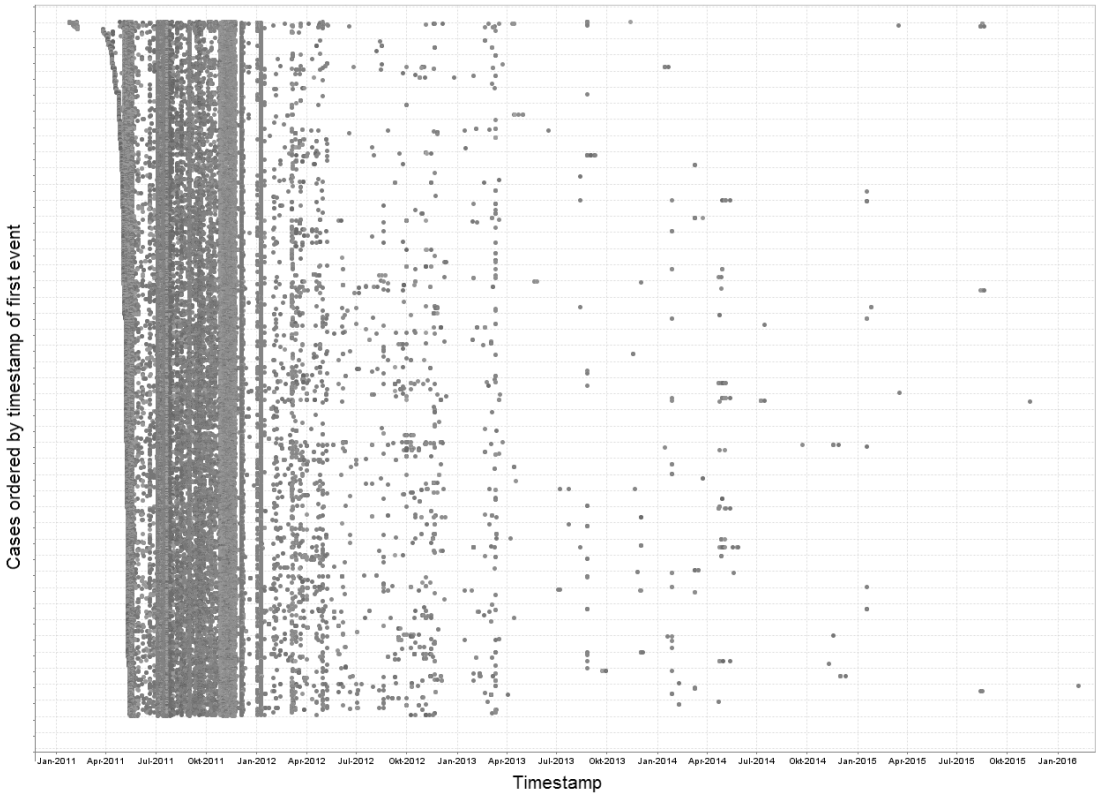
Figure 6.2: Distribution of variants in *ST* and *SH*. The x axis orders the most populous variants, the y axis indicates the corresponding number of cases

	***ST*-2011**	***SH*-2011**
# cases	4303	15484
# variants	3242	6819
# cases in top 10 variants[1]	531 (12 %)	4861 (31 %)
# activities	85	92
# events	196793	688475
Min # events	17	17
Max # events	983	683
Min case duration	32 wks	25 wks
Max case duration	261 wks	247 wks
Median case duration	35.9 wks	39.7 wks

Table 6.3: Statistics for all cases of one year (2011) in both datasets. The number of cases and variants are similar for all other years.
[1] We determined the 10 most frequent variants and summed their corresponding number of instances



(a) All cases in ST from 2009-2013



(b) Cases in ST from 2011

Figure 6.3: Dotted chart of cases in ST in ascending order of start time

6.3.1 Case attributes

The *case attributes* present in both logs are shown in table 6.4.

Attribute	Unit	Description
concept:name	ID	unique case id
applicant	ID	applicant number, unique per year
fp	ID	program number
year	a	application year
dep	ID	department number
electronic	0/1	electronic submisson
area	ha	declared area
numparcels	$\mathbb{N}_0$	number of parcels
cc	%	cross compliance penalty
rejected	0/1	rejection planned
cutting0	€	cutting (difference applied - established payment)
payment_granted0	€	granted payment
payment_actual0	€	actual payment (reduced granted payment)

Table 6.4: Case attributes, restricted to those being used in the experiments

Monetary values derived in the process, such as *payment_granted*0, are snapshots referring to a particular payment decision. The attributes suffixed with "0" refer to the initial payment. As we can see in the example in table 6.2, there are other case attributes for following payments (suffixes with "1", "2", ... respectively), but we will ignore them since they are not available at the time we try to make predictions.

6.3.2 Structure

We wanted to get a feeling how structured the processes are and how they differ across departments within one state.

In order to obtain a rough indicator for structuredness, we used the Heuristic Miner [93] with default settings on the level of individual departments and for all traces of a single year in each of the two federal states. We report the *fitness* value returned by the ProM plugin for the resulting C-net[3]. An example for the output of the Heuristic Miner plugin can be found in appendix B.

As for the fitness, we also considered prefixes of the cases up to the end of the corresponding year to better distinguish "normal" behaviour from exceptional behaviour of very long running cases.

The results are presented in tables 6.5 for ST and 6.6 for SH. We see that in ST the processes within single departments are fairly structured and the fitness values do not change much when restricted to prefixes. We also see that when looking at

[3]After digging into the source code of the Heuristic Miner plugin, we found that the fitness value reported by the plugin is calculated by the improved continuous semantics (ICS) measure defined by [31]. This measure may take negative values.

all departments, the Heuristic Miner cannot construct a fitting C-net. A possible conclusion is that in *ST* the departments share little commonalities in their process implementation.

The results for *SH* look different. The fitness within each department is lower than in *ST*, but does not drop as much when measured across departments. The values also increase noticably when constrained to prefixes, indicating that much exceptional behaviour is observed in unusually long cases. It is important to note that in *SH* some departments handle more applications than the whole state of *ST* combined, so directly comparing fitness values across states may be misleading.

To see how different years compare to each other, we calculated the fitness values for every year and all years combined in one department per state (see table 6.7). These fitness values vary over the years and the values for all years combined are much lower than the individual fitness values, which we attribute to concept drift taking place across years[4].

Dep.	**# cases**	**# variants**	**# events**	**Fitness**	**Fitness (prefix)**
101	694	557	32519	0.58	0.58
202	579	472	27268	0.52	0.48
268	384	285	17240	0.53	0.55
303	716	524	31173	0.56	0.74
357	448	363	22158	0.37	0.35
363	955	681	42181	0.43	0.43
370	528	394	24254	0.58	0.65
All	**4304**	**3242**	**196793**	**-0.07**	**-0.05**

Table 6.5: Variants and fitness values for year 2011 for different departments in *ST*

Dep.	**# cases**	**# variants**	**# events**	**Fitness**	**Fitness (prefix)**
1	3972	1776	177650	0.23	0.39
2	2022	1075	92791	0.33	0.49
3	4753	1990	204859	0.35	0.56
6	4737	2111	213175	0.30	0.52
All	**15484**	**6819**	**688475**	**0.15**	**0.26**

Table 6.6: Variants and fitness values for year 2011 for different departments in *SH*

	2009	**2010**	**2011**	**2012**	**2013**	**all**
***ST*-101**	0.36	0.40	0.58	0.63	0.27	0.06
***SH*-1**	0.38	0.26	0.23	0.46	0.62	0.05

Table 6.7: Fitness values for a single department in both states across years and for all years combined

[4]In fact, there are new EU regulations which need to be implemented every year, so we would definitely expect somewhat different workflows across different years.

7 Experiments

We have conducted several experiments to compare different trace profiles using the event logs presented in the previous chapter. In the experiments, we tried to predict different aspects of the future of running cases which may benefit from an early intervention.

We implemented the experimental setup in the Scala programming language [34] and used the following software libraries:

- OpenXES [48] for reading and manipulating event logs
- the ProM Inductive Miner plugin [60] for constructing process trees
- the WEKA [65] machine learning toolkit, in particular the implementation of Random Forest and the LibSVM [27] wrapper

7.1 Training and test sets

We used all cases that took place in profil c/s in 2013 as test data, as in reality, this data would arrive in one batch. Since we expect some sort of concept drift to happen across years, we also used 20% randomly picked cases from the rest of the data (2009-2012) as another test set, as depicted in fig. 7.1. We denote the latter as L_{te}^1 and the former as L_{te}^2.

This way, we can simulate a scenario where the test data comes from the same distribution as the training data, as is typically assumed in machine learning. Additionally, the differences in predictive performance will allow us to make statements about the impact of concept drift.

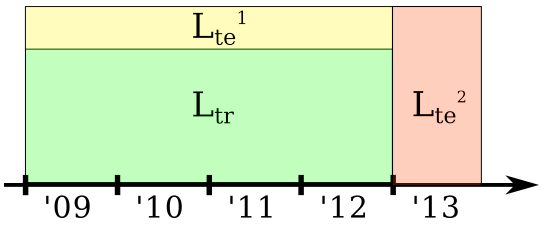
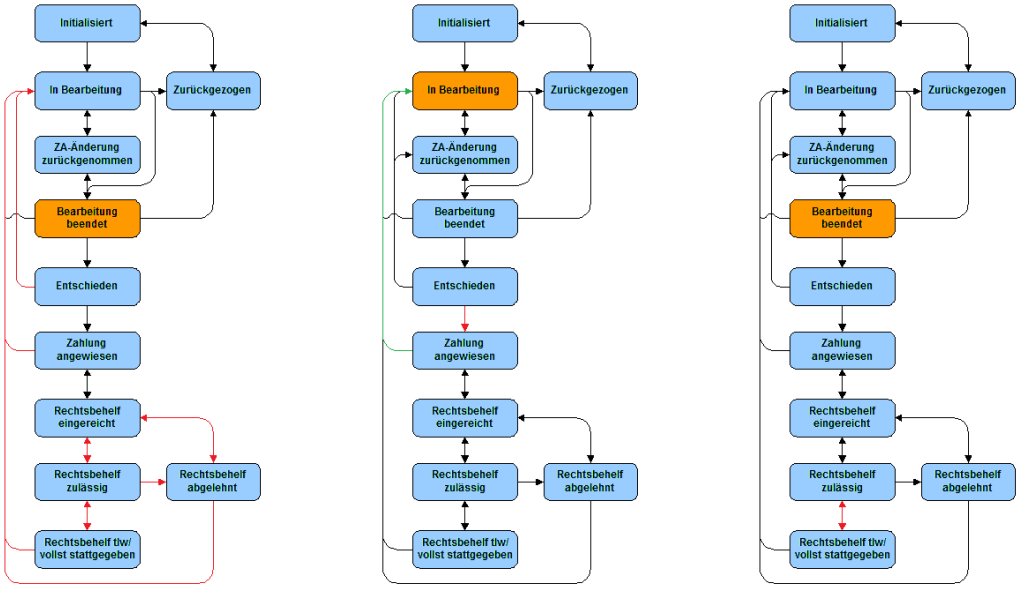


Figure 7.1: Partioning of the data sets in training and test data



Scenario 1:
Will the case need any kind of manual intervention after it has been decided?

Scenario 2:
If a case has been reopened (green arrow), was this necessary or will the activity be reverted?

Scenario 3:
Will the applicant (legitimately) object the payment decision?

Figure 7.2: Prediction scenarios. Orange states mark the point of prediction, red arrows mark the type of future activities we want to predict

7.2 Binary classification

7.2.1 Experimental design

Recall from the last chapter that some cases caused additional work after the initial payment and we would like to predict which these are at an early point. There are many kinds of "additional work", so we derived three scenarios with different target variables that can be tracked within the document "FP200".

For each scenario, we consider the occurence of one or more activities in the future as the criterion for belonging to the "positive" class. In fig. 7.2, the relevant activities and points of prediction are shown. In the second scenario, we assume that the case is already in its second iteration, i.e., it has been reopened.

We constructed several trace profiles by combining different implementations for the σ, π and Γ parts of ψ_{trace}, most of which have been described in section 4.1.1. The resulting profiles are depicted in table 7.1.

Values for the selection function σ

$$\text{single}_i(t) = \{\langle t(j)\rangle \mid 1 \leq j \leq |t| \wedge \#_{act.}(t(j)) = a_i\}$$

$$\text{index}(k)_{j+(i-1)|\mathcal{A}|}(t) = \{t(i) \mid \#_{act.}(t(i)) = a_j\} \quad (1 \leq j \leq k)$$

$$\text{tree}_i(t) \ldots \textit{as described in section 4.2.2}$$

Values for the projection function π

$$\text{count}(e) = 1$$

$$\text{date}(e) = \vartheta(\#_{time}(tl(e)))$$

$$\text{dur}(e) = \delta(\#_{time}(hd(e)), \#_{time}(tl(e)))$$

where we set $\vartheta(\cdot)$ to the *week of the year* to capture large scale organizational patterns in the administration and $\delta(\cdot,\cdot)$ to the difference in *minutes* to capture operational patterns on the level of subprocesses.

Values for the aggregation function Γ

$$bag(s) = s(1)$$

$$min'(s) = \begin{cases} min(s) & \text{if } |s| > 0 \\ \perp & \text{otherwise} \end{cases}$$

$$max'(s) = \begin{cases} max(s) & \text{if } |s| > 0 \\ \perp & \text{otherwise} \end{cases}$$

$$diff'(s) = \begin{cases} max(s) - min(s) & \text{if } |s| > 0 \\ \perp & \text{otherwise} \end{cases}$$

$$sum'(s) = \begin{cases} \sum_i s(i) & \text{if } |s| > 0 \\ \perp & \text{otherwise} \end{cases}$$

Leaving the index-based profiles aside, we chose the features such that each profile contains a superset of the features of the simpler profiles. Thus, we hope to be able to assess the experimental results in terms of less and more complex models.

The index-based profiles (losely based on [62]) serve as a reference for a different encoding style based on exact positions of activities. We are interested in how well such an approach performs in our setting with a lot of variability in the order of activities and considered prefixes of length 15 and 30.

Profile name	σ	π	Γ	**Imput.**[1]	**# feat. *ST*-101**[2]	**# feat. *SH*-1**[2]
case only	-	-	-	-	12	12
index-based 15	index(15)	count	bag	-	27/957 [4]	27/1032 [4]
index-based 30	index(30)	count	bag	-	42/1902 [4]	42/2052 [4]
bag-of-activities	single	count	bag	-	75	79
bag-of-activities+time	single	count	bag	-	264	280
		date	$\min'$	$2^{32}-1$		
			$\max'$	$2^{32}-1$		
			diff'	$2^{32}-1$		
tree-based	tree	count	bag	-	383	340
		date	$\min'$	$2^{32}-1$		
			$\max'$	$2^{32}-1$		
			diff'	$2^{32}-1$		
		dur[3]	$\min'$	$2^{32}-1$		
			$\max'$	$2^{32}-1$		
			sum'	$2^{32}-1$		

Table 7.1: Trace profiles used in binary classification experiments
[1] Imputation value if Γ can return $\perp$
[2] Number of features when applied to L_{tr}
[3] Only for $\rightarrow$, $\circlearrowleft$ and $\wedge$ nodes
[4] Categorical features / effective no. of features in $\mathbb{R}^d$

7.2.2 Implementation

For each scenario and each trace profile, a Random Forest of 500 trees was trained on all *prefixes* of cases in the training logs, cut off before the respective prediction point and labelled according to the scenario.

When predicting on L_{te}^1, we performed 5-fold cross validation with stratified folds of L_{tr} for model selection. In this case, this meant choosing the noise threshold of the Inductive Miner for the tree based-profile. We then applied the best model for each profile to L_{te}^1 and report the resulting AUC values.

When predicting on L_{te}^2, we used all cases from L_{tr} and L_{te}^1 for training, i.e., the complete data from 2009-2012. For cross validation, we created a fold for every year, training on all cases from 3 years and predicting on cases from the remaining year, then micro-averaging the results. This way, we tried to factor in the effect of predicting across years.

7.2.3 Results

We present a summary of the results for binary classification. For each scenario, we plotted the AUC values for L_{te}^1 and L_{te}^2 in all departments per federal state. Averaged results over all departments and the results for training on the whole dataset instead of a per-department basis are presented in tabular form.

For both training / test splits, we report the number of positive and negative instances in the training set (columns Tr+ and Tr-), the AUC achieved in cross validation (column CV), the number of positive and negative instances in the respective test set (columns Te+ and Te-) and finally the AUC on the test set (column AUC). We report the results per department, the weighted average for all departments and the results training on all departments at once.

The highest AUC values achieved are each printed in bold face. Since the most interesting number is the AUC on L_{te}^2, we underlined the highest value by this measure as well as the corresponding trace profile. This way, it can easily be seen which trace profile would have performed best in "practice".

Detailed results for all departments can be found in appendix C.

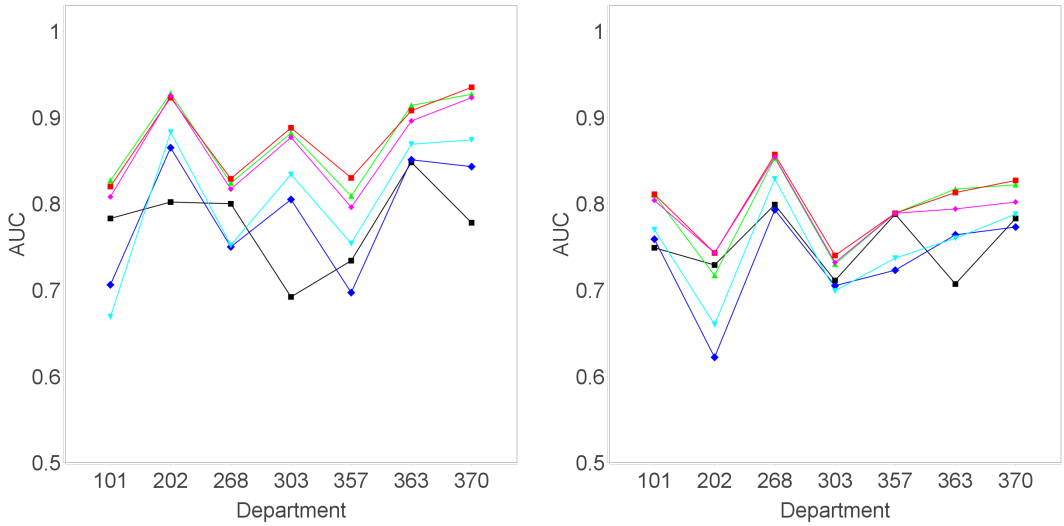
Scenario 1

The results for scenario 1 are shown in fig. 7.3 and table 7.2.

In the first scenario, the "bag-of-activities+time" profile performs best in almost all departments in ST, with the "bag-of-activities" and "tree-based" profiles slightly behind. In SH, the "bag-of-activities" profile achieves consistently the best results.

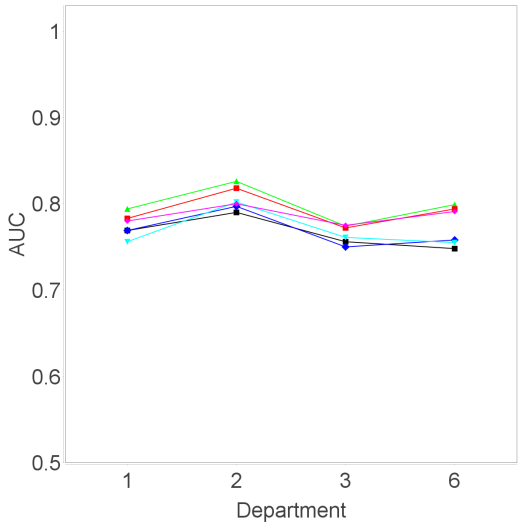
Interestingly, there is a notable drop in performance for all departments in ST when applied to L_{te}^2, i.e., the data that is not sampled from the same distribution as the training data. The results in SH are much more stable and actually improve when applied to L_{te}^2. We attribute this improvement to the model stability introduced by the larger number of training data that is available in SH and that the largest amount of training data is available when predicting on L_{te}^2.

Note that the improvement over using only case attributes is not really substantial, so most predictive power seems to come from the case attributes in this scenario. While it is worthwile to include some high-level workflow information, the ability to find behavioural patterns related to the outcome is obviously limited. In fact, using too may features can actually degrade performance, which can be a symptom of overfitting on the training data. Our guess is that part of the reasons that lead to manual intervention can be attributed to factors that are not present in our event log. A domain expert might be able to suggest what reasons these could be. Then, the event logs could be extended accordingly, either with additional case attributes or events.

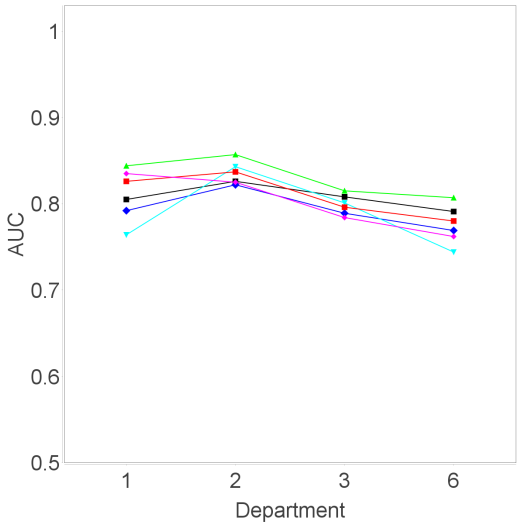


(a) L_{te}^1 performance in ST

(b) L_{te}^2 performance in ST



(c) L_{te}^1 performance in SH



(d) L_{te}^2 performance in SH

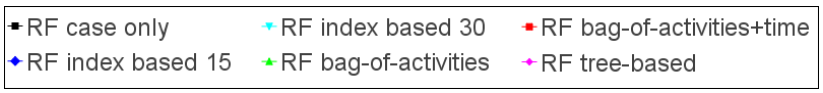


Figure 7.3: AUC values per department in scenario 1

	L^1_{te}						L^2_{te}					
	Tr+	**Tr-**	**CV**	**Te+**	**Te-**	**AUC**	**Tr+**	**Tr-**	**CV**	**Te+**	**Te-**	**AUC**
ST-avg												
case only	4676	9242	.788	1091	2407	.780	5767	11649	.790	864	3412	.743
index-based 15			.820			.796			.816			.733
index-based 30			.820			.810			.820			.745
bag-of-activities			.885			.879			.885			.789
bag-of-activities+time			**.886**			**.880**			**.888**			**.794**
tree-based			.877			.868			.879			.784
ST-all												
case only	4676	9242	.831	1091	2407	.827	5767	11649	.831	864	3412	.765
index-based 15			.859			.850			.861			.738
index-based 30			.858			.860			.861			.771
bag-of-activities			**.908**			**.909**			.909			.793
bag-of-activities+time			.907			.906			**.910**			**.805**
tree-based			.897			.897			.900			.774
SH-avg												
case only	8703	41315	.762	2117	10470	.761	10820	51785	.764	1174	13908	.804
index-based 15			.761			.763			.766			.788
index-based 30			.759			.763			.760			.780
bag-of-activities			**.796**			**.793**			**.796**			**.826**
bag-of-activities+time			.785			.788			.790			.804
tree-based			.783			.784			.783			.796
SH-all												
case only	8703	41315	.776	2117	10470	.772	10820	51785	.776	1174	13908	.823
index-based 15			.778			.779			.781			.819
index-based 30			.776			.756			.775			.811
bag-of-activities			**.806**			**.803**			.807			**.842**
bag-of-activities+time			.798			.797			.799			.816
tree-based			.779			.792			**.816**			.805

Table 7.2: Summarized binary classification results for scenario 1 in *ST* and *SH*

Scenario 2

The results for scenario 2 are shown in fig. 7.4 and table 7.3.

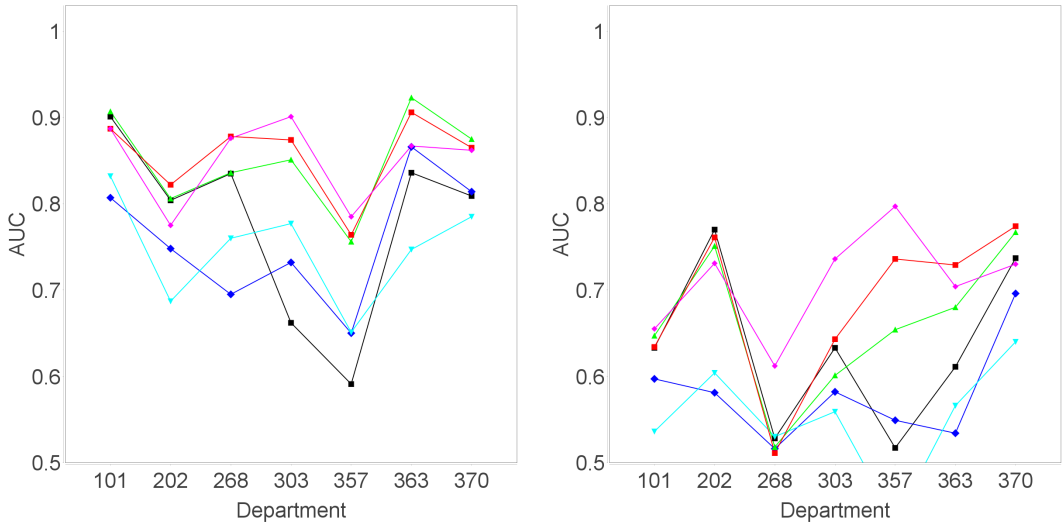
We see that in scenario 2 the more complex feature sets in general perform best. We attribute this to the fact that the decision to reopen a case and then again close it or not can be explained by factors internal to the process (as opposed to scenario 1).

Again, for all feature sets, in *ST* we observe a drop in performance when applied to L^2_{te}. The results for *SH* are again more stable. There is a notable exception in department 6: here, the class distribution changed drastically in 2013, which was a major problem for all trained models. While the tree-based trace profile performed best in the majority of departments, its performance dropped on L^2_{te} when applied to all departments, confirming the results from chapter 6 that it is generally harder to learn a process model that fits multiple departments.

Scenario 3

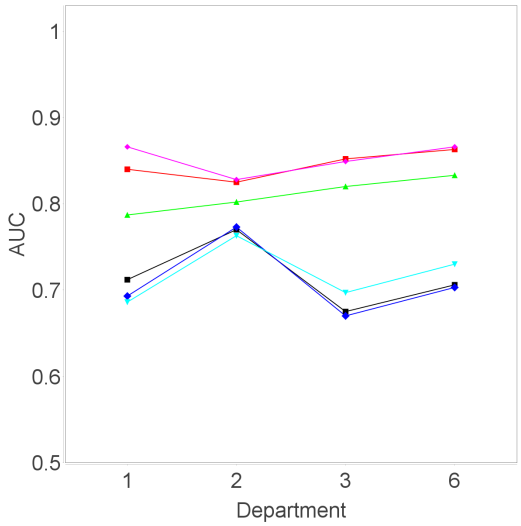
The results for scenario 3 are shown in fig. 7.5 and table 7.4.

In this scenario, the results are extremely noisy. We attribute this to the very low number of positive instances in the training set. See fig. 7.6 for ROC curves for different scenarios. The three curves (a - c) are much smoother than (d) for scenario 3, therefore we can have more confidence in the area under the former curves than in the latter. This means that even though most AUC values are larger than 0.5, we could hardly choose an optimal configuration in a practical application.

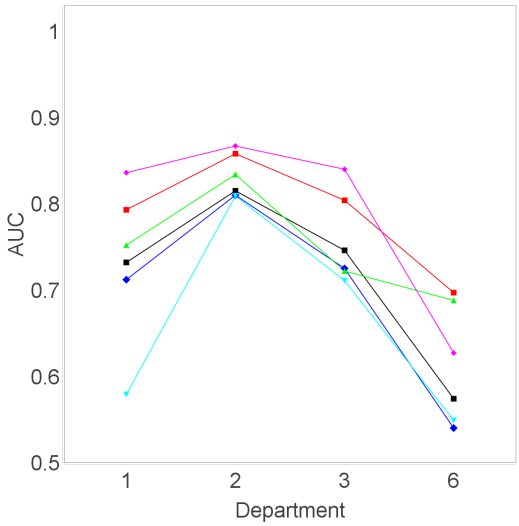


(a) L_{te}^1 performance in ST

(b) L_{te}^2 performance in ST



(c) L_{te}^1 performance in SH



(d) L_{te}^2 performance in SH

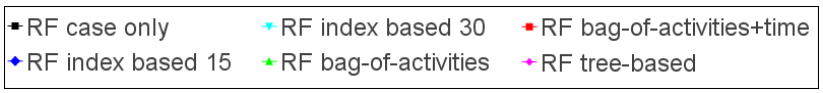


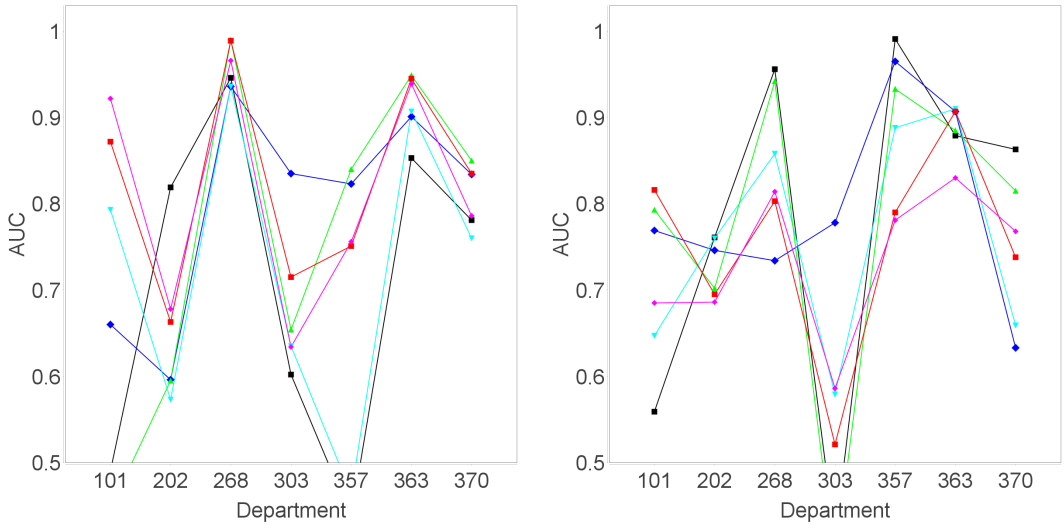
Figure 7.4: AUC values per department in scenario 2

	L^1_{te}						L^2_{te}					
	Tr+	**Tr-**	**CV**	**Te+**	**Te-**	**AUC**	**Tr+**	**Tr-**	**CV**	**Te+**	**Te-**	**AUC**
ST-avg												
case only	2089	2500	.742	499	569	.768	2588	3069	.751	339	502	.672
index-based 15			.763			.773			.766			.592
index-based 30			.741			.751			.755			.572
bag-of-activities			.829			.856			.843			.685
bag-of-activities+time			.837			**.862**			.850			.707
tree-based			**.852**			.854			**.855**			**.719**
ST-all												
case only	2089	2500	.795	499	569	.807	2588	3069	.801	339	502	.650
index-based 15			.812			.818			.812			.704
index-based 30			.792			.799			.800			.695
bag-of-activities			.872			**.880**			.875			**.752**
bag-of-activities+time			.876			.884			**.880**			.744
tree-based			**.880**			.875			.876			.722
SH-avg												
case only	3202	4931	.739	732	1240	.712	3934	6171	.735	519	564	.737
index-based 15			.744			.706			.734			.718
index-based 30			.732			.718			.734			.696
bag-of-activities			.835			.813			.832			.748
bag-of-activities+time			.845			.846			.852			.801
tree-based			**.853**			**.852**			**.853**			**.816**
SH-all												
case only	3202	4931	.747	732	1240	.720	3934	6171	.746	519	564	.760
index-based 15			.760			.730			.752			.749
index-based 30			.744			.730			.748			.729
bag-of-activities			.840			.818			.839			.779
bag-of-activities+time			**.859**			.854			.856			**.822**
tree-based			.858			**.856**			**.861**			.796

Table 7.3: Summarized binary classification results for scenario 2 in *ST* and *SH*

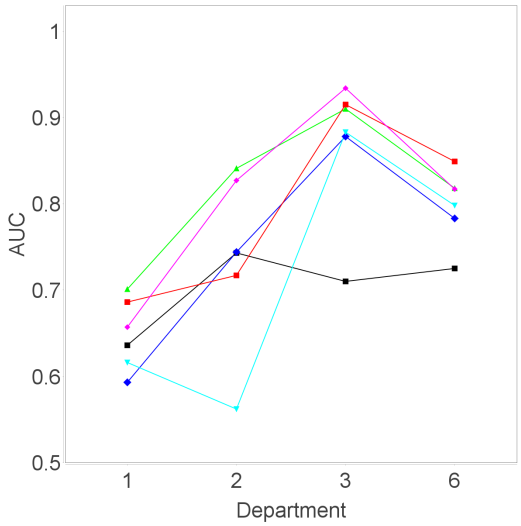
	L^1_{te}						L^2_{te}					
	Tr+	**Tr-**	**CV**	**Te+**	**Te-**	**AUC**	**Tr+**	**Tr-**	**CV**	**Te+**	**Te-**	**AUC**
ST-avg												
case only	449	13469	.784	116	3382	.704	565	16851	.734	84	4192	.751
index-based 15			.803			.798			.785			**.799**
index-based 30			.791			.740			.760			.756
bag-of-activities			**.835**			.750			.793			.764
bag-of-activities+time			.796			**.829**			.802			.758
tree-based			.841			.816			**.834**			.733
ST-all												
case only	449	13469	.892	116	3382	.888	565	16851	.886	84	4192	.791
index-based 15			.903			.902			.908			.737
index-based 30			.903			.907			.905			.777
bag-of-activities			**.939**			.934			**.934**			.749
bag-of-activities+time			.936			**.936**			.932			**.866**
tree-based			.927			.924			.930			.808
SH-avg												
case only	442	49576	.697	106	12481	.700	548	62057	.692	43	15039	.706
index-based 15			.766			.759			.772			**.753**
index-based 30			.766			.748			.787			.751
bag-of-activities			.809			**.820**			**.830**			.746
bag-of-activities+time			.793			.811			.803			.742
tree-based			**.822**			.814			.829			.653
SH-all												
case only	442	49576	.742	106	12481	.765	548	62057	.756	43	15039	.765
index-based 15			.836			.828			.825			.792
index-based 30			.819			.837			.828			.769
bag-of-activities			.873			.852			.874			.763
bag-of-activities+time			.867			.869			.873			**.793**
tree-based			**.875**			**.862**			**.879**			.730

Table 7.4: Summarized binary classification results for scenario 3 in *ST* and *SH*

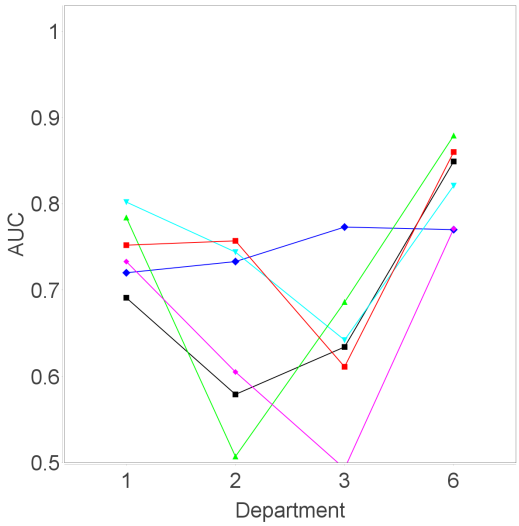


(a) L_{te}^1 performance in ST

(b) L_{te}^2 performance in ST



(c) L_{te}^1 performance in SH



(d) L_{te}^2 performance in SH

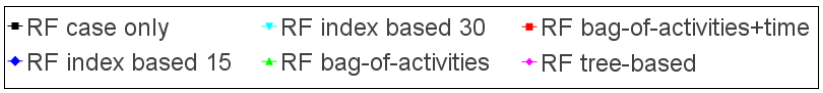
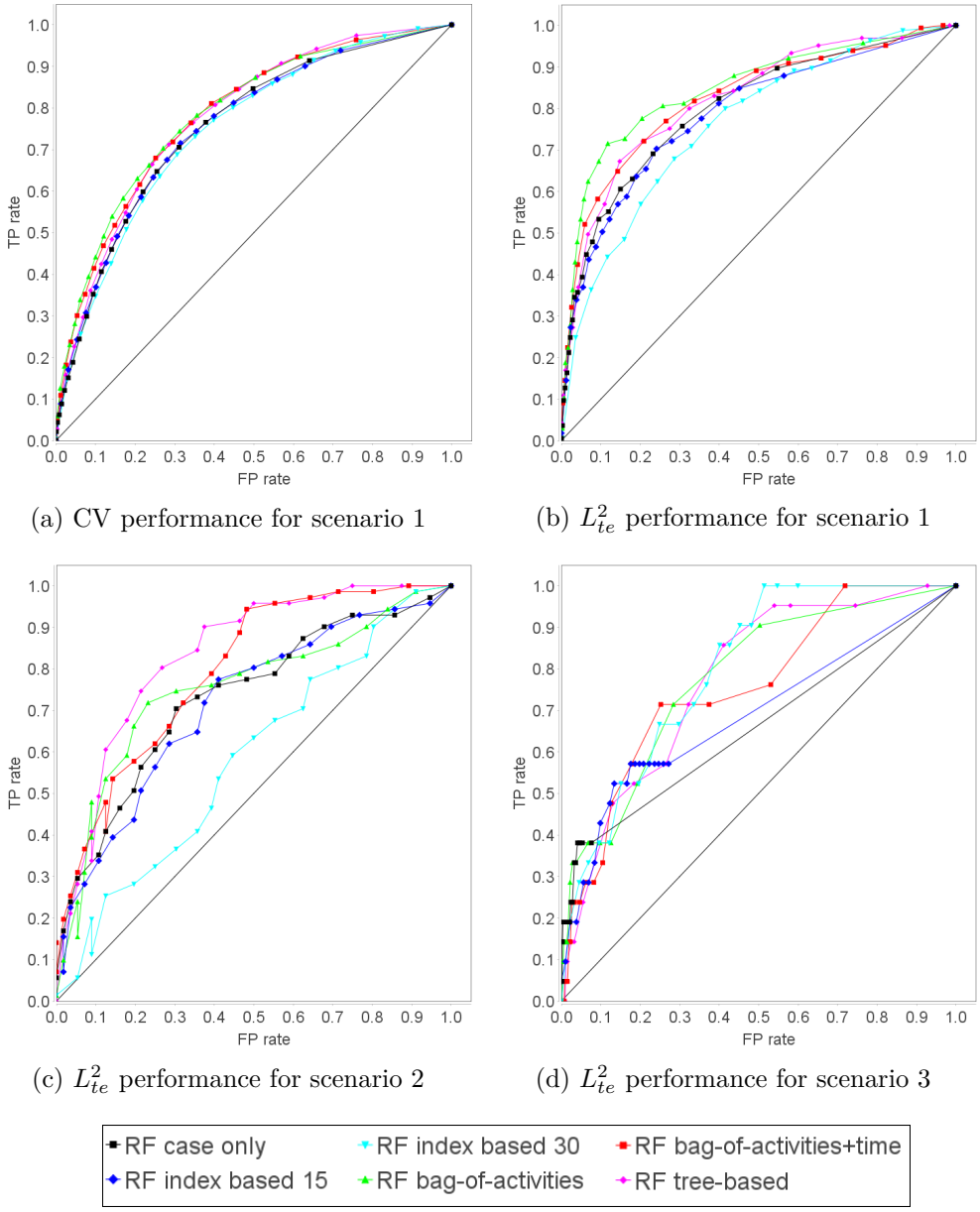


Figure 7.5: AUC values per department in scenario 3



(a) CV performance for scenario 1

(b) L_{te}^2 performance for scenario 1

(c) L_{te}^2 performance for scenario 2

(d) L_{te}^2 performance for scenario 3

Figure 7.6: ROC curves for different scenarios in department 1, *SH*

7.2.4 Discussion

In general, we found the combination of case attributes and simple bag-of-activities encoding to be a challenging baseline in our domain. We think that the presence, absence and repetition of certain activities is already a very good criterion to divide applications into groups. However, in some cases it was beneficial to include more complex features. In particular, the inclusion of absolute timestamps resulted in performance improvements. As we have seen in fig. 6.3, the processes in our domain all start every year at about the same time. We think that in domains where this is not the case, the merits of using absolute timestamps may be limited. The tree-based encodings outperformed the other trace profiles in a few cases, such as scenario 2 in *SH*. However, the difference are not very large. We think that the tree-based encoding suffers from the relatively low degree of structure across departments. The index-based encodings, on the other hand, often performed even worse than using only case attributes, therefore we conclude that this style of encoding is not appropriate in the presence of large variability.

7.2.5 Further experiments

Before settling on the final setup, we performed a number of other experiments whose results we do not discuss in detail, because they were only carried out in single scenarios in a rather unsystematic manner. Nevertheless, we gained some insights which we do not want to withhold:

- We conducted single experiments using other learning algorithms, namely Naive Bayes, C4.5, Logistic regression and a Linear SVM, all of which had lower cross validation performance than Random Forest
- We applied Principal Component Analysis [98] for unsupervised feature selection, which also did not improve cross validation performance
- We used a wrapper approach [55] for supervised feature selection using forward search, but evaluation of all feature subsets in combination with hyperparameter optimization took several days, so we discarded this approach for practical reasons. In fact, finding an optimal feature subset is an NP-hard problem [70]. However, we think that the tree learning procedure used by Random Foresst already does a good job at selecting features. For a detailed discussion of feature selection methods that also might have applied, see for example [51].

7.3 Outlier detection

Since binary classification does not seem to be appropriate when there are very little positive examples, we tried to tackle this situation as an outlier detection problem. The reasoning behind this is that, possibly, all normal instances are alike and there is a higher probability of an undesired outcome in cases which are atypical in some sense.

By adopting a semi-supervised approach, we also avoid the problem of learning models and predicting across years with potentially different distributions.

Profile name	σ	π	Γ	**Imput.**[1]	**# feat. *ST*-101**[2]	**# feat. *SH*-1**[2]
case only	-	-	-	-	12	12
bag-of-activities	single	count	bag	-	66	65
bag-of-activities+time	single	count	bag	-	228	224
		date	$\min'$	mean		
			$\max'$	mean		
			diff'	mean		
time w/o count	single	date	$\min'$	mean	174	171
			$\max'$	mean		
			diff'	mean		
tree-based	tree	count	bag	-	328	316
		date	$\min'$	mean		
			$\max'$	mean		
			diff'	mean		
		dur^3	$\min'$	mean		
			$\max'$	mean		
			sum'	mean		
tree-based w/o count	tree	date	$\min'$	mean	258	248
			$\max'$	mean		
			diff'	mean		
		dur^3	$\min'$	mean		
			$\max'$	mean		
			sum'	mean		

Table 7.5: Trace profiles used in outlier detection experiments
[1] Imputation value if Γ can return $\perp$
[2] Number of features when applied to L^2_{te}
[3] Only for $\rightarrow$, $\circlearrowleft$ and $\wedge$ nodes

7.3.1 Experimental design

In the process of managing applications for direct payments, it is required to decide manually about a 5% sample of applications, whereas the rest can be decided automatically. Instead of just selecting this sample at random, we propose an outlier detection approach, i.e., we create a sample containing (about) 5% of the most atypical instances.

We compare the results using several trace profiles, shown in table 7.5. Since we cannot rely on the learning algorithm to select the right features based on correlation with the labels, we use more profiles than before in order to detect different types of outliers. We omitted further investigation of the index-based encodings, since they do not seem to work very well in our application.

Note that it is somewhat hard to evaluate if the sample of outliers is meaningful for the domain. We chose to use the number of legal complaints (Scenario 3) in the sample, as well as an additional Scenario 4, considering the applications whose decision need to be reverted, which corresponds to the occurence of event "FP200 Entscheidung zurücknehmen" somewhere in the future. Both types of instances have in common that they are relatively rare, but it is still useful to detect them at an early stage, i.e., before the decision is made.

7.3.2 Implementation

Again, for each scenario and each trace profile, we considered *prefixes* of cases in the logs, cut off before the prediction point. We splitted the training logs (L_{tr}, L^1_{te}) by year, thus we had one sample per year 2009-2012.

We trained an OC-SVM with $\nu = 0.05$ and RBF kernel for each of these years to choose the value of the kernel parameter γ. Performance was evaluated by the average F-score over all training years w.r.t. the labels from the respective scenario.

We then trained the OC-SVM with these hyperparameters on L^2_{te} and predicted the labels on the same dataset, which should result in about 5% outlier instances. Even though we are actually interested in the recall (given the constant sample size), our measure of evaluation was the F-score to account for fluctuations in the actual sample size.

Since we use the labelled training data for hyperparameter selection but train the models on unlabelled data, we operate in a semi-supervised learning setting.

We applied min-max-normalization to rescale the features and relied on the imputation procedure implemented by the WEKA LibSVM wrapper, which replaced missing values with the attribute mean.

7.3.3 Results

Again, we report a summary of the outlier detection results. The detailed values for all departments can again be found in appendix C. Fig. 7.7 and 7.8 illustrate the true positive rates for scenario 3 and 4.

The averaged results over all departments and when applied to dataset of all departments combined are presented in tables 7.6 and 7.7 for scenario 3 and 4 respectively.

We report the number of positive and negative instances in the training set 2009-2012 (Tr+, Tr−) as well as the averaged F-score for the optimal value of γ. For the test set L^2_{te}, we report the number of positive and negative instances (Te+, Te−). To assess the performance of outlier detection, we report the number of true positives (TP), false positives (FP), true negatives (TN), false negatives (FN) and the derived values

for recall (R), precision (Pr) and F-score (F1). We also show the size of the sample returned by the OC-SVM, where sample sizes that are more than ±0.5% off the target of 5% are coloured red. The highest F-scores and highest numbers of TP are printed in bold face, the profile that achieved the highest F1-score on L^2_{te}, i.e. in 2013, is underlined. Again, we computed the average performance over all departments, this time by microaveraging the numbers of TP, FP, TN and FN.

The results for *ST* show that in both scenarios, using only case attributes without any trace profiles resulted in the best performance. When we included execution history by different trace profiles, the OC-SVM often did return much more than 5% outliers, which makes the results hard to apply in the motivating application. While the tree-based profiles achieved the highest recall on average in scenario 4, it resulted in about 8% being flagged as outliers.

The results in *SH* are again more stable, which we attribute to our findings in chapter 6, which indicated that traces in *SH* exhibit larger stability in the mainstream behaviour with fewer deviations than in *ST*. Additionally, more data is available per department, which we think leads to a smoother decision boundary. On average, the combination of bag-of-activities and timestamps performed best. When the dataset was not split by department, using only timestamps performed equally good.

7.3.4 Discussion

We find the results for scenario 3 in *SH* especially encouring: more than one third of all legal claims were contained in a 5% sample, without knowing anything about the relation of process execution and legal claims. Scenario 4 underlines the benefits of including trace information: the performance of using only case features was hardly better than a random guess.

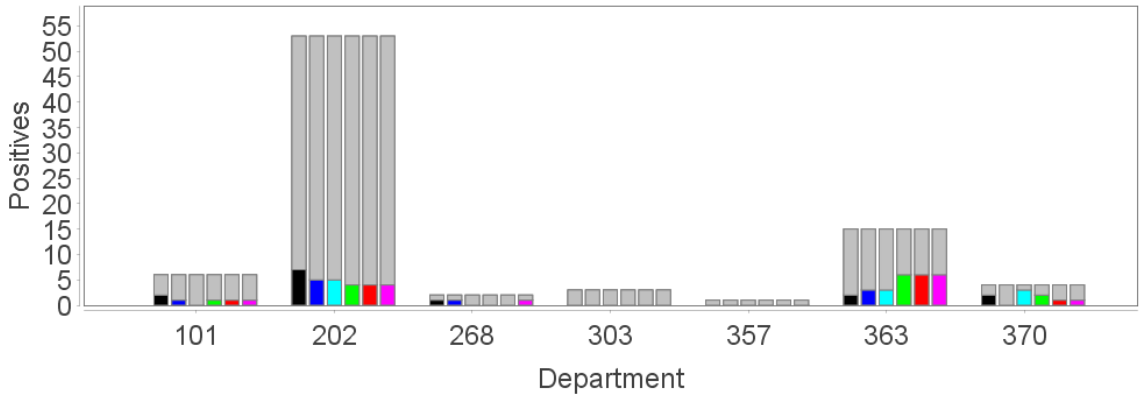
However, our evaluation measures are only rough indicators for the usefulness of the outlier detection approach. The samples may contain other kinds of anomalous behaviour, which might be interesting to detect but would not necessarily result in one of the actions we used for evaluation.

	L^2_{te}											
	Tr+	**Tr-**	**Tr-F1**	**Te+**	**Te-**	**TP**	**FP**	**TN**	**FN**	**R**	**Pr**	**F1**	**%**
ST-avg													
case only	565	16851	.079	84	4192	**14**	200	3992	70	.167	.065	**.094**	5.005
bag-of-activities			.070			10	217	3975	74	.119	.044	.064	5.309
bag-of-activities+time			.069			11	298	3894	73	.131	.036	.056	7.226
time w/o count			.072			13	347	3845	71	.155	.036	.059	8.419
tree-based			.074			12	382	3810	72	.143	.030	.050	9.214
tree-based w/o count			**.080**			13	425	3767	71	.155	.030	.050	1.243
ST-all													
case only	565	16851	.075	84	4192	**11**	198	3994	73	.131	.053	**.075**	4.888
bag-of-activities			.068			7	206	3986	77	.083	.033	.047	4.981
bag-of-activities+time			.074			7	207	3985	77	.083	.033	.047	5.005
time w/o count			**.078**			9	207	3985	75	.107	.042	.060	5.051
tree-based			**.078**			9	202	3990	75	.107	.043	.061	4.935
tree-based w/o count			.076			10	201	3991	74	.119	.047	.068	4.935
SH-avg													
case only	548	62057	.046	43	15039	12	740	14299	31	.279	.016	.030	4.986
bag-of-activities			.051			11	742	14297	32	.256	.015	.028	4.993
bag-of-activities+time			.045			**15**	733	14306	28	.349	.020	**.038**	4.960
time w/o count			.040			**15**	749	14290	28	.349	.020	.037	5.066
tree-based			.045			13	782	14257	30	.302	.016	.031	5.271
tree-based w/o count			.041			12	774	14265	31	.279	.015	.029	5.212
SH-all													
case only	548	62057	.044	43	15039	13	737	14302	30	.302	.017	.033	4.97
bag-of-activities			**.046**			11	747	14292	32	.256	.015	.027	5.03
bag-of-activities+time			.044			11	757	14282	32	.256	.014	.027	5.09
time w/o count			.040			**15**	743	14296	28	.349	.020	**.037**	5.03
tree-based			.044			11	779	14260	32	.256	.014	.026	5.24
tree-based w/o count			.036			13	746	14293	30	.302	.017	.032	5.03

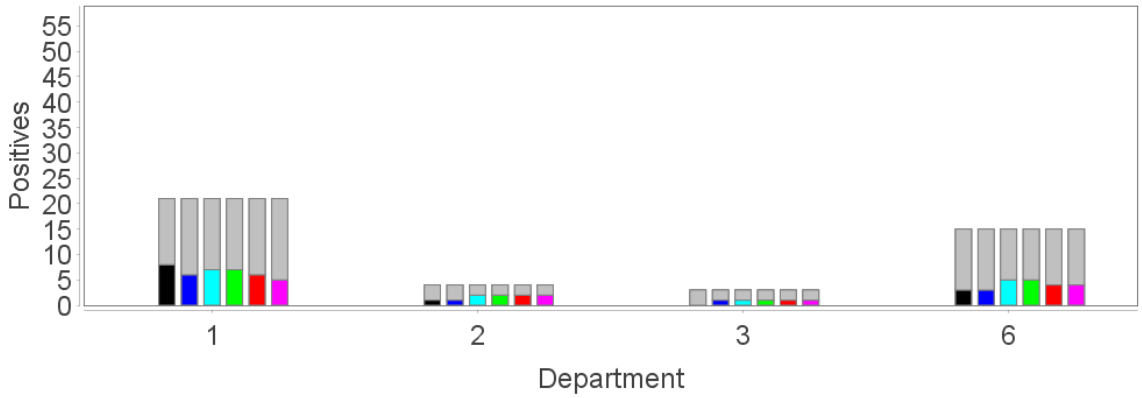
Table 7.6: Summarized outlier detection results for scenario 3 in *ST* and *SH*

	L^2_{te}											
	Tr+	**Tr-**	**Tr-F1**	**Te+**	**Te-**	**TP**	**FP**	**TN**	**FN**	**R**	**Pr**	**F1**	**%**
ST-avg													
case only	291	17125	.076	46	4230	7	207	4023	39	.152	.033	**.054**	5.005
bag-of-activities			**.087**			4	215	4015	42	.087	.018	.030	5.122
bag-of-activities+time			.079			8	290	3940	38	.174	.027	.047	6.969
time w/o count			.081			9	364	3866	37	.196	.024	.043	8.723
tree-based			.078			**10**	331	3899	36	.217	.029	.052	7.975
tree-based w/o count			.074			9	367	3863	37	.196	.024	.043	8.793
ST-all													
case only	291	17125	.062	46	4230	6	206	4024	40	.130	.028	**.047**	4.958
bag-of-activities			**.080**			5	208	4022	41	.109	.023	.039	4.981
bag-of-activities+time			.064			5	209	4021	41	.109	.023	.038	5.005
time w/o count			.069			4	267	3963	42	.087	.015	.025	6.338
tree-based			.063			**7**	253	3977	39	.152	.027	.046	6.080
tree-based w/o count			.071			5	230	4000	41	.109	.021	.036	5.496
SH-avg													
case only	722	61883	.033	95	14987	7	992	13995	88	.074	.007	.013	6.62
bag-of-activities			.050			14	736	14251	81	.147	.019	.033	4.97
bag-of-activities+time			**.065**			**20**	721	14266	75	.211	.027	**.048**	4.91
time w/o count			.062			19	777	14210	76	.200	.024	.043	5.28
tree-based			.063			18	796	14191	77	.189	.022	.04	5.40
tree-based w/o count			.061			19	734	14253	76	.200	.025	.045	4.99
SH-all													
case only	722	61883	.032	95	14987	8	744	14243	87	.084	.011	.019	4.99
bag-of-activities			.052			16	742	14245	79	.168	.021	.038	5.03
bag-of-activities+time			.059			19	749	14238	76	.200	.025	.044	5.09
time w/o count			.056			**20**	744	14243	75	.211	.026	**.047**	5.07
tree-based			**.067**			19	749	14238	76	.200	.025	.044	5.09
tree-based w/o count			.060			**20**	744	14243	75	.211	.026	**.047**	5.07

Table 7.7: Summarized outlier detection results for scenario 4 in *ST* and *SH*



(a) L^2_{te} performance for scenario 3 in ST



(b) L^2_{te} performance for scenario 3 in SH

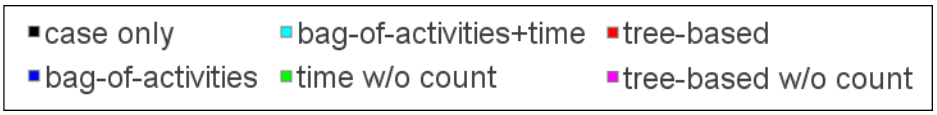
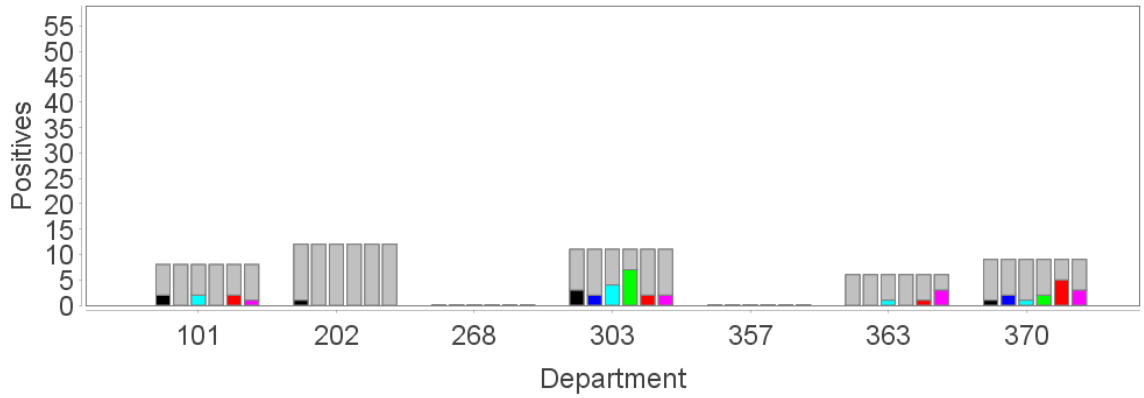
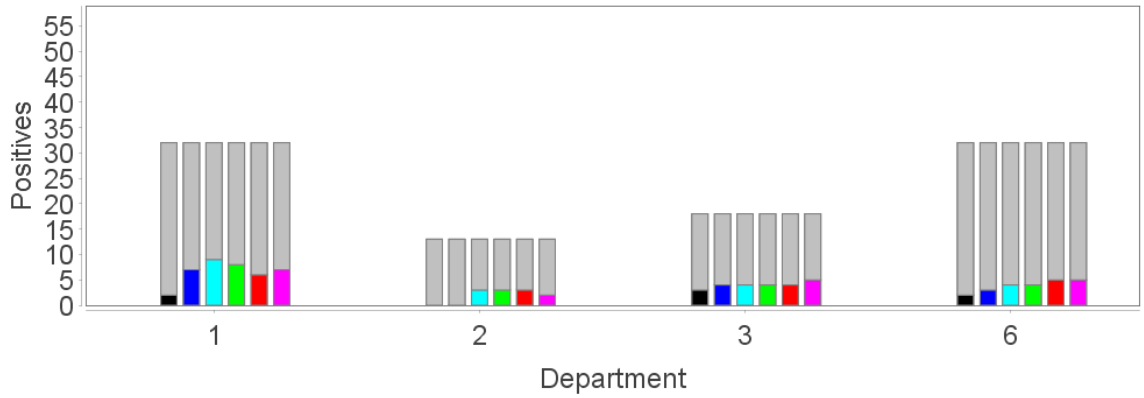


Figure 7.7: Recall for outlier detection scenario 3 in ST and SH. The coloured bars show the number of true positives for different trace profiles, the grey bars indicate the total number of positives in the test set



(a) L^2_{te} performance for scenario 4 in ST



(b) L^2_{te} performance for scenario 4 in SH

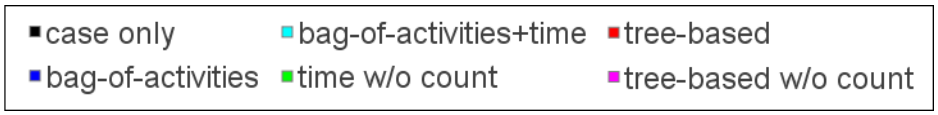


Figure 7.8: Recall for outlier detection scenario 4 in ST and SH. The coloured bars show the number of true positives for different trace profiles, the grey bars indicate the total number of positives in the test set

8 Conclusion & Outlook

The goal of this thesis was to show how different aspects of event log traces can be turned into features for predictive monitoring of business processes. We discussed several proposals from the literature and developed an algorithm that can capture hierarchical business process structures in traces. To this end, we employed the Inductive Miner to construct an intermediate process tree of these structures and explicitly deal with concurrency and noise.

We used these trace profiles for binary classification and outlier detection to proactively predict undesired behaviour using event logs from the public administration. The results showed that simple features, like a combination of a bag-of-activities with timestamps, were a very challenging baseline in most cases.

We actually think that this is good news, as one can start with these simple feature sets and then analyze if it is a good idea to construct more complex features, which was beneficial is some of our experiments but not in others. In the cases where simple features performed best, it might be more judicious to obtain more types of events or more handcrafted case features instead of working on a better trace profile. In particular, which trace profiles worked best depended on the type of prediction and supply of training data. In our experiments, we also addressed the impact of variability across organizational units and concept drift taking place over time.

We found the outlier detection approach to be particularly interesting for practical applications, as it allowed us to pick a sample of predefined size without the need to have many labelled instances for very rare but undesirable outcomes. However, this approach required a relatively large and homogenous amount of data to work well. Initial descriptive analysis, using basic process mining tools, provided some good hints on which type of predictions and features may be worthwile for a particular dataset.

8.1 Using our results in practice

The technical implementation of the process whose historical logs we used has drastically changed over the last years. While the current process should be conceptually somewhat similar, the models we learned are not directly applicable for current event logs.

To apply our findings to this and other future projects in our domain, we recommend the following strategy:

1. Extract event logs for the current processes and do some initial analysis with tools such as ProM or Disco

2. Discuss the results with a domain expert to find interesting opportunities for predictions
3. Ask the domain expert for good features that she thinks might influence the prediction target
4. Use these features as case attributes and create trace profiles using the techniques presented in this thesis
5. Obtain some initial results as a baseline for predictive performance
6. Ask the domain expert if the results make sense
7. Analyze what to do next:
   - Is it possible to obtain more training examples?
   - Is it possible to include more events?
   - Does it help to use more complex features?
   - Can we get better handcrafted (case) features?
   - Shall we redefine the prediction target?
   - Can we phrase the problem as an unsupervised, e.g., outlier detection or clustering problem?
8. Iterate 5.-7. until the results are satisfactory

8.2 Future work

8.2.1 Cost functions

Our evaluation was based on standard performance measures, such as AUC and F-score. Dependending on the application, false positives and false negatives may have very different "costs", e.g., missing a positive legal claim may be much worse than inspecting a false positive. Providing a good cost matrix may therefore yield results which are favourable in practice [35]. We could also think of incorporating a reject option when the classifier is very insecure about a certain instance [11].

8.2.2 Representation learning

Our feature sets were based on relatively boring mathematical functions. A promising field for future research would be the application of artifical neural networks for feature learning. Deep networks have shown their ability to learn multiple levels of abstraction [57], as sketched in fig. 8.1 for a face recognition network. This is a property that might elegantly apply to hierachical constructs present in business processes. It would be very interesting to see how deep learned features compare to our semi-handcrafted hierarchical representations.

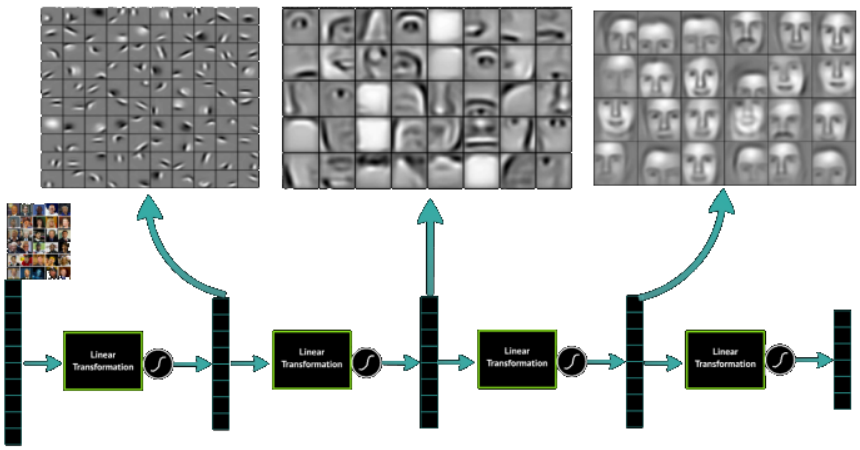


Figure 8.1: Multiple levels of features for face recognition in a deep network, from [9]

8.2.3 Interpretable models

While we focused on predictive performance, interpretable models may be of interest for stakeholders. We can think of different ways to achieve this. The simplest one would be to use more transparent learning algorithms, even though we found the performance of some of them to be inferior to Random Forests in our experiments. The sequential dependency structures could be explicitly modelled using probabilistic graphical models like Hidden Markov models or Bayesian networks. In general, Bayesian approaches could allow for modelling assumptions explicitly in the form of appropriate prior distributions.

8.2.4 Outlier detection

Our approach to outlier detection could be extended in multiple directions: first of all, an outlier score might be more interesting than a simple discriminant function. This could be achieved by explicit density estimation of the probability distribution of cases. Furthermore, more sophisticated extensions of the OC-SVM have been propopsed: [46] introduce a Hidden Markov anomaly detection approach that deals with latent dependency structures in sequences, as they are present in event log traces.

8.2.5 Other datasets

Our experiments were restricted to two relatively similar datasets, limiting the validity of our findings. We could easily adapt our experimental setup to other processes. The software system whose event logs we used supports other kinds of processes beside the management of direct payments, where other types of predictions could be of interest. These logs could be easily extracted from the system in much the same way as we did. Using standard datasets, i.e., from Business Process Intelligence challenges [75], we could also compare our approach directly to other work.

Bibliography

[1] W. M. P. van der Aalst. "Process Mining: Discovery, Conformance and Enhancement of Business Processes". 1st. Springer Publishing Company, Incorporated, 2011 (cited on pages 5, 6, 8).

[2] W. M. P. van der Aalst, A. A. de Medeiros, and A. Weijters. "Genetic process mining". In: *International Conference on Application and Theory of Petri Nets*. Springer. 2005, pp. 48–69 (cited on page 9).

[3] W. M. P. van der Aalst, M. H. Schonenberg, and M. Song. "Time prediction based on process mining". In: *Information Systems* 36.2 (2011), pp. 450–475 (cited on page 22).

[4] W. M. P. van der Aalst and M. Song. "Mining Social Networks: Uncovering interaction patterns in business processes". In: *International Conference on Business Process Management*. Springer. 2004, pp. 244–260 (cited on page 9).

[5] W. M. P. van der Aalst, T. Weijters, and L. Maruster. "Workflow mining: Discovering process models from event logs". In: *IEEE Transactions on Knowledge and Data Engineering* 16.9 (2004), pp. 1128–1142 (cited on page 9).

[6] W. M. P. van der Aalst et al. "Process mining manifesto". In: *Business process management workshops*. Springer. 2011, pp. 169–194 (cited on page 10).

[7] A. Adriansyah, B. F. van Dongen, and W. M. van der Aalst. "Conformance checking using cost-based fitness analysis". In: *Enterprise Distributed Object Computing Conference (EDOC), 2011 15th IEEE International*. IEEE. 2011, pp. 55–64 (cited on page 38).

[8] A. Adriansyah et al. "Alignment Based Precision Checking". English. In: *Business Process Management Workshops*. Ed. by M. La Rosa and P. Soffer. Vol. 132. Lecture Notes in Business Information Processing. Springer Berlin Heidelberg, 2013, pp. 137–149 (cited on page 10).

[9] D. Akagi. "A primer on deep learning". https://www.datarobot.com/blog/a-primer-on-deep-learning/. Accessed 2016-09-09 (cited on page 77).

[10] W. J. Anderson. "Continuous-time Markov chains: An applications-oriented approach". Springer Science & Business Media, 2012 (cited on page 13).

[11] P. L. Bartlett and M. H. Wegkamp. "Classification with a reject option using a hinge loss". In: *Journal of Machine Learning Research* 9.Aug (2008), pp. 1823–1840 (cited on page 76).

[12] D. J. Berndt and J. Clifford. "Using Dynamic Time Warping to Find Patterns in Time Series." In: *KDD workshop*. Vol. 10. 16. Seattle, WA. 1994, pp. 359–370 (cited on page 12).

[13] C. M. Bishop. "Pattern Recognition and Machine Leaning". 1st ed. Springer, 2006 (cited on page 13).

[14] R. P. J. C. Bose and W. M. P. van der Aalst. "Abstractions in process mining: A taxonomy of patterns". In: *Business Process Management*. Springer, 2009, pp. 159–175 (cited on pages 20, 21).

[15] R. P. J. C. Bose and W. M. P. van der Aalst. "Context Aware Trace Clustering: Towards Improving Process Mining Results." In: *SDM*. SIAM. 2009, pp. 401–412 (cited on page 19).

[16] R. P. J. C. Bose and W. M. P. van der Aalst. "Discovering signature patterns from event logs". In: *Computational Intelligence and Data Mining (CIDM), 2013 IEEE Symposium on*. IEEE. 2013, pp. 111–118 (cited on page 21).

[17] R. P. J. C. Bose and W. M. P. van der Aalst. "Trace clustering based on conserved patterns: Towards achieving better process models". In: *Business Process Management Workshops*. Springer. 2009, pp. 170–181 (cited on page 20).

[18] R. P. J. C. Bose et al. "Dealing with concept drifts in process mining". In: *Neural Networks and Learning Systems, IEEE Transactions on* 25.1 (2014), pp. 154–171 (cited on pages 17, 19, 20).

[19] L. Breiman. "Random forests". In: *Machine Learning* 45.1 (2001), pp. 5–32 (cited on page 39).

[20] L. Breiman et al. "Statistical modeling: The two cultures (with comments and a rejoinder by the author)". In: *Statistical Science* 16.3 (2001), pp. 199–231 (cited on page 2).

[21] L. Breiman et al. "Classification and regression trees". CRC press, 1984 (cited on page 39).

[22] D. Breuker et al. "Comprehensible Predictive Models for Business Processes". In: *MIS Quarterly* forthcoming (2016). Publication status: In press (cited on page 21).

[23] J. C. Buijs, B. F. van Dongen, and W. M. van der Aalst. "A genetic algorithm for discovering process trees". In: *2012 IEEE Congress on Evolutionary Computation*. IEEE. 2012, pp. 1–8 (cited on page 29).

[24] J. C. Buijs, B. F. Van Dongen, and W. M. van der Aalst. "On the role of fitness, precision, generalization and simplicity in process discovery". In: *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*. Springer. 2012, pp. 305–322 (cited on page 9).

[25] M. Ceci et al. “Completion Time and Next Activity Prediction of Processes Using Sequential Pattern Mining”. In: *Discovery Science: 17th International Conference, DS 2014, Bled, Slovenia, October 8-10, 2014. Proceedings*. Ed. by S. Džeroski et al. Cham: Springer International Publishing, 2014, pp. 49–61 (cited on page 22).

[26] V. Chandola, A. Banerjee, and V. Kumar. “Anomaly detection: A survey”. In: *ACM computing surveys (CSUR)* 41.3 (2009), p. 15 (cited on page 40).

[27] C.-C. Chang and C.-J. Lin. “LibSVM - A Library for Support Vector Machines”. https://www.csie.ntu.edu.tw/~cjlin/libsvm/. Accessed 2016-08-24 (cited on page 57).

[28] R. Conforti et al. “A recommendation system for predicting risks across multiple business process instances”. In: *Decision Support Systems* 69 (2015), pp. 1–19 (cited on page 22).

[29] R. Conforti et al. “BPMN Miner: Automated discovery of BPMN process models with hierarchical structure”. In: *Information Systems* 56 (2016), pp. 284–303 (cited on page 2).

[30] R. Conforti et al. “PRISM–A Predictive Risk Monitoring Approach for Business Processes”. In: *International Conference on Business Process Management*. Springer. 2016, pp. 383–400 (cited on page 22).

[31] A. A. De Medeiros. “Genetic process mining”. PhD thesis. Technische Universiteit Eindhoven, 2006 (cited on page 54).

[32] C. Di Francescomarino et al. “Predictive Business Process Monitoring Framework with Hyperparameter Optimization”. In: *International Conference on Advanced Information Systems Engineering*. Springer. 2016, pp. 361–376 (cited on page 21).

[33] T. G. Dietterich. “Machine Learning for Sequential Data: A Review”. In: *Structural, Syntactic, and Statistical Pattern Recognition*. Ed. by T. Caelli et al. Vol. 2396. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2002, pp. 15–30 (cited on page 21).

[34] École Polytechnique Fédérale de Lausanne. “Scala programming language”. http://www.scala-lang.org/. Accessed 2016-09-02 (cited on page 57).

[35] C. Elkan. “The foundations of cost-sensitive learning”. In: *International joint conference on artificial intelligence*. Vol. 17. 1. LAWRENCE ERLBAUM ASSOCIATES LTD. 2001, pp. 973–978 (cited on page 76).

[36] European Union. “Annex 1: Indicative figures on the distribution of aid, by size-class of aid, received in the context of direct aid paid the producers according to coundil regulation (EC)NO 73/2009”. Financial year 2014 (cited on page 47).

[37] European Union. “The EU explained: Agriculture”. European Commission, Directorate-General for Communication, Brussels. 2014 (cited on page 47).

[38] J. Evermann, J.-R. Rehse, and P. Fettke. “A Deep Learning Approach for Predicting Process Behaviour at Runtime”. In: *1st International Workshop on Runtime Analysis of Process-Aware Information Systems*. 2016 (cited on page 21).

[39] B. Fazzinga et al. “Classifying Traces of Event Logs on the Basis of Security Risks”. In: *New Frontiers in Mining Complex Patterns*. Springer, 2015, pp. 108–124 (cited on page 22).

[40] M. Fernández-Delgado et al. “Do we need hundreds of classifiers to solve real world classification problems?” In: *J. Mach. Learn. Res* 15.1 (2014), pp. 3133–3181 (cited on page 39).

[41] D. R. Ferreira and D. Gillblad. “Discovering process models from unlabelled event logs”. In: *International Conference on Business Process Management*. Springer. 2009, pp. 143–158 (cited on page 5).

[42] Fluxicon. “Disco”. `https://fluxicon.com/disco/`. Accessed 2016-08-23 (cited on page 15).

[43] F. Folino, M. Guarascio, and L. Pontieri. “Discovering context-aware models for predicting business process performances”. In: *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*. Springer. 2012, pp. 287–304 (cited on page 22).

[44] F. Folino et al. “Mining usage scenarios in business processes: Outlier-aware discovery and run-time prediction”. In: *Data & Knowledge Engineering* 70.12 (2011), pp. 1005–1029 (cited on page 20).

[45] T. Gärtner. “A survey of kernels for structured data”. In: *ACM SIGKDD Explorations Newsletter* 5.1 (2003), pp. 49–58 (cited on page 12).

[46] N. Goernitz, M. Braun, and M. Kloft. “Hidden markov anomaly detection”. In: *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*. 2015, pp. 1833–1842 (cited on page 77).

[47] C. W. Günther and W. M. P. van der Aalst. “Fuzzy mining–adaptive process simplification based on multi-perspective metrics”. In: *Business Process Management*. Springer, 2007, pp. 328–343 (cited on page 9).

[48] C. W. Günther and E. Verbeek. “OpenXES - Developer Guide”. Tech. rep. Technische Universiteit Eindhoven, Mar. 28, 2014 (cited on pages 14, 57).

[49] C. W. Günther and E. Verbeek. “XES - Standard Definition”. Tech. rep. Technische Universiteit Eindhoven, Mar. 28, 2014 (cited on page 15).

[50] M. Gupta et al. “Outlier detection for temporal data”. In: *Synthesis Lectures on Data Mining and Knowledge Discovery* 5.1 (2014), pp. 1–129 (cited on page 19).

[51] I. Guyon et al. “Feature extraction: foundations and applications”. Vol. 207. Springer, 2008 (cited on page 68).

[52] T. Hastie, R. Tibshirani, and J. Friedman. "The Elements of Statistical Learning". Springer Series in Statistics. New York, NY, USA: Springer New York Inc., 2001 (cited on pages 12, 39, 42, 45).

[53] S. Hochreiter and J. Schmidhuber. "Long short-term memory". In: *Neural computation* 9.8 (1997), pp. 1735–1780 (cited on page 13).

[54] A. Karpathy. "The Unreasonable Effectiveness of Recurrent Neural Networks". http://karpathy.github.io/2015/05/21/rnn-effectiveness/. Accessed 2016-08-21 (cited on page 13).

[55] R. Kohavi and G. H. John. "Wrappers for feature subset selection". In: *Artificial intelligence* 97.1 (1997), pp. 273–324 (cited on page 68).

[56] G. T. Lakshmanan et al. "A markov prediction model for data-driven semi-structured business processes". In: *Knowledge and Information Systems* 42.1 (2015), pp. 97–126 (cited on page 21).

[57] H. Lee et al. "Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations". In: *Proceedings of the 26th annual international conference on machine learning*. ACM. 2009, pp. 609–616 (cited on page 76).

[58] S. J. Leemans, D. Fahland, and W. M. P. van der Aalst. "Scalable process discovery with guarantees". In: *Enterprise, Business-Process and Information Systems Modeling*. Springer, 2015, pp. 85–101 (cited on pages 9, 29, 37).

[59] S. J. Leemans, D. Fahland, and W. M. van der Aalst. "Discovering block-structured process models from event logs - a constructive approach". In: *Application and Theory of Petri Nets and Concurrency*. Springer, 2013, pp. 311–329 (cited on pages 9, 28, 29, 35, 37).

[60] S. J. Leemans, D. Fahland, and W. M. van der Aalst. "Discovering block-structured process models from event logs containing infrequent behaviour". In: *Business Process Management Workshops*. Springer. 2013, pp. 66–78 (cited on pages 9, 29, 37, 57).

[61] S. J. Leemans, D. Fahland, and W. M. van der Aalst. "Discovering block-structured process models from incomplete event logs". In: *Application and Theory of Petri Nets and Concurrency*. Springer, 2014, pp. 91–110 (cited on pages 9, 29, 37).

[62] A. Leontjeva et al. "Complex Symbolic Sequence Encodings for Predictive Monitoring of Business Processes". In: *Business Process Management*. Springer, 2015, pp. 297–313 (cited on pages 20, 21, 26, 59).

[63] N. Lesh, M. J. Zaki, and M. Ogihara. "Mining features for sequence classification". In: *Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 1999, pp. 342–346 (cited on page 20).

[64] G. Louppe. “Understanding Random Forests: From Theory to Practice”. PhD thesis. Universite de Liege, Liege, Belgique, 2014 (cited on page 40).

[65] Machine Learning Group at the University of Waikato. “Weka 3: Data Mining Software in Java”. http://www.cs.waikato.ac.nz/ml/weka/. Accessed 2016-08-18 (cited on page 57).

[66] F. M. Maggi et al. “Predictive Monitoring of Business Processes”. English. In: *Advanced Information Systems Engineering*. Ed. by M. Jarke et al. Vol. 8484. Lecture Notes in Computer Science. Springer International Publishing, 2014, pp. 457–472 (cited on pages 19, 21).

[67] A. Metzger et al. “Comparing and combining predictive business process monitoring techniques”. In: *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 45.2 (2015), pp. 276–290 (cited on page 21).

[68] M. Mohri, A. Rostamizadeh, and A. Talwalkar. “Foundations of machine learning”. MIT press, 2012 (cited on pages 41, 43).

[69] J. Muñoz-Gama and J. Carmona. “A Fresh Look at Precision in Process Conformance”. English. In: *Business Process Management*. Ed. by R. Hull, J. Mendling, and S. Tai. Vol. 6336. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2010, pp. 211–226 (cited on page 10).

[70] A. Y. Ng. “On feature selection: learning with exponentially many irrevelant features as training examples”. In: *Massachusetts Institute of Technology, Dept. of Electrical Engineering and Computer Science* (1998) (cited on page 68).

[71] N. Oliver and E. Horvitz. “A comparison of HMMs and dynamic Bayesian networks for recognizing office activities”. In: *International conference on user modeling*. Springer. 2005, pp. 199–209 (cited on page 13).

[72] A. Pika et al. “Predicting deadline transgressions using event logs”. In: *Business Process Management Workshops*. Springer. 2012, pp. 211–216 (cited on page 22).

[73] A. Pika et al. “Evaluating and predicting overall process risk using event logs”. In: *Information Sciences* 352 (2016), pp. 98–120 (cited on page 22).

[74] J. Platt. “Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods”. In: *Advances in large margin classifiers* 10.3 (1999), pp. 61–74 (cited on page 44).

[75] Process Mining Group at Eindhoven Technical University. “BPI challenge”. http://www.win.tue.nl/bpi. Accessed 2016-09-24. 2016 (cited on page 77).

[76] Process Mining Group at Eindhoven Technical University. “ProM Tools”. http://www.promtools.org/. Accessed 2016-08-22 (cited on page 15).

[77] J. R. Quinlan. “C4.5: Programs for Machine Learning”. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1993 (cited on page 39).

[78] L. Rabiner and B. Juang. “An introduction to hidden Markov models”. In: *IEEE Assp Magazine* 3.1 (1986), pp. 4–16 (cited on page 13).

[79] A. Rogge-Solti and G. Kasneci. “Temporal anomaly detection in business processes”. In: *Business Process Management*. Springer, 2014, pp. 234–249 (cited on page 22).

[80] A. Rogge-Solti, L. Vana, and J. Mendling. “Time Series Petri Net Models”. In: *SIMPDA* (2015) (cited on page 22).

[81] A. Rogge-Solti and M. Weske. “Prediction of remaining service execution time using stochastic petri nets with arbitrary firing delays”. In: *Service-Oriented Computing*. Springer, 2013, pp. 389–403 (cited on page 22).

[82] A. Rozinat and W. M. van der Aalst. “Conformance checking of processes based on monitoring real behavior”. In: *Information Systems* 33.1 (2008), pp. 64–95 (cited on page 10).

[83] A. Rozinat and W. M. van der Aalst. “Decision mining in ProM”. In: *International Conference on Business Process Management*. Springer. 2006, pp. 420–425 (cited on page 19).

[84] B. Schölkopf et al. “Estimating the support of a high-dimensional distribution”. In: *Neural computation* 13.7 (2001), pp. 1443–1471 (cited on pages 40, 41).

[85] H. Schonenberg et al. “Supporting flexible processes through recommendations based on history”. In: *International Conference on Business Process Management*. Springer. 2008, pp. 51–66 (cited on page 22).

[86] M. Song and W. M. van der Aalst. “Supporting process mining by showing events at a glance”. In: *Proceedings of the 17th Annual Workshop on Information Technologies and Systems (WITS)*. 2007, pp. 139–145 (cited on page 50).

[87] M. Song, C. W. Günther, and W. M. P. van der Aalst. “Trace clustering in process mining”. In: *Business Process Management Workshops*. Springer. 2008, pp. 109–120 (cited on pages 19, 20, 22, 24, 25).

[88] S. Suriadi et al. “Root cause analysis with enriched process logs”. In: *Business Process Management Workshops*. Springer. 2012, pp. 174–186 (cited on page 20).

[89] I. Teinemaa et al. “Predictive Business Process Monitoring with Structured and Unstructured Data”. In: *International Conference on Business Process Management*. Springer. 2016, pp. 401–417 (cited on page 21).

[90] T. Thaler et al. “A Comparative Analysis of Process Instance Cluster Techniques.” In: *Wirtschaftsinformatik*. 2015, pp. 423–437 (cited on page 19).

[91] M. Unuvar, G. T. Lakshmanan, and Y. N. Doganata. “Leveraging path information to generate predictions for parallel business processes”. In: *Knowledge and Information Systems* 47.2 (2016), pp. 433–461 (cited on page 20).

[92] I. Verenich et al. “Complex symbolic sequence clustering and multiple classifiers for predictive process monitoring”. In: *11th International Workshop on Business Process Intelligence 2015* (2015) (cited on page 21).

[93] A. Weijters and J. Ribeiro. “Flexible heuristics miner (FHM)”. In: *Computational Intelligence and Data Mining (CIDM), 2011 IEEE Symposium on*. IEEE. 2011, pp. 310–317 (cited on pages 9, 37, 54).

[94] J. M. E. van der Werf et al. “Process discovery using integer linear programming”. In: *International Conference on Applications and Theory of Petri Nets*. Springer. 2008, pp. 368–387 (cited on page 9).

[95] Wikipedia, the free encyclopedia. “A process with decisions for creating a new bank customer”. `https://en.wikipedia.org/wiki/File:DMN_NewBankAccountProcess_BPMN.jpg`. Accessed 2016-08-14 (cited on page 1).

[96] Wikipedia, the free encyclopedia. “An example of a CART classification tree”. `https://commons.wikimedia.org/wiki/File:CART_tree_titanic_survivors.png`. Accessed 2016-09-04 (cited on page 40).

[97] I. H. Witten, E. Frank, and M. A. Hall. “Data Mining: Practical Machine Learning Tools and Techniques”. 3rd. Morgan Kaufmann Publishers Inc., 2011 (cited on pages 42, 44).

[98] S. Wold, K. Esbensen, and P. Geladi. “Principal component analysis”. In: *Chemometrics and intelligent laboratory systems* 2.1-3 (1987), pp. 37–52 (cited on page 68).

[99] Z. Xing, J. Pei, and E. Keogh. “A brief survey on sequence classification”. In: *ACM SIGKDD Explorations Newsletter* 12.1 (2010), pp. 40–48 (cited on page 11).

[100] S. van Zelst, B. van Dongen, and W. van der Aalst. “ILP-based process discovery using hybrid regions”. In: *Technische Universiteit Eindhoven* (2015) (cited on page 9).

A Appendix: XES example

Listing A.1 beginning of a trace in an XES file

```
<!-- This file has been generated with the OpenXES library. -->
<!-- It conforms to the XML serialization of the XES standard -->
<!-- for log storage and management. -->
<!-- XES standard version: 1.0, OpenXES library version: 1.0RC7 -->
<log xes.version="1.0" xes.features="nested-attributes">
  <string key="concept:name" value="sh_fp200_2009.xes"/>
  <trace>
    <string key="concept:name" value="20009001269"/>
    <string key="applicant" value="019610230044"/>
    <string key="year" value="2009"/>
    <string key="dep" value="3"/>
    <boolean key="electronic" value="true"/>
    <string key="fp" value="200"/>
    <boolean key="rejected" value="false"/>
    <int key="numparcels" value="4"/>
    <float key="area" value="18.74"/>
    <float key="cc" value="0.0"/>
    <float key="cutting0" value="0.0"/>
    <float key="payment_granted0" value="3942.33"/>
    <float key="payment_actual0" value="3942.33"/>
    <event>
      <string key="concept:name" value="DokumentEingegangen"/>
      <string key="org:resource" value="Import"/>
      <date key="time:timestamp" value="2009-03-25T00:00:00+01:00"/>
      <string key="doctype" value="FP 200"/>
      <boolean key="successful" value="true"/>
    </event>
    <event>
      <string key="concept:name" value="DokumentGueltig"/>
      <string key="org:resource" value="0009"/>
      <date key="time:timestamp" value="2009-04-21T00:00:00+02:00"/>
      <string key="doctype" value="FP 200"/>
      <boolean key="successful" value="true"/>
    </event>
```

```
    <event>
      <string key="concept:name" value="Initialisieren"/>
      <string key="org:resource" value="Dummy"/>
      <date key="time:timestamp" value="2009-05-01T05:47:55+02:00"/>
      <string key="doctype" value="Flaechen"/>
      <boolean key="successful" value="true"/>
    </event>
    <event>
      <string key="concept:name" value="BearbeitungBeginnen"/>
      <string key="org:resource" value="Dummy"/>
      <date key="time:timestamp" value="2009-05-01T05:47:56+02:00"/>
      <string key="doctype" value="Flaechen"/>
      <boolean key="successful" value="true"/>
    </event>
    <!-- more events -->
  </trace>
  <trace>
    <string key="concept:name" value="20009004911"/>
    <string key="applicant" value="019540980018"/>
    <string key="year" value="2009"/>
    <string key="dep" value="6"/>
    <boolean key="electronic" value="true"/>
    <string key="fp" value="200"/>
    <boolean key="rejected" value="false"/>
    <int key="numparcels" value="42"/>
    <float key="area" value="105.2601"/>
    <float key="cc" value="15.0"/>
    <float key="amount" value="0.0"/>
    <float key="cutting0" value="0.0"/>
    <float key="payment_granted0" value="32811.27"/>
    <float key="payment_actual0" value="32811.27"/>
    <float key="cutting1" value="0.0"/>
    <float key="payment_granted1" value="38601.5"/>
    <float key="payment_actual1" value="5790.23"/>
    <!-- more events -->
  </trace>
<!-- more traces -->
</log>
```

B Appendix: Process models

Here, we show some example process models that resulted from the discovery algorithms we used. We see that without appropriate log preprocessesing, the raw results a hardly human readible. However, we think that a high level view on these models gives some intuition about what the algorithms do. We applied the algorithms to a part of the log, namely all cases from department 1 in *SH*, using their prefixes until the first payment, see chapter 6 for details.

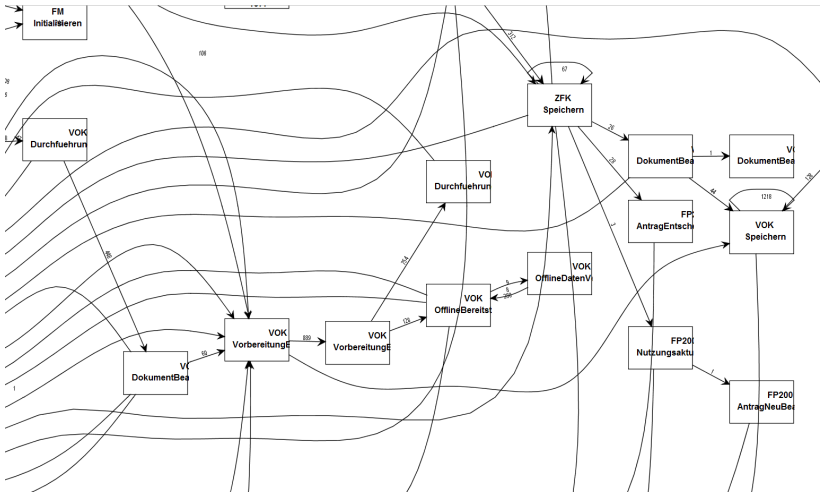


Figure B.1: Part of the dependency graph produced by the ProM Heuristic Miner plugin. The C-net semantics (splits, joins) are not included, since it would have obfuscated the picture even further.

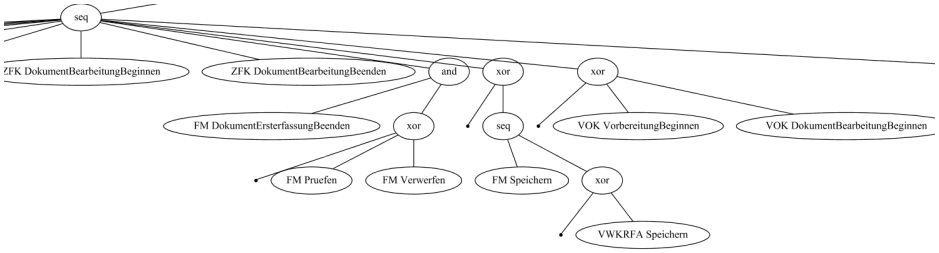


Figure B.2: Part of the process tree produced by the ProM Inductive Miner plugin. There are slight notational differences to our definition: *and* corresponds to $\wedge$, *seq* to $\rightarrow$, *xor* to $\times$ and an empty leaf node to τ.

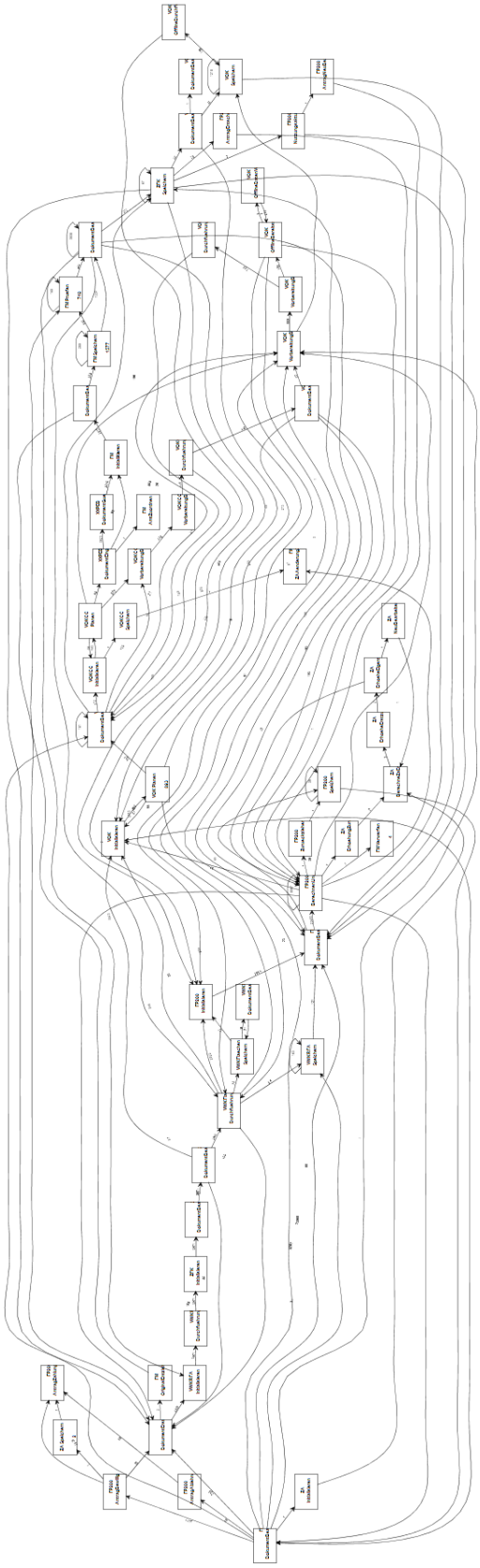


Figure B.3: Complete dependency graph produced by the ProM Heuristic Miner plugin with default settings. Note that this figure is meant to provide an idea about the model structure – the activity labels are of no interest for this purpose. When creating these models with ProM, there is a zoom function for navigating this graph.

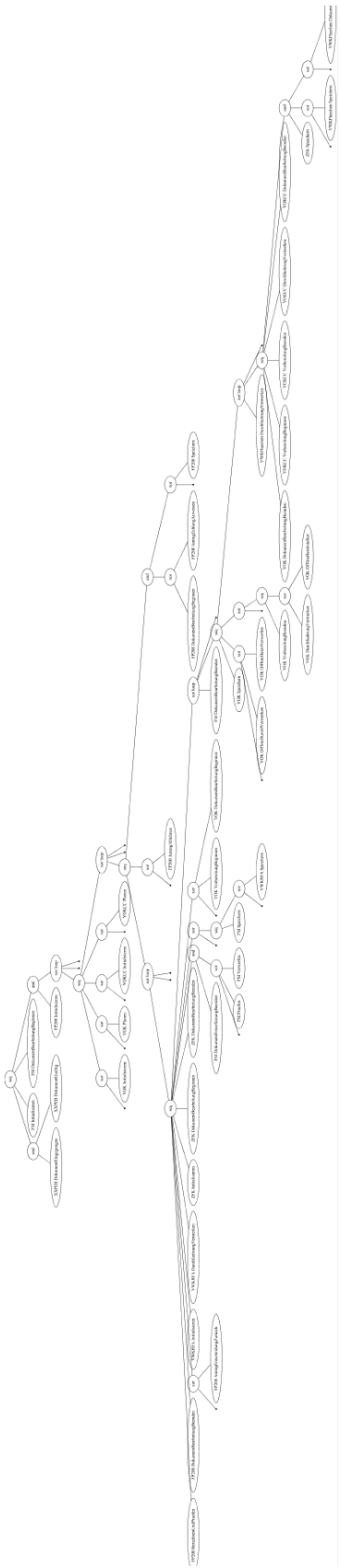


Figure B.4: Complete process tree produced by the ProM Inductive Miner plugin with default settings

C Appendix: Experimental results

C.1 Binary classification

The detailed experimental results for binary classification are shown in the following tables:

- tables C.1 and C.2 show the results for scenario 1
- tables C.3 and C.4 show the results for scenario 2
- tables C.5 and C.6 show the results for scenario 3

For both training / test splits, we report the number of positive and negative instances in the training set (columns Tr+ and Tr-), the AUC achieved in cross validation (column CV), the number of positive and negative instances in the respective test set (columns Te+ and Te-) and finally the AUC on the test set (column AUC). We report the results per department, the weighted average for all departments and the results training on all departments at once.

C.2 Outlier detection

The detailed experimental results in the outlier detection scenarios are shown in the following tables:

- tables C.7 and C.8 show the results for scenario 3
- tables C.9 and C.10 show the results for scenario 4

We report the number of positive and negative instances in the training set 2009-2012 (Tr+, Tr−) as well as the averaged F-score for the optimal value of γ. For the test set L^2_{te}, we report the number of positive and negative instances (Te+, Te−). To assess the performance of outlier detection, we report the number of true positives (TP), false positives (FP), true negatives (TN), false negatives (FN) and the derived values for recall (R), precision (Pr) and F-score (F1). We also show the size of the sample returned by the OC-SVM, where sample sizes that are more than $\pm 0.5\%$ off the target of 5% are coloured red. The highest F-scores and highest numbers of TP are printed in bold face, the profile that achieved the highest F1-score on L^2_{te}, i.e. in 2013, is underlined. Again, we compute the average performance over all departments, this time by microaveraging the numbers of TP, FP, TN and FN.

	L^1_{te}						L^2_{te}					
ST-101	**Tr+**	**Tr-**	**CV**	**Te+**	**Te-**	**AUC**	**Tr+**	**Tr-**	**CV**	**Te+**	**Te-**	**AUC**
case only	278	1941	.768	55	529	.783	333	2470	.775	91	594	.749
index-based 15			.755			.706			.742			.759
index-based 30			.738			.669			.737			.770
bag-of-activities			**.845**			.827			.844			.810
bag-of-activities+time			**.845**			.820			**.849**			**.811**
tree-based			.841			.808			.839			.804
ST-202												
case only	872	1018	.800	207	239	.802	1079	1257	.798	253	330	.729
index-based 15			.879			.865			.874			.622
index-based 30			.892			.883			.889			.660
bag-of-activities			.912			**.928**			.915			.717
bag-of-activities+time			**.920**			.923			.921			**.743**
tree-based			.917			.925			.920			**.743**
ST-268												
case only	211	1027	.796	55	266	.800	266	1293	.811	60	326	.799
index-based 15			.775			.750			.779			.793
index-based 30			.786			.752			.766			.829
bag-of-activities			.860			.824			.849			.853
bag-of-activities+time			**.866**			**.829**			.857			**.857**
tree-based			.861			.817			**.859**			.854
ST-303												
case only	1139	1147	.726	279	330	.692	1418	1477	.718	154	554	.711
index-based 15			.833			.805			.828			.705
index-based 30			.829			.834			.832			.699
bag-of-activities			.898			.882			.896			.730
bag-of-activities+time			**.907**			**.888**			**.906**			**.740**
tree-based			.895			.877			.891			.732
ST-357												
case only	359	1139	.784	63	272	.734	422	1411	.793	74	384	.788
index-based 15			.750			.697			.750			.723
index-based 30			.740			.754			.747			.737
bag-of-activities			**.849**			.809			**.850**			**.789**
bag-of-activities+time			.830			**.830**			.840			**.789**
tree-based			.825			.796			.824			**.789**
ST-363												
case only	782	2299	.841	187	585	.848	969	2884	.844	79	867	.707
index-based 15			.868			.851			.859			.764
index-based 30			.857			.869			.857			.760
bag-of-activities			**.901**			**.914**			**.904**			**.817**
bag-of-activities+time			.896			.908			.899			.813
tree-based			.889			.896			.892			.794
ST-370												
case only	1035	671	.784	245	186	.778	1280	857	.785	153	357	.783
index-based 15			.827			.843			.836			.773
index-based 30			.860			.874			.872			.788
bag-of-activities			.914			.927			.916			.822
bag-of-activities+time			**.916**			**.935**			**.921**			**.827**
tree-based			.895			.923			.905			.802
ST-avg												
case only	4676	9242	.788	1091	2407	.780	5767	11649	.790	864	3412	.743
index-based 15			.820			.796			.816			.733
index-based 30			.820			.810			.820			.745
bag-of-activities			.885			.879			.885			.789
bag-of-activities+time			**.886**			**.880**			**.888**			**.794**
tree-based			.877			.868			.879			.784
ST-all												
case only	4676	9242	.831	1091	2407	.827	5767	11649	.831	864	3412	.765
index-based 15			.859			.850			.861			.738
index-based 30			.858			.860			.861			.771
bag-of-activities			**.908**			**.909**			.909			.793
bag-of-activities+time			.907			.906			**.910**			**.805**
tree-based			.897			.897			.900			.774

Table C.1: Binary classification results for scenario 1 in *ST*

	L^1_{te}						L^2_{te}					
SH-1	**Tr+**	**Tr-**	**CV**	**Te+**	**Te-**	**AUC**	**Tr+**	**Tr-**	**CV**	**Te+**	**Te-**	**AUC**
case only	1679	11120	.754	412	2777	.769	2091	13897	.763	165	3700	.805
index-based 15			.748			.769			.762			.792
index-based 30			.749			.756			.748			.764
bag-of-activities			**.787**			**.794**			**.791**			**.844**
bag-of-activities+time			.780			.783			.787			.826
tree-based			.783			.780			.775			.835
SH-2												
case only	1921	4619	.779	454	1177	.790	2375	5796		268	1720	.826
index-based 15			.785			.797			.789			.822
index-based 30			.778			.802			.784			.843
bag-of-activities			**.813**			**.826**			**.814**			**.857**
bag-of-activities+time			.799			.818			.800			.837
tree-based			.793			.800			.795			.825
SH-3												
case only	2738	12621	.776	659	3239	.756	3397	15860	.770	560	4102	.808
index-based 15			.776			.750			.770			.789
index-based 30			.772			.761			.771			.801
bag-of-activities			**.800**			.774			**.796**			**.815**
bag-of-activities+time			.786			.772			.784			.796
tree-based			.788			**.775**			.781			.784
SH-6												
case only	2365	12955	.749	592	3277	.748	2957	16232	.751	181	4386	.791
index-based 15			.745			.758			.754			.769
index-based 30			.745			.755			.749			.744
bag-of-activities			**.791**			**.799**			**.794**			**.807**
bag-of-activities+time			.783			.794			.793			.780
tree-based			.775			.791			.785			.762
SH-avg												
case only	8703	41315	.762	2117	10470	.761	10820	51785	.764	1174	13908	.804
index-based 15			.761			.763			.766			.788
index-based 30			.759			.763			.760			.780
bag-of-activities			**.796**			**.793**			**.796**			**.826**
bag-of-activities+time			.785			.788			.790			.804
tree-based			.783			.784			.783			.796
SH-all												
case only	8703	41315	.776	2117	10470	.772	10820	51785	.776	1174	13908	.823
index-based 15			.778			.779			.781			.819
index-based 30			.776			.756			.775			.811
bag-of-activities			**.806**			**.803**			.807			**.842**
bag-of-activities+time			.798			.797			.799			.816
tree-based			.779			.792			**.816**			.805

Table C.2: Binary classification results for scenario 1 in *SH*

	L^1_{te}						L^2_{te}					
ST-101	**Tr+**	**Tr-**	**CV**	**Te+**	**Te-**	**AUC**	**Tr+**	**Tr-**	**CV**	**Te+**	**Te-**	**AUC**
case only	101	156	.647	23	28	.901	124	184	.712	47	40	.633
index-based 15			.626			.807			.690			.597
index-based 30			.656			.832			.703			.536
bag-of-activities			.732			**.907**			.787			.647
bag-of-activities+time			.723			.887			.771			.634
tree-based			**.740**			.887			**.801**			**.655**
ST-202												
case only	177	674	.735	60	145	.804	237	819	.765	141	107	.670
index-based 15			.749			.748			.748			.581
index-based 30			.736			.687			.735			.604
bag-of-activities			.795			.806			.802			.751
bag-of-activities+time			**.823**			**.822**			**.838**			**.761**
tree-based			.816			.775			.812			.731
ST-268												
case only	119	88	.697	28	24	.835	147	112	.727	44	16	.528
index-based 15			.696			.695			.703			.516
index-based 30			.658			.760			.697			.530
bag-of-activities			.750			.836			.796			.518
bag-of-activities+time			.777			**.878**			.803			.511
tree-based			**.862**			.876			**.868**			**.612**
ST-303												
case only	558	573	.647	130	147	.662	688	720	.656	18	129	.633
index-based 15			.751			.732			.747			.582
index-based 30			.724			.777			.737			.559
bag-of-activities			.825			.851			.834			.601
bag-of-activities+time			.837			.874			.852			.643
tree-based			**.869**			**.901**			**.874**			**.736**
ST-357												
case only	123	231	.728	13	48	.591	136	279	.691	9	65	.517
index-based 15			.726			.650			.734			.549
index-based 30			.735			.651			.713			.432
bag-of-activities			.803			.756			.812			.654
bag-of-activities+time			.821			.764			.828			.736
tree-based			**.870**			**.785**			**.868**			**.797**
ST-363												
case only	582	176	.824	136	44	.836	718	220	.827	35	40	.611
index-based 15			.810			.866			.817			.534
index-based 30			.767			.747			.800			.566
bag-of-activities			.862			**.923**			.883			.680
bag-of-activities+time			.880			.906			**.888**			**.729**
tree-based			**.886**			.867			.886			.704
ST-370												
case only	429	602	.829	109	133	.809	538	735	.823	45	105	.737
index-based 15			.814			.814			.806			.696
index-based 30			.785			.785			.797			.640
bag-of-activities			**.885**			**.875**			**.890**			.767
bag-of-activities+time			.862			.865			.867			**.774**
tree-based			.856			.862			.854			.730
ST-avg												
case only	2089	2500	.742	499	569	.768	2588	3069	.751	339	502	.672
index-based 15			.763			.773			.766			.592
index-based 30			.741			.751			.755			.572
bag-of-activities			.829			.856			.843			.685
bag-of-activities+time			.837			**.862**			.850			.707
tree-based			**.852**			.854			**.855**			**.719**
ST-all												
case only	2089	2500	.795	499	569	.807	2588	3069	.801	339	502	.650
index-based 15			.812			.818			.812			.704
index-based 30			.792			.799			.800			.695
bag-of-activities			.872			**.880**			.875			**.752**
bag-of-activities+time			.876			.884			**.880**			.744
tree-based			**.880**			.875			.876			.722

Table C.3: Binary classification results for scenario 2 in *ST*

	L^1_{te}						L^2_{te}					
SH-1	**Tr+**	**Tr-**	**CV**	**Te+**	**Te-**	**AUC**	**Tr+**	**Tr-**	**CV**	**Te+**	**Te-**	**AUC**
case only	535	1053	.730	127	268	.712	662	1321	.730	71	56	.732
index-based 15			.721			.693			.716			.712
index-based 30			.710			.686			.712			.579
bag-of-activities			.831			.787			.831			.752
bag-of-activities+time			.857			.840			.862			.793
tree-based			**.878**			**.866**			**.876**			**.836**
SH-2												
case only	807	1083	.810	172	273	.770	979	1356	.795	103	156	.815
index-based 15			.796			.773			.789			.810
index-based 30			.783			.763			.776			.809
bag-of-activities			**.847**			.802			.836			.834
bag-of-activities+time			.839			.825			**.839**			.858
tree-based			.834			**.828**			.833			**.867**
SH-3												
case only	1144	1493	.704	251	378	.675	1395	1871	.704	223	323	.746
index-based 15			.718			.670			.701			.725
index-based 30			.704			.697			.716			.711
bag-of-activities			.834			.820			.829			.722
bag-of-activities+time			.830			**.852**			**.844**			.804
tree-based			**.837**			.849			.840			**.840**
SH-6												
case only	716	1302	.727	182	321	.706	898	1623	.724	122	29	.574
index-based 15			.749			.703			.740			.540
index-based 30			.738			.730			.734			.549
bag-of-activities			.827			.833			.833			.688
bag-of-activities+time			.862			.863			.866			**.697**
tree-based			**.873**			**.866**			**.872**			.627
SH-avg												
case only	3202	4931	.739	732	1240	.712	3934	6171	.735	519	564	.737
index-based 15			.744			.706			.734			.718
index-based 30			.732			.718			.734			.696
bag-of-activities			.835			.813			.832			.748
bag-of-activities+time			.845			.846			.852			.801
tree-based			**.853**			**.852**			**.853**			**.816**
SH-all												
case only	3202	4931	.747	732	1240	.720	3934	6171	.746	519	564	.760
index-based 15			.760			.730			.752			.749
index-based 30			.744			.730			.748			.729
bag-of-activities			.840			.818			.839			.779
bag-of-activities+time			**.859**			.854			.856			**.822**
tree-based			.858			**.856**			**.861**			.796

Table C.4: Binary classification results for scenario 2 in *SH*

	L^1_{te}						L^2_{te}					
ST-101	**Tr+**	**Tr-**	**CV**	**Te+**	**Te-**	**AUC**	**Tr+**	**Tr-**	**CV**	**Te+**	**Te-**	**AUC**
case only	3	2216	**.824**	2	582	.490	5	2798	.680	6	679	.559
index-based 15			.794			.660			.721			.769
index-based 30			.728			.793			.681			.647
bag-of-activities			.801			.447			.612			.793
bag-of-activities+time			.560			.872			.602			**.816**
tree-based			.742			**.922**			**.723**			.685
ST-202												
case only	21	1869	.695	5	441	**.819**	26	2310	.746	53	530	**.761**
index-based 15			.781			.596			.714			.746
index-based 30			.676			.573			.641			.760
bag-of-activities			.740			.595			.709			.702
bag-of-activities+time			.768			.663			**.807**			.695
tree-based			**.808**			.678			.794			.686
ST-268												
case only	32	1206	.844	4	317	.946	36	1523	.793	2	384	**.956**
index-based 15			.788			.936			.803			.734
index-based 30			.778			.936			.773			.858
bag-of-activities			.852			**.990**			.853			.942
bag-of-activities+time			.855			.989			**.862**			.803
tree-based			**.861**			.966			**.862**			.814
ST-303												
case only	18	2268	.788	5	604	.602	23	2872	.691	3	705	.412
index-based 15			.748			**.835**			.730			**.778**
index-based 30			.734			.635			.754			.579
bag-of-activities			.819			.654			.790			.384
bag-of-activities+time			.798			.715			.785			.521
tree-based			**.851**			.634			**.833**			.586
ST-357												
case only	8	1490	.607	1	334	.436	9	1824	.480	1	457	**.991**
index-based 15			.713			.823			.741			.965
index-based 30			**.835**			.464			.652			.888
bag-of-activities			.774			**.840**			.737			.933
bag-of-activities+time			.756			.751			.737			.790
tree-based			.797			.756			**.803**			.781
ST-363												
case only	324	2757	.850	86	686	.853	410	3443	.854	15	931	.879
index-based 15			.913			.901			.906			.907
index-based 30			.913			.907			.908			**.910**
bag-of-activities			**.936**			**.949**			**.942**			.885
bag-of-activities+time			**.936**			.945			.938			.907
tree-based			.931			.939			.934			.830
ST-370												
case only	43	1663	.818	13	418	.781	56	2081	.810	4	506	**.863**
index-based 15			.805			.834			.827			.633
index-based 30			.826			.760			.822			.659
bag-of-activities			.865			**.850**			**.863**			.815
bag-of-activities+time			**.873**			.835			.851			.738
tree-based			.855			.786			.852			.768
ST-avg												
case only	449	13469	.784	116	3382	.704	565	16851	.734	84	4192	.751
index-based 15			.803			.798			.785			**.799**
index-based 30			.791			.740			.760			.756
bag-of-activities			**.835**			.750			.793			.764
bag-of-activities+time			.796			**.829**			.802			.758
tree-based			.841			.816			**.834**			.733
ST-all												
case only	449	13469	.892	116	3382	.888	565	16851	.886	84	4192	.791
index-based 15			.903			.902			.908			.737
index-based 30			.903			.907			.905			.777
bag-of-activities			**.939**			.934			**.934**			.749
bag-of-activities+time			.936			**.936**			.932			**.866**
tree-based			.927			.924			.930			.808

Table C.5: Binary classification results for scenario 3 in *ST*

	L^1_{te}						L^2_{te}					
SH-1	**Tr+**	**Tr-**	**CV**	**Te+**	**Te-**	**AUC**	**Tr+**	**Tr-**	**CV**	**Te+**	**Te-**	**AUC**
case only	42	12757	.700	10	3179	.636	52	15936	.686	21	3844	.691
index-based 15			.710			.593			.737			.720
index-based 30			.716			.616			**.777**			**.802**
bag-of-activities			**.780**			**.701**			.755			.784
bag-of-activities+time			.694			.686			.730			.752
tree-based			.758			.657			.768			.733
SH-2												
case only	13	6527	.497	5	1626	.743	18	8153	.512	4	1984	.579
index-based 15			.595			.744			.585			.733
index-based 30			.536			.562			.550			.744
bag-of-activities			.503			**.841**			**.720**			.507
bag-of-activities+time			.583			.717			.612			**.757**
tree-based			**.632**			.827			.654			.605
SH-3												
case only	174	15185	.710	45	3853	.710	219	19038	.709	3	4659	.634
index-based 15			.826			.878			.839			**.773**
index-based 30			.862			.883			.873			.642
bag-of-activities			**.900**			.910			.900			.686
bag-of-activities+time			.891			.915			.897			.611
tree-based			.892			**.934**			**.908**			.493
SH-6												
case only	213	15107	.768	46	3823	.725	259	18930	.756	15	4552	.849
index-based 15			.825			.783			.812			.770
index-based 30			.809			.798			.811			.821
bag-of-activities			.873			.818			**.870**			**.879**
bag-of-activities+time			.868			**.849**			.851			.860
tree-based			**.888**			.817			.874			.771
SH-avg												
case only	442	49576	.697	106	12481	.700	548	62057	.692	43	15039	.706
index-based 15			.766			.759			.772			**.753**
index-based 30			.766			.748			.787			.751
bag-of-activities			.809			**.820**			**.830**			.746
bag-of-activities+time			.793			.811			.803			.742
tree-based			**.822**			.814			.829			.653
SH-all												
case only	442	49576	.742	106	12481	.765	548	62057	.756	43	15039	.765
index-based 15			.836			.828			.825			.792
index-based 30			.819			.837			.828			.769
bag-of-activities			.873			.852			.874			.763
bag-of-activities+time			.867			.869			.873			**.793**
tree-based			**.875**			**.862**			**.879**			.730

Table C.6: Binary classification results for scenario 3 in *SH*

	L^2_{te}											
ST-101	**Tr+**	**Tr-**	**Tr-F1**	**Te+**	**Te-**	**TP**	**FP**	**TN**	**FN**	**R**	**Pr**	**F1**	**%**
case only	5	2798	**.028**	6	679	**2**	33	646	4	.333	.057	**.098**	5.109
bag-of-activities	5	2798	.000	6	679	1	33	646	5	.167	.029	.050	4.964
bag-of-activities+time	5	2798	.013	6	679	0	34	645	6	.000	.000	.000	4.964
time w/o count	5	2798	.014	6	679	1	33	646	5	.167	.029	.050	4.964
tree-based	5	2798	.014	6	679	1	36	643	5	.167	.027	.047	5.401
tree-based w/o count	5	2798	.014	6	679	1	35	644	5	.167	.028	.048	5.255
ST-202													
case only	26	2310	.085	53	530	**7**	23	507	46	.132	.233	**.169**	5.146
bag-of-activities	26	2310	.070	53	530	5	25	505	48	.094	.167	.120	5.146
bag-of-activities+time	26	2310	.070	53	530	5	23	507	48	.094	.179	.123	4.803
time w/o count	26	2310	.082	53	530	4	26	504	49	.075	.133	.096	5.146
tree-based	26	2310	.071	53	530	4	26	504	49	.075	.133	.096	5.146
tree-based w/o count	26	2310	**.099**	53	530	4	25	505	49	.075	.138	.098	4.974
ST-268													
case only	36	1523	**.161**	2	384	**1**	17	367	1	.500	.056	**.100**	4.663
bag-of-activities	36	1523	.097	2	384	**1**	23	361	1	.500	.042	.077	6.218
bag-of-activities+time	36	1523	.091	2	384	0	19	365	2	.000	.000	.000	4.922
time w/o count	36	1523	.089	2	384	0	21	363	2	.000	.000	.000	5.440
tree-based	36	1523	.103	2	384	0	25	359	2	.000	.000	.000	6.477
tree-based w/o count	36	1523	.085	2	384	**1**	67	317	1	.500	.015	.029	17.617
ST-303													
case only	23	2872	**.085**	3	705	0	35	670	3	0	0	0	4.944
bag-of-activities	23	2872	.058	3	705	0	39	666	3	0	0	0	5.508
bag-of-activities+time	23	2872	.036	3	705	0	36	669	3	0	0	0	5.085
time w/o count	23	2872	.036	3	705	0	36	669	3	0	0	0	5.085
tree-based	23	2872	.047	3	705	0	35	670	3	0	0	0	4.944
tree-based w/o count	23	2872	.048	3	705	0	32	673	3	0	0	0	4.520
ST-357													
case only	9	1824	.020	1	457	0	23	434	1	0	0	0	5.022
bag-of-activities	9	1824	**.040**	1	457	0	23	434	1	0	0	0	5.022
bag-of-activities+time	9	1824	.039	1	457	0	22	435	1	0	0	0	4.803
time w/o count	9	1824	.039	1	457	0	22	435	1	0	0	0	4.803
tree-based	9	1824	.020	1	457	0	23	434	1	0	0	0	5.022
tree-based w/o count	9	1824	.020	1	457	0	23	434	1	0	0	0	5.022
ST-363													
case only	410	3443	.093	15	931	2	46	885	13	.133	.042	.063	5.074
bag-of-activities	410	3443	.091	15	931	3	47	884	12	.200	.060	**.092**	5.285
bag-of-activities+time	410	3443	.132	15	931	3	126	805	12	.200	.023	.042	13.636
time w/o count	410	3443	.137	15	931	**6**	173	758	9	.400	.034	.062	18.922
tree-based	410	3443	.150	15	931	**6**	206	725	9	.400	.028	.053	22.410
tree-based w/o count	410	3443	**.165**	15	931	**6**	219	712	9	.400	.027	.050	23.784
ST-370													
case only	56	2081	.098	4	506	2	23	483	2	.500	.080	**.138**	4.902
bag-of-activities	56	2081	**.141**	4	506	0	27	479	4	.000	.000	.000	5.294
bag-of-activities+time	56	2081	.082	4	506	**3**	38	468	1	.750	.073	.133	8.039
time w/o count	56	2081	.079	4	506	2	36	470	2	.500	.053	.095	7.451
tree-based	56	2081	.080	4	506	1	31	475	3	.250	.031	.056	6.275
tree-based w/o count	56	2081	.086	4	506	1	24	482	3	.250	.040	.069	4.902
ST-avg													
case only	565	16851	.079	84	4192	**14**	200	3992	70	.167	.065	**.094**	5.005
bag-of-activities	565	16851	.070	84	4192	10	217	3975	74	.119	.044	.064	5.309
bag-of-activities+time	565	16851	.069	84	4192	11	298	3894	73	.131	.036	.056	7.226
time w/o count	565	16851	.072	84	4192	13	347	3845	71	.155	.036	.059	8.419
tree-based	565	16851	.074	84	4192	12	382	3810	72	.143	.030	.050	9.214
tree-based w/o count	565	16851	**.080**	84	4192	13	425	3767	71	.155	.030	.050	1.243
ST-all													
case only	565	16851	.075	84	4192	**11**	198	3994	73	.131	.053	**.075**	4.888
bag-of-activities	565	16851	.068	84	4192	7	206	3986	77	.083	.033	.047	4.981
bag-of-activities+time	565	16851	.074	84	4192	7	207	3985	77	.083	.033	.047	5.005
time w/o count	565	16851	**.078**	84	4192	9	207	3985	75	.107	.042	.060	5.051
tree-based	565	16851	**.078**	84	4192	9	202	3990	75	.107	.043	.061	4.935
tree-based w/o count	565	16851	.076	84	4192	10	201	3991	74	.119	.047	.068	4.935

Table C.7: Outlier detection results for scenario 3 in *ST*

						L^2_{te}						
SH-1	**Tr+**	**Tr-**	**Tr-F1**	**Te+**	**Te-**	**TP**	**FP**	**TN**	**FN**	**R**	**Pr**	**F1**	**%**
case only	52	15936	**.035**	21	3844	**8**	185	3659	13	.381	.041	**.075**	4.99
bag-of-activities			.019			6	187	3657	15	.286	.031	.056	4.99
bag-of-activities+time			.024			7	189	3655	14	.333	.036	.065	5.07
time w/o count			.026			7	188	3656	14	.333	.036	.065	5.05
tree-based			.029			6	197	3647	15	.286	.030	.054	5.25
tree-based w/o count			.028			5	192	3652	16	.238	.025	.046	5.10
SH-2													
case only	18	8153	**.028**	4	1984	1	97	1887	3	.250	.010	.020	4.93
bag-of-activities			**.028**			1	100	1884	3	.250	.010	.019	5.08
bag-of-activities+time			.019			**2**	96	1888	2	.500	.020	.039	4.93
time w/o count			.019			**2**	106	1878	2	.500	.019	.036	5.43
tree-based			.018			**2**	113	1871	2	.500	.017	.034	5.78
tree-based w/o count			.014			**2**	95	1889	2	.500	.021	**.040**	4.88
SH-3													
case only	219	19038	.034	3	4659	0	233	4426	3	.000	.000	.000	5.00
bag-of-activities			.046			**1**	230	4429	2	.333	.004	**.009**	4.95
bag-of-activities+time			.042			**1**	243	4416	2	.333	.004	.008	5.23
time w/o count			.032			**1**	232	4427	2	.333	.004	.008	5.00
tree-based			.045			**1**	240	4419	2	.333	.004	.008	5.17
tree-based w/o count			.041			**1**	233	4426	2	.333	.004	.008	5.02
SH-6													
case only	259	18930	.076	15	4552	3	225	4327	12	.200	.013	.025	4.99
bag-of-activities			**.092**			3	225	4327	12	.200	.013	.025	4.99
bag-of-activities+time			.076			**5**	205	4347	10	.333	.024	**.044**	4.60
time w/o count			.069			**5**	223	4329	10	.333	.022	.041	4.99
tree-based			.071			4	232	4320	11	.267	.017	.032	5.17
tree-based w/o count			.062			4	254	4298	11	.267	.016	.029	5.65
SH-avg													
case only	548	62057	.046	43	15039	12	740	14299	31	.279	.016	.030	4.986
bag-of-activities			.051			11	742	14297	32	.256	.015	.028	4.993
bag-of-activities+time			.045			**15**	733	14306	28	.349	.020	**.038**	4.960
time w/o count			.040			**15**	749	14290	28	.349	.020	.037	5.066
tree-based			.045			13	782	14257	30	.302	.016	.031	5.271
tree-based w/o count			.041			12	774	14265	31	.279	.015	.029	5.212
SH-all													
case only	548	62057	.044	43	15039	13	737	14302	30	.302	.017	.033	4.97
bag-of-activities			**.046**			11	747	14292	32	.256	.015	.027	5.03
bag-of-activities+time			.044			11	757	14282	32	.256	.014	.027	5.09
time w/o count			.040			**15**	743	14296	28	.349	.020	**.037**	5.03
tree-based			.044			11	779	14260	32	.256	.014	.026	5.24
tree-based w/o count			.036			13	746	14293	30	.302	.017	.032	5.03

Table C.8: Outlier detection results for scenario 3 in *SH*

						L^2_{te}							
ST-101	**Tr+**	**Tr-**	**Tr-F1**	**Te+**	**Te-**	**TP**	**FP**	**TN**	**FN**	**R**	**Pr**	**F1**	**%**
case only	34	2769	**.093**	8	677	**2**	33	644	6	.250	.057	**.093**	5.109
bag-of-activities	34	2769	**.093**	8	677	0	35	642	8	.000	.000	.000	5.109
bag-of-activities+time	34	2769	.065	8	677	**2**	65	612	6	.250	.030	.053	9.781
time w/o count	34	2769	.070	8	677	0	34	643	8	.000	.000	.000	4.964
tree-based	34	2769	.057	8	677	**2**	35	642	6	.250	.054	.089	5.401
tree-based w/o count	34	2769	.053	8	677	1	35	642	7	.125	.028	.045	5.255
ST-202													
case only	42	2294	.052	12	571	**1**	28	543	11	.083	.034	**.049**	4.974
bag-of-activities	42	2294	.088	12	571	0	30	541	12	.000	.000	.000	5.146
bag-of-activities+time	42	2294	.083	12	571	0	35	536	12	.000	.000	.000	6.003
time w/o count	42	2294	.086	12	571	0	30	541	12	.000	.000	.000	5.146
tree-based	42	2294	**.107**	12	571	0	37	534	12	.000	.000	.000	6.346
tree-based w/o count	42	2294	.076	12	571	0	29	542	12	.000	.000	.000	4.974
ST-268													
case only	9	1550	.047	0	386	0	18	368	0	.000	.000	.000	4.663
bag-of-activities	9	1550	**.095**	0	386	0	20	366	0	.000	.000	.000	5.181
bag-of-activities+time	9	1550	.048	0	386	0	19	367	0	.000	.000	.000	4.922
time w/o count	9	1550	.046	0	386	0	21	365	0	.000	.000	.000	5.440
tree-based	9	1550	.046	0	386	0	20	366	0	.000	.000	.000	5.181
tree-based w/o count	9	1550	.022	0	386	0	20	366	0	.000	.000	.000	5.181
ST-303													
case only	53	2842	.041	11	697	3	32	665	8	.273	.086	.130	4.944
bag-of-activities	53	2842	**.072**	11	697	2	35	662	9	.182	.054	.083	5.226
bag-of-activities+time	53	2842	.061	11	697	4	32	665	7	.364	.111	**.170**	5.085
time w/o count	53	2842	.071	11	697	**7**	144	553	4	.636	.046	.086	21.328
tree-based	53	2842	.065	11	697	2	37	660	9	.182	.051	.080	5.508
tree-based w/o count	53	2842	.069	11	697	2	38	659	9	.182	.050	.078	5.650
ST-357													
case only	22	1811	.053	0	458	0	23	435	0	.000	.000	.000	5.022
bag-of-activities	22	1811	.085	0	458	0	26	432	0	.000	.000	.000	5.677
bag-of-activities+time	22	1811	.103	0	458	0	26	432	0	.000	.000	.000	5.677
time w/o count	22	1811	.088	0	458	0	24	434	0	.000	.000	.000	5.240
tree-based	22	1811	.087	0	458	0	22	436	0	.000	.000	.000	4.803
tree-based w/o count	22	1811	**.104**	0	458	0	24	434	0	.000	.000	.000	5.240
ST-363													
case only	55	3798	.081	6	940	0	48	892	6	.000	.000	.000	5.074
bag-of-activities	55	3798	.065	6	940	0	46	894	6	.000	.000	.000	4.863
bag-of-activities+time	55	3798	.072	6	940	1	89	851	5	.167	.011	.021	9.514
time w/o count	55	3798	**.082**	6	940	0	86	854	6	.000	.000	.000	9.091
tree-based	55	3798	.070	6	940	1	51	889	5	.167	.019	.034	5.497
tree-based w/o count	55	3798	.071	6	940	**3**	128	812	3	.500	.023	**.044**	13.848
ST-370													
case only	76	2061	**.164**	9	501	1	25	476	8	.111	.038	.057	5.098
bag-of-activities	76	2061	.132	9	501	2	23	478	7	.222	.080	**.118**	4.902
bag-of-activities+time	76	2061	.132	9	501	1	24	477	8	.111	.040	.059	4.902
time w/o count	76	2061	.118	9	501	2	25	476	7	.222	.074	.111	5.294
tree-based	76	2061	.124	9	501	**5**	129	372	4	.556	.037	.070	26.275
tree-based w/o count	76	2061	.125	9	501	3	93	408	6	.333	.031	.057	18.824
ST-avg													
case only	291	17125	.076	46	4230	7	207	4023	39	.152	.033	**.054**	5.005
bag-of-activities	291	17125	**.087**	46	4230	4	215	4015	42	.087	.018	.030	5.122
bag-of-activities+time	291	17125	.079	46	4230	8	290	3940	38	.174	.027	.047	6.969
time w/o count	291	17125	.081	46	4230	9	364	3866	37	.196	.024	.043	8.723
tree-based	291	17125	.078	46	4230	**10**	331	3899	36	.217	.029	.052	7.975
tree-based w/o count	291	17125	.074	46	4230	9	367	3863	37	.196	.024	.043	8.793
ST-all													
case only	291	17125	.062	46	4230	6	206	4024	40	.130	.028	**.047**	4.958
bag-of-activities	291	17125	**.080**	46	4230	5	208	4022	41	.109	.023	.039	4.981
bag-of-activities+time	291	17125	.064	46	4230	5	209	4021	41	.109	.023	.038	5.005
time w/o count	291	17125	.069	46	4230	4	267	3963	42	.087	.015	.025	6.338
tree-based	291	17125	.063	46	4230	**7**	253	3977	39	.152	.027	.046	6.080
tree-based w/o count	291	17125	.071	46	4230	5	230	4000	41	.109	.021	.036	5.496

Table C.9: Outlier detection results for scenario 4 in *ST*

						L^2_{te}							
SH-1	**Tr+**	**Tr-**	**Tr-F1**	**Te+**	**Te-**	**TP**	**FP**	**TN**	**FN**	**R**	**Pr**	**F1**	**%**
case only	139	15849	.019	32	3833	2	191	3642	30	.063	.010	.018	4.99
bag-of-activities			.032			7	186	3647	25	.219	.036	.062	4.99
bag-of-activities+time			.053			**9**	190	3643	23	.281	.045	**.078**	5.15
time w/o count			.054			8	206	3627	24	.250	.037	.065	5.54
tree-based			.056			6	197	3636	26	.188	.030	.051	5.25
tree-based w/o count			**.057**			7	190	3643	25	.219	.036	.061	5.10
SH-2													
case only	92	8079	.044	13	1975	0	98	1877	13	.000	.000	.000	4.93
bag-of-activities			.044			0	98	1877	13	.000	.000	.000	4.93
bag-of-activities+time			**.046**			**3**	95	1880	10	.231	.031	**.054**	4.93
time w/o count			.043			**3**	105	1870	10	.231	.028	.050	5.43
tree-based			.044			**3**	138	1837	10	.231	.021	.039	7.09
tree-based w/o count			.044			2	95	1880	11	.154	.021	.036	4.88
SH-3													
case only	139	19118	.024	18	4644	3	477	4167	15	.167	.006	.012	1.30
bag-of-activities			.036			4	227	4417	14	.222	.017	.032	4.95
bag-of-activities+time			**.058**			4	230	4414	14	.222	.017	.032	5.02
time w/o count			.054			4	230	4414	14	.222	.017	.032	5.02
tree-based			**.058**			4	230	4414	14	.222	.017	.032	5.02
tree-based w/o count			.055			**5**	228	4416	13	.278	.021	**.040**	5.00
SH-6													
case only	352	18837	.047	32	4535	2	226	4309	30	.063	.009	.015	4.99
bag-of-activities			.078			3	225	4310	29	.094	.013	.023	4.99
bag-of-activities+time			**.092**			4	206	4329	28	.125	.019	.033	4.60
time w/o count			.085			4	236	4299	28	.125	.017	.029	5.26
tree-based			.082			**5**	231	4304	27	.156	.021	.037	5.17
tree-based w/o count			.079			**5**	221	4314	27	.156	.022	**.039**	4.95
SH-avg													
case only	722	61883	.033	95	14987	7	992	13995	88	.074	.007	.013	6.62
bag-of-activities			.050			14	736	14251	81	.147	.019	.033	4.97
bag-of-activities+time			**.065**			**20**	721	14266	75	.211	.027	**.048**	4.91
time w/o count			.062			19	777	14210	76	.200	.024	.043	5.28
tree-based			.063			18	796	14191	77	.189	.022	.04	5.40
tree-based w/o count			.061			19	734	14253	76	.200	.025	.045	4.99
SH-all													
case only	722	61883	.032	95	14987	8	744	14243	87	.084	.011	.019	4.99
bag-of-activities			.052			16	742	14245	79	.168	.021	.038	5.03
bag-of-activities+time			.059			19	749	14238	76	.200	.025	.044	5.09
time w/o count			.056			**20**	744	14243	75	.211	.026	**.047**	5.07
tree-based			**.067**			19	749	14238	76	.200	.025	.044	5.09
tree-based w/o count			.060			**20**	744	14243	75	.211	.026	**.047**	5.07

Table C.10: Outlier detection results for scenario 4 in *SH*

Selbständigkeitserklärung

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig verfasst und noch nicht für andere Prüfungen eingereicht habe. Sämtliche Quellen einschließlich Internetquellen, die unverändert oder abgewandelt wiedergegeben werden, insbesondere Quellen für Texte, Grafiken, Tabellen und Bilder, sind als solche kenntlich gemacht. Mir ist bekannt, dass bei Verstößen gegen diese Grundsätze ein Verfahren wegen Täuschungsversuchs bzw. Täuschung eingeleitet wird.

Berlin, den 26. September 2016 .....