UNIVERSITY OF TARTU
Institute of Computer Science
Computer Science Curriculum

Joonas Puura

Advanced Methods in Business Process Deviance Mining

Master's Thesis (30 ECTS)

Supervisors: Fabrizio Maria Maggi, PhD
Chiara Di Francescomarino, PhD
Chiara Ghidini, PhD

Tartu 2019

Advanced Methods in Business Process Deviance Mining

Abstract:

Business process deviance refers to the phenomenon whereby a subset of the executions of a business process deviate, in a negative or positive way, with respect to its expected or desirable outcomes. Deviant executions of a business process include those that violate compliance rules, or executions that undershoot or exceed performance targets. Deviance mining is concerned with uncovering the reasons for deviant executions by analyzing business process event logs.

In this thesis, the problem of explaining deviations in business processes is first investigated by using features based on sequential and declarative patterns, and a combination of them. The explanations are further improved by leveraging the data payload of events and traces in event logs through features based on pure data attribute values and data-aware declare constraints. The explanations characterizing the deviances are then extracted by direct and indirect methods for rule induction. Using synthetic and real-life logs from multiple domains, a range of feature types and different forms of decision rules are evaluated in terms of their ability to accurately discriminate between non-deviant and deviant executions of a process as well as in terms of the final outcome returned to the users.

Keywords:

Deviance mining, business process, Declare, classification

CERCS: P170: Computer science, numerical analysis, systems, control

Edasijõudnud meetodid äriprotsesside hälbe kaevandamiseks

Lühikokkuvõte:

Äriprotsessi hälve on nähtus, kus alamhulk äriprotsessi täitmistest erinevad soovitud või ettenähtud tulemusest, kas positiivses või negatiivses mõttes. Äriprotsesside hälbega täitmised sisaldavad endas täitmisi, mis ei vasta ettekirjutatud reeglitele või täitmised, mis on jäävad alla või ületavad tulemuslikkuse eesmärke. Hälbekaevandus tegeleb hälbe põhjuste otsimisega, analüüsides selleks äriprotsesside sündmuste logisid.

Antud töös lähenetakse protsessihälvete põhjuste otsimise ülesandele, esmalt kasutades järjestikkudel põhinevaid või deklaratiivseid mustreid ning nende kombinatsiooni. Hälbekaevandusest saadud põhjendusi saab parendada, kasutades sündmustes ja sündmusjälgede atribuutides sisalduvaid andmelaste. Andmelastidest konstrueeritakse uued tunnused nii otsekoheselt atribuute ekstraheerides ja agregeerides kui ka andmeteadlike deklaratiivseid piiranguid kasutades. Hälbeid iseloomustavad põhjendused ekstraheeritakse kasutades kaudset ja otsest meetodit reeglite induktsiooniks. Kasutades sünteetilisi ja reaalseid logisid, hinnatakse erinevaid tunnuseid ja tulemuseks saadud otsustusreegleid nii nende võimekuses täpselt eristada hälbega ja hälbeta protsesside täitmiseid kui ka kasutajatele antud lõpptulemustes.

Võtmesõnad:
hälbekaevandus, äriprotsessid, Declare, liigitamine

CERCS: P170: Arvutiteadus, arvutusmeetodid, süsteemid, juhtimine (automaatjuhtimis-teooria)

Contents

1 Introduction

The increasing adoption of ERP (Enterprise Resource Planning) and management applications able to track information about process executions in the so called execution *traces*, has opened up the possibility of extracting knowledge from traces, collected in *event logs*. Different techniques have been developed in the context of *process mining* to *discover* models from event logs, to *check* conformance between an event log and a process model, or to *enhance* existing process models starting from event logs. Among the different types of knowledge that can be extracted from event logs, a crucial role is played by the explanation of *deviant* cases, i.e., business process executions that deviate in a positive or negative way from the expected outcome. Indeed, discovering why some executions take more (less) time than others, or what characterizes the cases that end up with a faulty (or particularly good) outcome could be very useful for business analysts in order to understand what can be improved to reduce negative deviances and spread the positive ones.

Business process deviance mining is a branch of process mining which aims at analyzing event logs in order to discover and characterize business process deviances. Deviant executions of a business process include those cases that do not reach or achieve targets (e.g., very slow and very fast cases), or those that violate some constraints.

The input of deviance mining approaches is an event log, in which each trace is labeled as *deviant* or *non-deviant*. The purpose is to discover a *characterization* of the deviant traces that, on the one hand, allows for accurately classifying a new trace as deviant or non-deviant and, on the other hand, is able to provide an informative explanation of the characteristics of these deviant cases. A good characterization of these deviant executions, expressed in terms of patterns that are easy to understand for analysts, would indeed allow them to have an idea of the causes that have generated these executions and hence where it is better to intervene to improve the process.

Most of the existing business process deviance mining approaches look at the problem as a classification one. Relevant patterns describing execution traces are used as features for encoding labeled traces. The labeled encoded traces are then used for training a classifier that is in charge of discriminating between deviant and non-deviant executions based on those patterns. The most obvious choice to describe execution traces [BvdA13], which are sequences of events, is resorting to *sequential* patterns, that is sequences of adjacent events.

However, other types of features can be used to describe a process case, as for instance *declarative* patterns, i.e., patterns related to the validity of predefined predicates in the case, or combinations of sequential and declarative patterns (*hybrid encoding*).

In addition to features extracted from control-flow perspective, it is also possible to extract additional features by making use of data attributes, which can help in characterizing differences between deviant and non-deviant executions.

Firstly, this thesis frames the problem of investigating the impact that different types of features (sequential, declarative and their combination) have on business process deviance mining results. Secondly, it provides two different ways of using data payload as additional features and their impact on business process deviance mining results. Using synthetic and real-life logs from multiple domains, a range of feature types are evaluated in terms of their ability to accurately discriminate between non-deviant and deviant executions of a process.

Thirdly, the thesis thesis takes a look at the outcomes returned to the users by the business process deviance mining approach. Two different methods returning decision rules are compared both in terms of classification performance and in terms of amount and length of decision rules (hence potentially investigate user readability). More concretely, the two methods leverage, respectively, decision trees and a procedure for the extraction of decision rules from the tree, and the RIPPER algorithm, which allows for direct extraction of decision rules. The evaluation is done to investigate the trade-off between the performance of the deviance mining approach and the complexity of the outcome rules.

The thesis is structured in the following way.

Section 2 gives an overview of previous research related to this thesis.

Section 3 introduces the necessary background knowledge to understand the concepts and techniques used.

In Section 4, a problem statement for business process deviance mining is given, which serves as a motivating example for this thesis.

Section 5 contains the different approaches used within the thesis. Section 5.1 introduces the pipeline used for business process deviance mining and how sequential, declarative and the combination of features were used. Section 5.2 gives the approach for the use of data-payload features. Section 5.3 covers the approach for extracting business process deviance mining outcomes in the form of decision rules.

Section 6 gives an overview on how the evaluation was carried out for each of the approaches and reports the results. In detail, Section 6.1 reports the procedure and results related to the impact of different types of features based on control-flow. Section 6.2 describes how the impact of the payload based features was evaluated and provides an overview of the results. Section 6.3 describes evaluation procedure and results related to the extraction of decision rules, returned by the deviance mining approaches.

Finally, Section 7 concludes the thesis, by giving an overview of the work done, the obtained results and possible future work.

2 Related work

The main works related to business process deviance mining can be classified into two main families: the ones using delta-analysis to (manually) identify differences between the model discovered from deviant and non-deviant cases (e.g.,[SMW$^+$14]) and those based on classification techniques [SWO$^+$13, PWS$^+$15, BvdA09, Bv13, LKL07, CHX11].

This work falls in the latter group. The works in this second group leverage classification techniques to discriminate between normal and deviant cases. These approaches usually discover patterns that are then used to build the classifier. They can be further classified based on the type of features used for training the classifier.

In [SWO$^+$13, PWS$^+$15], the authors use the frequency of individual activities in order to train classifiers in a financial and a clinical scenario, respectively. Bose and van der Aalst in [BvdA09, Bv13] employ sequential pattern mining to discover sequential patterns as tandem repeats, maximal repeats and alphabet repeats to be used as features for training a classifier. Similarly, in [LRW13], association rules are used to discover co-occurrence patterns in the context of deviant classes in a healthcare scenario. In [LKL07, CHX11], discriminative mining is used to discover discriminative patterns, i.e., patterns that, although not necessarily very frequent, clearly discriminate between deviant and non-deviant cases. Differently from these works that propose new patterns to be used as features, the aim of this thesis is investigating the impact of different types of features.

A benchmark collecting all these works and evaluating and comparing them in terms of different feature types and classifiers is presented in [NDLR$^+$14, NDR$^+$16].

Different types of patterns have also been combined together in [CFGP15, CFGP16, CFGP17]. In detail, in [CFGP15], in order to avoid the redundant representation deriving from mixing different families of patterns, the authors propose an ensemble learning approach in which multiple learners are trained encoding the log according to different types of patterns. In [CFGP16], event data payloads have also been taken into account in the discovery phase as well as in the classifier training. Finally in [CFGP17], the authors enhance the previous work [CFGP15] by proposing an alternative multi-learning approach probabilistically combining various classification methods. The focus of these latter works, however, is on the individual, sequential and discriminative patterns separately or on the combination of families of behavioral sequential patterns.

The finding of data-aware DECLARE constraints has been previously discussed in [MDGBM13]. In that work, differently from this one, the focus is on discovery of data-aware DECLARE models. In this thesis, the focus is on discovery of discriminating data-aware DECLARE constraints for use in deviance mining and by taking into account the labeling of event logs.

Differently and in addition from the previously described works this thesis:

- takes into account also a completely different and unexplored family of patterns,

i.e., the family of the declarative patterns;
- combines family of declarative patterns with sequential patterns (by facing the feature redundancy problem with feature selection approaches);
- evaluates the use of payload features and two different methods of payload feature extraction from execution logs;
- compares and evaluates two methods for extracting business process deviance mining outcomes in the form of decision rules.

3 Background

This chapter gives the necessary background information needed to understand the content of the thesis.

3.1 Business Processes and Logs

As the work in this thesis concentrates on business process event logs, this section first gives an overview on what is a business process and secondly on how its data representation looks like.

3.1.1 Business Process

A business process is a set of activites or events, which are performed in order to achieve a particular goal in a business operation.

Some types of common business processes include:

- Application-to-approval, where a sequence of activities related to the approval process, follow after the arrival of an application, with the end goal of the application being either approved or denied. Examples on where this type of processes could be found are in student's application process to universities or hiring process at companies. Typical activities here could be calling the references or scoring the application;
- Order-to-cash, which begins with a customer wanting to purchase a product (or use a service) and ends with the product being delivered and the payment for the product being received by the seller. Typical activities could include checking the inventory for stock or estimating the price quote for the customer.

Together with business processes being executed it is possible to track and store the process execution information for further analysis by turning the activities and events into execution traces, which are then collected in a form of event logs.

3.1.2 Event log

One of the possible data standards for storage and manipulation of event logs is *XES* standard [WGV14]. XES is acronym for eXtensible Event Stream, which is an XML-based format specialized for storage of event log data.

Figure 1 shows the UML [RJB04] diagram for the complete meta-model of XES standard. The basic hierarchy of a XES document contains one log object. One log object can contain any number of trace objects. Each trace can contain any number of event objects.

All event information related to a specific process is contained within a **log**. Some examples of a process could be:

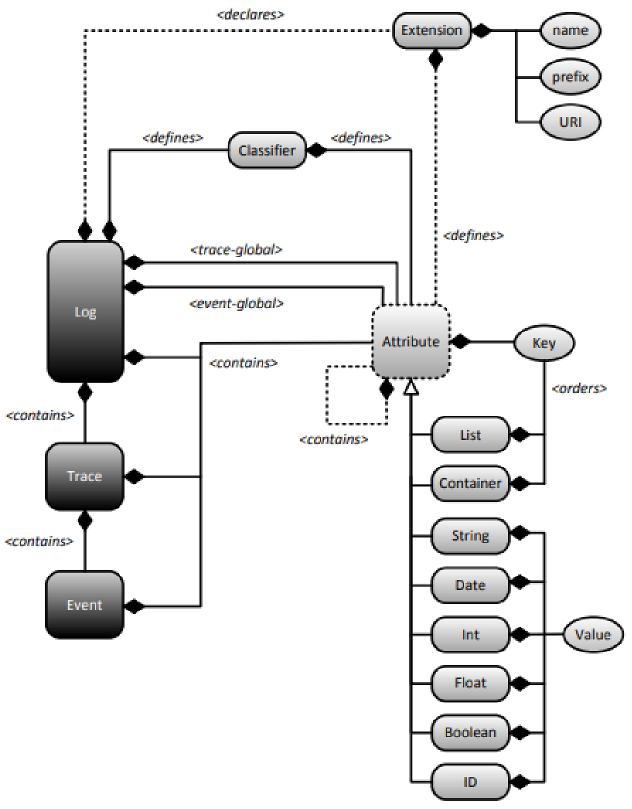


Figure 1. Meta-model for the XES standard [WGV14]

- A medical assessment process;
- A hiring process for new workers.

A **trace** describes a (usually) time-ordered execution of a specific process. Going by examples for processes, the corresponding trace examples could be:

- The specific assessment of an individual;
- The specific hiring of a worker for an organization.

An **event** represents an observed activity at an atomic-level. A possible event in traces given above could respectively be:

- The completion of adding blood tests results to individual's health records;
- The decision of hiring by a human resource specialist.

Attributes. Log, trace and event objects define the structure of XES document, but do not contain any information in them. The relevant information is instead stored in an arbitrary number of attributes, which describe their parent element [WGV14]. According to the standard [WGV14], there are 6 elementary attribute types: *String*, *Date*, *Int*, *Float*, *Boolean*, *ID*. Additionally the standard describes two collection type attributes *List* and *Container*. By itself XES standard does not specify a specific set of required attributes, but allows, by making use of the concept called extensions, to introduce commonly understood attributes. The commonly acknowledged set of attributes can be crucial for many event log analysis techniques [WGV14]. Some examples, which are included as standard extensions of XES are concept, lifecycle, time and organization, which are found in most of the logs. Additionally, for deviance mining purpose traces often have an attribute called Label, giving information on whether the trace is considered to be deviant or normal.

Several concepts are shown in example in Figure 2, which depicts the first events of a trace from log *Sepsis* (described later in Section 6.1.1). In the example a trace has attribute of type *String* with a key *concept:name* and a value *C*, which describes the name of the trace. In the first event of the trace there are many attributes of type *Boolean*. Moreover, each event has an attribute *concept:name* giving the name of the activity, *time:timestamp* describing the date when the event occured, *org:group* describing to which group/resource the event belongs to and *lifecycle:transition* which represents a stage of event's lifecycle, where "complete" states the completion of the activity.

```
<trace>
        <string key="concept:name" value="C"/>
        <event>
                <boolean key="InfectionSuspected" value="true"/>
                <string key="org:group" value="A"/>
                <boolean key="DiagnosticBlood" value="true"/>
                <boolean key="DisfuncOrg" value="false"/>
                <boolean key="SIRSCritTachypnea" value="false"/>
                <boolean key="Hypotensie" value="false"/>
                <boolean key="SIRSCritHeartRate" value="true"/>
                <boolean key="Infusion" value="true"/>
                <boolean key="DiagnosticArtAstrup" value="true"/>
                <string key="concept:name" value="ER Registration"/>
                <int key="Age" value="55"/>
                <boolean key="DiagnosticIC" value="true"/>
                <boolean key="DiagnosticSputum" value="false"/>
                <boolean key="DiagnosticLiquor" value="false"/>
                <boolean key="DiagnosticOther" value="false"/>
                <boolean key="SIRSCriteria2OrMore" value="true"/>
                <boolean key="DiagnosticXthorax" value="false"/>
                <boolean key="SIRSCritTemperature" value="true"/>
                <date key="time:timestamp" value="2014-02-09T19:29:29.000+01:00"/>
                <boolean key="DiagnosticUrinaryCulture" value="true"/>
                <boolean key="SIRSCritLeucos" value="false"/>
                <boolean key="Oligurie" value="false"/>
                <boolean key="DiagnosticLacticAcid" value="true"/>
                <string key="lifecycle:transition" value="complete"/>
                <string key="Diagnose" value="C"/>
                <boolean key="Hypoxie" value="false"/>
                <boolean key="DiagnosticUrinarySediment" value="true"/>
                <boolean key="DiagnosticECG" value="true"/>
        </event>
        <event>
                <string key="org:group" value="C"/>
                <string key="lifecycle:transition" value="complete"/>
                <string key="concept:name" value="ER Triage"/>
                <date key="time:timestamp" value="2014-02-09T20:05:23.000+01:00"/>
        </event>
        <event>
                <string key="org:group" value="A"/>
                <string key="lifecycle:transition" value="complete"/>
                <string key="concept:name" value="ER Sepsis Triage"/>
                <date key="time:timestamp" value="2014-02-09T20:05:33.000+01:00"/>
        </event>
```

Figure 2. An example of a portion of XES log.

In this work OpenXES was used, which is an open source reference implementation of First XES standard.

3.2 Log patterns

In this section an overview of relevant pattern types, which can be used to describe business processes event logs is given.

3.2.1 Sequential Patterns

Sequential patterns [Bv13] represent one of the pattern types that can be used to describe event log traces. Sequential patterns are sequences of events that occur frequently or in a non-deviant way within traces, thus capturing particular control flow relations in a process execution trace. Among the main types of sequential patterns, we can find:

- *Tandem Repeats (TR)*: this type of pattern denotes sequences of events that are repeated consecutively within a trace; these sequences correspond to process loops.
- *Maximal Repeats (MR)*: this type of pattern denotes maximal sequences of events that are repeated in an event log; these sequences correspond to process sub-processes.
- *Tandem Repeats Alphabet (TRA)*: this type of pattern denotes tandem repeats that share the same activities (i.e., the alphabet of unique activities); these sequences correspond to variations of TR taking into account process parallelism.
- *Maximal Repeats Alphabet (MRA)*: this type of pattern denotes maximal repeats that share the same activities (i.e., the alphabet of unique activities); these sequences correspond to variations of MR taking into account process parallelism.

For instance, given a trace $T = \langle \mathsf{a}, \mathsf{b}, \mathsf{c}, \mathsf{a}, \mathsf{b}, \mathsf{c}, \mathsf{d}, \mathsf{a}, \mathsf{b} \rangle$, the set of TR is $\{abc\}$, as abc is the only pattern to be repeated two times consecutively.

For TRA ,the ordering of activities within the pattern does not matter, but the patterns have to appear consecutively. In this case, given a trace $T = \langle \mathsf{a}, \mathsf{b}, \mathsf{c}, \mathsf{c}, \mathsf{b}, \mathsf{a}, \mathsf{a}, \mathsf{b}, \mathsf{c} \rangle$, the set of TRA is $\{abc, cb, ab, c, a\}$ with abc being repeated 3 times in a row, ab and cb twice in a row and a and c also twice in a row (without requiring specific order of events within the pattern).

A pattern is considered to be maximal repeat, if it cannot be extended to left or to right for a longer repeat covering all the subsequences of the shorter pattern. Considering trace $T = \langle \mathsf{a}, \mathsf{b}, \mathsf{c}, \mathsf{a}, \mathsf{b}, \mathsf{c}, \mathsf{d}, \mathsf{a}, \mathsf{b} \rangle$, the set of MR is $\{ab, abc\}$ (c and bc are not maximal, because both occurences can be extended to the left to be abc, which includes all c and bc withing the trace). Pattern ab is a maximal repeat, because it occurs multiple times and extending it would not cover the last occurence of ab.

For MRA, the ordering of activities within the pattern does not matter. For trace $T = \langle \mathsf{b}, \mathsf{a}, \mathsf{c}, \mathsf{a}, \mathsf{b}, \mathsf{c}, \mathsf{d}, \mathsf{b}, \mathsf{a} \rangle$, the set of MRA is $\{ab, abc\}$. Pattern abc occurs twice and is therefore a repeat. Pattern ab is repeated three times and is maximal, because it cannot be extended in a way that it covers all the occurences of ab. Patterns c, b are not maximal, because all c also occur within pattern abc, and all b within pattern ab, which are both maximal patterns. The MR and MRA in an event log are found by concatenating all traces in the event log in a single trace.

3.2.2 Declare

DECLARE was introduced in [Pes08] as a declarative process modeling language. The declarative approach for modeling was designed to be more flexible and able to grasp loosely-structured processes, in comparison to imperative approaches, which require a more rigid of a specification. A declarative approach, instead of telling users exact steps to take, allows to shift the decision making from the system to its users [PSv07].

The basic building block of a DECLARE model is a *constraint*, which is a template instantiated on a set of (atomic) activities. A *template* is an abstract parameterized pattern, which is instantiated with real activities [BCDFM14].

Linear Temporal Logic (LTL) has been the main choice of a logical language for formalizing the semantics of DECLARE templates [MPv$^+$10]. The semantics of LTL operators used for describing templates are provided in Table 2.

Additionally, DECLARE templates have a graphical notation, which makes them simpler to use and interpret for an analyst. Table 1 gives an overview of more commonly used DECLARE templates, their graphical representations, formalization in LTL and a textual description. The parameters of a template are in capital letters and real activities within constraints are in lower-case (e.g., `response`(a,b) is an instantiation of the template `response` and the activities a and b).

DECLARE templates can be grouped in three main categories: *existence* templates (first 4 rows of the table), which involve only one event; *relation* templates (rows from 5 to 12), which describe a dependency between two events; and *negative relation* templates (last row), which describe a negative dependency between two events.

To give some examples on constraints let's consider the following four traces:

1. $\langle \mathsf{a}, \mathsf{a}, \mathsf{b}, \mathsf{c} \rangle$;
2. $\langle \mathsf{b}, \mathsf{b}, \mathsf{c}, \mathsf{d} \rangle$;
3. $\langle \mathsf{a}, \mathsf{b}, \mathsf{c}, \mathsf{b} \rangle$;
4. $\langle \mathsf{a}, \mathsf{b}, \mathsf{a}, \mathsf{c} \rangle$.

First consider a simple constraint on an existence template `init`(a). By looking at the description of the `init` template in Table 1 and replacing the parameter with real activity a we can read the description as following: "Each instance has to start with the execution of a". In our samples we can see that it is satisfied in traces 1, 3 and 4, but not satisfied for trace 2, since it does not start with a.

For a second example, consider a constraint based on relation template: `response`(a, b). Referring to Table 1 again and replacing parameter A with a and parameter B with

Table 1. Graphical notation and LTL formalization of some Declare templates [BCDFM14].

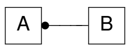
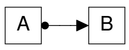
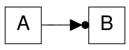
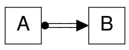
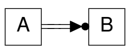
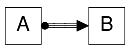
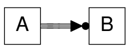
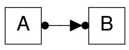
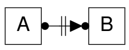
TEMPLATE	FORMALIZATION	NOTATION	DESCRIPTION
EXISTENCE TEMPLATES			
`existence(n,A)`	$\Diamond(A \wedge \bigcirc(existence(n-1, A)))$	$\overset{n..*}{\boxed{A}}$	A has to occur at least n times
`absence(n,A)`	$\neg existence(n, A)$	$\overset{0..n}{\boxed{A}}$	A can happen at most n-1 times
`exactly(n,A)`	$existence(n, A) \wedge absence(n+1, A)$	$\overset{n}{\boxed{A}}$	A has to occur exactly n times
`init(A)`	A	$\overset{init}{\boxed{A}}$	Each instance has to start with the execution of A
RELATION TEMPLATES			
`resp. existence(A,B)`	$\Diamond A \rightarrow \Diamond B$		If A occurs, B must occur as well
`response(A,B)`	$\Box(A \rightarrow \Diamond B)$		If A occurs, B must eventually follow
`precedence(A,B)`	$\neg B \, \mathcal{W} \, A$		B can occur only if A has occurred before
`alternate response(A,B)`	$\Box(A \rightarrow \bigcirc(\neg A \, \mathcal{U} \, B))$		If A occurs, B must eventually follow, without any other A in between
`alternate precedence(A,B)`	$(\neg B \, \mathcal{W} \, A) \wedge$ $\Box(B \rightarrow \bigcirc(\neg B \, \mathcal{W} \, A))$		B can occur only if A has occurred before, without any other B in between
`chain response(A,B)`	$\Box(A \rightarrow \bigcirc B)$		If A occurs, B must occur next
`chain precedence(A,B)`	$\Box(\bigcirc B \rightarrow A)$		B can occur only immediately after A
`succession(A,B)`	$\neg B \, \mathcal{W} \, A \wedge \Box(A \rightarrow \Diamond B)$		A occurs if and only if B occurs after A
NEGATIVE RELATION TEMPLATES			
`not succession(A,B)`	$\Box(A \rightarrow \neg\Diamond B)$		A cannot occur before B

Table 2. SEMANTICS OF LTL OPERATORS [MMv11].

operator	*semantics*
$\bigcirc\varphi$	*φ has to hold in the next position of a path.*
$\Box\varphi$	*φ has to hold always in the subsequent positions of a path.*
$\Diamond\varphi$	*φ has to hold eventually (somewhere) in the subsequent positions of a path.*
$\varphi \, U \psi$	*φ has to hold in a path at least until ψ holds. ψ must hold in the current or in a future position.*
$\varphi \, W \psi$	*φ has to hold in the subsequent positions of a path at least until ψ holds. If ψ never holds, φ must hold everywhere.*

b the constraint's description is "'If a occurs, then b must eventually follow". From our samples we can first see that this condition is satisfied in traces 1, 2 and 3, but not satisfied in trace 4, as the second occurrence of a is not eventually followed by a b.

Activation of a constraint. An *activation* of a constraint in a trace is an event, that on appearance imposes obligations on another event (the *target*) in the context of same trace

[BCDFM14]. In constraint response(a,b) a is an activation, because the execution of a imposes an obligation to b, forcing it to be eventually executed. Event b is a target. Referring back to sample traces again, one can see that in traces 1, 3 and 4 a occurs somewhere in the trace, therefore the constraint response(a,b) is activated on those traces. Trace 2 does not include a and the constraint is therefore not activated in that trace.

An activation of a constraint in a trace is either a *fulfillment* or a *violation* for that constraint within the trace. If every activation of a constraint in a trace leads to a fulfillment, then the constraint is *satisfied* for that trace. Taking a look at constraint response(a,b), it was activated in traces 1, 3 and 4. For trace 1 ($\langle \mathsf{a}, \mathsf{a}, \mathsf{b}, \mathsf{c} \rangle$) it is activated twice and both occurrences of a are eventually followed by b, therefore they are also both fulfilled. In sample trace 3, the constraint is activated and fulfilled once. In trace 4 - $\langle \mathsf{a}, \mathsf{b}, \mathsf{a}, \mathsf{c} \rangle$, the constraint is activated twice, but the second activation a, is not eventually followed by b and therefore leads to a violation. The trace is considered *unsatisfied* for the constraint, if at least one activation leads to a violation.

Vacuous satisfaction. There exist cases in which the constraint is not activated at all. Consider, for instance, the sample trace 2 - $\langle \mathsf{b}, \mathsf{b}, \mathsf{c}, \mathsf{d} \rangle$. The considered *response* constraint is satisfied in a trivial way in this trace, due to a never occurring. In such a case, the constraint is considered to be *vacuously satisfied* [KV99].

3.2.3 Data-Aware Declare Constraints

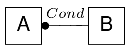
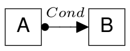
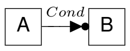
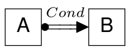
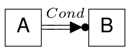
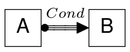
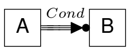
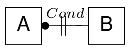
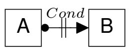
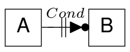
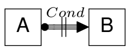
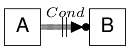
While DECLARE in its original form is mainly used to set constraints on control-flow aspects of a process, data-aware DECLARE constraints extend DECLARE constraints so as to include conditions on data [MCMM13]. Data-aware DECLARE sets additional requirement on data for an activation of a constraint to happen. The difference between graphical notations and formalizations of DECLARE (see Table 1) and data-aware constraint templates (see Table 3) is the data condition.

Activation of a data-aware constraint. For demonstration of applying data-aware declare constraints, let's consider the following examples of traces, which are modified versions from the previous section on DECLARE:

1. $\langle \mathsf{a}\{g = 1\}, \mathsf{a}\{g = 2\}, \mathsf{b}, \mathsf{c} \rangle$;
2. $\langle \mathsf{b}\{g = 1\}, \mathsf{b}\{g = 1\}, \mathsf{c}, \mathsf{d} \rangle$;
3. $\langle \mathsf{a}\{g = 2\}, \mathsf{b}, \mathsf{c}, \mathsf{b} \rangle$;
4. $\langle \mathsf{a}\{g = 1\}, \mathsf{b}, \mathsf{a}\{g = 2\}, \mathsf{c} \rangle$.

In the traces $\mathsf{a}\{g = 1\}$ stands for activity a, which has an attribute *g* that has a value of *1*.

Table 3. Graphical notation and LTL formalization of some Data-aware Declare constraint templates [MDGBM13].

TEMPLATE	FORMALIZATION	NOTATION	DESCRIPTION
RELATION TEMPLATES			
`resp. existence`(A,B,Cond)	$\Diamond(A \wedge Cond) \rightarrow \Diamond B$		If A occurs and Cond holds, B must occur as well
`response`(A,B,Cond)	$\Box((A \wedge Cond) \rightarrow \Diamond B)$		If A occurs and Cond holds, B must eventually follow
`precedence`(A,B,Cond)	$\neg(B \wedge Cond)\, \mathcal{W}\, A$		If B occurs and Cond holds, A must have occurred before
`alternate response`(A,B)	$\Box((A \wedge Cond) \rightarrow \bigcirc(\neg A\, \mathcal{U}\, B))$		If A occurs and Cond holds, B must eventually follow, without any other A in between
`alternate precedence`(A,B,Cond)	$(\neg(B \wedge Cond)\, \mathcal{W}\, A) \wedge$ $\Box((B \wedge Cond) \rightarrow \bigcirc(\neg B\, \mathcal{W}\, A))$		If B occurs and Cond holds, A A must have occurred before, without any other B in between
`chain response`(A,B,Cond)	$\Box((A \wedge Cond) \rightarrow \bigcirc B)$		If A occurs and Cond holds, B must occur next
`chain precedence`(A,B,Cond)	$\Box(\bigcirc(B \wedge Cond) \rightarrow A)$		If B occurs and Cond holds, A must have occurred immediately before
NEGATIVE RELATION TEMPLATES			
not resp. existence(A,B,Cond)	$\Diamond(A \wedge Cond) \rightarrow \neg\Diamond B$		If A occurs and Cond holds, B cannot occur
not response(A,B,Cond)	$\Box((A \wedge Cond) \rightarrow \neg\Diamond B)$		If A occurs and Cond holds, B cannot eventually follow
not precedence(A,B,Cond)	$\Box(A \rightarrow \neg\Diamond(B \wedge Cond))$		If B occurs and Cond holds a cannot have occurred before
not chain response(A,B,Cond)	$\Box((A \wedge Cond) \rightarrow \neg\bigcirc B)$		If A occurs and Cond holds, B cannot occur next
not chain precedence(A,B,Cond)	$\Box(A \rightarrow \neg\bigcirc(B \wedge Cond))$		If B occurs and Cond holds A cannot occur immediately before B

Consider a data-aware constraint `response`(a,b,{g=1}). This constraint requires the trace to have an activity a with an attribute $g = 1$ for it to be an activation. In sample Trace 1, the first activity a is an activation but the second one is not, due the data condition not being fulfilled. The only activation in this trace leads to a fulfillment and therefore in that trace the constraint is satisfied. Trace 2 does not have any activations and is therefore *vacuously* satisfied. In Trace 3, there are no activations either, due to the sole activity a not having the matching data condition, which makes the constraint vacuously sastisfied. Trace 4($\langle \mathsf{a}\{g = 1\}, \mathsf{b}, \mathsf{a}\{g = 2\}, \mathsf{c}\rangle$) is activated once on the first a and leads to a fulfillment and therefore the constraint is satisfied.

3.3 Classification

In machine learning, the aim of classification is to identify, based on a training set of observations, in which category a previously unseen observation belongs to [Alp14]. A classifier is an algorithm, which implements the classification procedure. In business process deviance mining, classification techniques are widely in use (as described in Section 2) as they allow to classify traces as normal or deviant. In this thesis they play a central role in evaluating different types of features and finding characteristics, which describe differences between deviant and normal traces.

Classification approaches are usually based on two phases: a training and an evaluation phase. The training of a classifier refers to learning a mapping between observations and their categories. A common way of training classifiers is to have your dataset split into two non-overlapping partitions: training set and test set. When performing hyperparameter optimization (e.g. choosing maximum depth of a decision tree), it is often wise to also consider a third partition, often referred to as validation set, on which the hyperparameters are selected. The classifiers are then trained on training set, optionally optimized by making use of a validation set and then the final performance evaluated on a test set. Evaluation is done to measure a classifier's performance, by calculating different comparable metrics.

In this thesis the observations are traces, which are encoded as vectors of numbers (encoding described in 5.1). Each trace has a category label of either being deviant or normal. The event log is split into partitions of training and test set.

There are many different classifiers such as perceptrons, logistic regression models, support vector machines, artificial neural networks, decision trees and rule-based learners. In this thesis decision trees and rule-based learner RIPPER (described in Section 3.3.2) were considered, which allow for simple extraction of interpretable classification rules.

3.3.1 Decision tree

Decision trees are one of the more popular approaches in predicting and characterizing relationships between observations and their target values [RM08]. Decision trees can be used for both classification and regression tasks, namely known as classification trees and regression trees. Regression trees targets are real values (e.g. price of a property). Classification trees target values are a classes/categories (e.g. car is a sedan or an SUV). In this work the focus is on the task of classification and therefore classification trees are used.

An example of a classification tree can be seen Figure 3, where a toy decision process of either approving or disapproving a loan based on attributes of present loan application is depicted. Internal nodes are attributes, based on which a decision at that point is made. The decision either follows left or right path depending on the conditions shown on the edges. The leaves show the result of classification, which in the sample is either a rejection or an approval of a loan. For example, if an applicant has salary over 2000 and has a credit score above 5, then the loan is approved.

Since finding an optimal decision tree is often computationally infeasible, the methods of decision tree construction are mostly heuristic [RM08]. The input for classification trees is a set of observations with their respective categories. Decision tree construction algorithms mostly work recursively in a top-down and a greedy manner [RM08]. At every iteration the algorithm picks an attribute and a corresponding condition, which best split the observations according to some metric. Two more common metrics are Gini impurity (it measures how often a randomly chosen observation is incorrectly classified when considering the attribute) and Information gain (it measures the purity of resulting new nodes). On each partition, which resulted from the split, the process is repeated until some stopping criterion is fulfilled. Examples of stopping criteria are the limitation on the maximum depth of the tree or when all the considered observations have the same category [RM08].

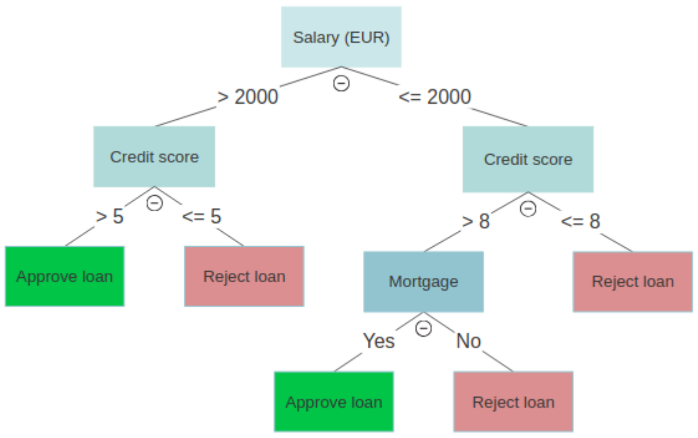


Figure 3. Example of a classification tree.

Decision rule is an IF-THEN statement, which consists of a conjunction of conditions and the target prediction. An example of a decision rule could be: *IF* age is above 50 *AND* sex is male *THEN* person is in a risk group. A single decision rule or a set of decision rules can be used to make predictions. Getting a set of decision rules can either be done in a direct or an indirect manner. Direct is when rules are extracted directly from the data and indirect when a classifier, such as a decision tree, is first trained and then rules are extracted.

Trained classification trees can be turned into a set of decision rules in a straightforward manner [RM08]. Rules can be extracted by considering all the paths from a root of a decision tree to its leaves, by conjoining the decisions made along the path from the root to leaves with desired labels. The category of a leaf is the same as the majority of training observations within the leaf. The categories of the leaves will be the predicted categories for the rules. An example rule extracted from a decision tree depicted in Figure 3 is: *IF* Salary (EUR) <= 2000 *AND* Credit score > 8 *AND* Mortgage = Yes *THEN* Approve loan. As this method takes a decision tree as an input, it is an indirect method for getting decision rules.

3.3.2 RIPPER

Repeated Incremental Pruning to Produce Error Reduction (RIPPER) is a propositional rule learner first which was proposed by Cohen, W. W. as an improvement to algorithm *Incremental Reduced Error Pruning* (IREP) [Coh95]. Contrary to first building a decision tree, RIPPER learns rules directly from the data. Generalization performance of RIPPER is very competitive with a C4.5rules algorithm [Coh95], which is based on a well known C4.5 decision tree building algorithm [Qui93]. RIPPER is able to work with both nominal and continuous data. A toy ruleset which resembles RIPPER output and covers same set of rules as decision tree in Figure 3 is shown in Figure 4.

```
(Salary > 2000) and (Credit score > 5) => Label=1
(Salary <= 2000) and (Credit score > 8) and (Mortgage = Yes) => Label=1
 => Label=0
```

Figure 4. Sample output of RIPPER. First two lines are rules describing positive classes (Label=1). Last line shows the default rule when none of above applies (Label=0).

Sequential covering. RIPPER is considered to be a variant of sequential covering algorithms [Mol19], which are general procedures to learn rules one-by-one in order to create a list of decision rules. The basic scheme of sequential covering is the following one:

- Find a good decision rule, which covers some observations;
- Remove observations covered by the found rule;
- Repeat until rules cover all observations or another stopping criterion (e.g. the found rule is of too low quality) is met.

The graphical overview of such procedure is shown on Figure 5.

The procedure of RIPPER is seperated into two stages: the first stage builds the initial ruleset whileas the second stage optimizes the generated rules for better generalization capabilities. The building of the initial ruleset is further done in two parts: growing and pruning. In two-class classification scenarios, that is also worked with in the thesis, one class is deemed to be positive and the other negative.

Building the set of rules. In order to grow a rule, the training set of observations is first randomly split into two sets: a growing set and a pruning set, which is done in 2-to-1 ratio, with $\frac{2}{3}$ of observations in growing set and $\frac{1}{3}$ in pruning set. Growing of a rule is done greedily, by iteratively adding conditions, that maximize first-order inductive learner's (FOIL) information gain. FOIL's information gain is calculated as

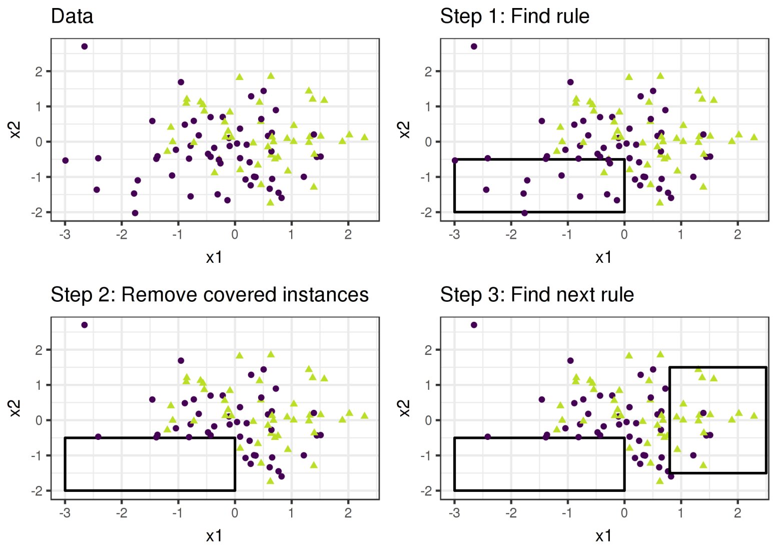


Figure 5. Feature space is sequentially covered with one rule at a time. After each rule, observations (instances) covered by that rule are removed from further consideration. [Mol19].

$IG = p \cdot (\log_2 \frac{p}{t} - \log_2 \frac{P}{T})$, where P and p are counts of positive observations covered by the rule before and after the addition of a condition, T and t are counts of total observations covered before and after. Rule is grown until it is perfect - e.g. it no longer covers negative examples (if possible).

After each rule is grown, it is immediately pruned using a procedure called reduced error pruning (REP). Pruning is done iteratively, by deleting any final sequence of conditions from the rule. The procedure is repeated until the pruning measure $V(rule)$ no longer increases. For pruning the following measure is used: $V(rule) = \frac{p-n)}{p+n}$, where p and n are the counts of positive and negative observations covered by the rule in the pruning set.

Both negative and positive observations covered by the pruned rule are now removed from the training set and the process of growing and pruning is repeated. The stopping condition occurs when there are no more positive observations remaining, the accuracy of the rule to be added is less than 50% or the description length of the ruleset has grown considerably when compared to the simplest ruleset. Description length is an information theoretical heuristic, which helps in balancing between maximizing accuracy on the train set and minimizing the complexity of the ruleset [Qui95].

Optimizing the set of rules. In the optimization part every rule, in the same order in which they are found, is checked as a candidate for optimization. For every rule two alternatives are considered: one by replacement and second by revision. For replacement, a new rule is grown and then pruned in such a way that it minimizes the error of the entire set of rules. For revision, extra conditions are added to the rule. Out of three rules: initial, replacement and revision, the one which minimizes the description length of the entire ruleset is chosen. If after optimizing the ruleset, there are some uncovered positive observations, new rules are built using the initial way of growing and pruning rules. The same optimization process can be run several times in order to further optimize the ruleset.

4 Problem

In this section is provided an example, motivating throughout the thesis the importance of approaches for business process deviance mining and the advantages of exploring different types of features. Let us consider the customer support process carried out by a big company selling services. The support process aims at helping customers whenever a problem occurs or when special features are requested. The process, depicted in BPMN in Figure 6, starts when the customer support office receives a request from a client. The customer support office registers the request and performs a first evaluation. If the request can be easily solved, it is solved and the customer notified. In case the request is difficult to solve, a more in-depth evaluation is carried out, a solution is developed and the customer notified, waiting for her feedback on the proposed solution. If the customer is not happy with the solution, the issue is evaluated again in-depth, an alternative solution found and the customer notified again until a good solution is found. In both cases (easy and complex requests), if the request was unknown up to that time, a report has to be prepared documenting the request and the solutions before the request can be closed.

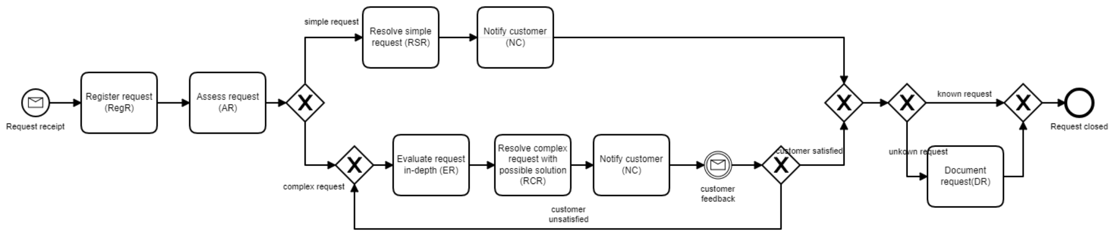


Figure 6. Customer support example

The company's business process analysts have noticed that some of the executions of the customer support process take longer than others, i.e., these executions *deviate* with respect to the expected process cycle time. Analysts are hence interested in understanding what these deviant cases are, how to characterize them and the reasons why they take more time. This information, is indeed crucial for them to be able to improve the process and avoid the occurrence of these delayed executions in the future.

5 Approach

5.1 Exploring Business Process Deviance with Sequential and Declarative Patterns

Fig. 7 shows the pipeline used for experimenting the business process deviance mining approach. The pipeline starts with discovering discriminative features from a labeled event log. Feature discovery is carried out by using both sequential pattern discovery and DECLARE discovery methods. Due to a large number of features that can be discovered, the next step is to extract a subset of features that are used to encode traces. Traces are encoded as numeric feature vectors where each element of the vector represents the number of occurrences in the trace of each pattern selected in the previous step. The feature vectors are used to train a decision tree (or the RIPPER model). The last step is to extract rules from the classifier (or to query the RIPPER model), which characterize deviant cases thus explaining the cause of the deviations.

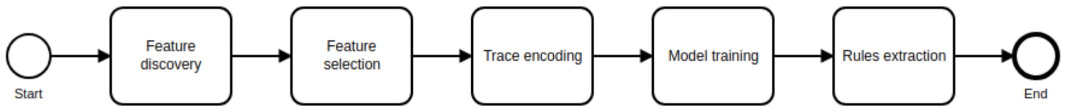


Figure 7. High-level pipeline

5.1.1 Feature Discovery

In order to transform traces into numeric feature vectors, the first step is to discover sequential and declarative patterns.

Sequential patterns are discovered by using the approach presented in [Bv13] to extract TR, TRA, MR and MRA.

The initial steps for Declare pattern discovery are similar to what was done in [MBvdA12], with the purpose of discovering declarative process models. The process of discovering declarative patterns starts with finding frequent sets of activities within an event log. It is done by using a procedure similar to Apriori algorithm described in [AS], which is frequently used for association rule mining. The found frequent sets of activities will be used to instantiate DECLARE templates in order to create a set of candidate constraints.

An activity set (a set of activies) is *frequent*, if it has a support above a given activity set support threshold $supp_{\min}$. *Support* is a measure that assesses the relevance of the activity set within an event log. The support of an activity set is calculated by finding the fraction of traces in an event log, which contains all the activities in the set.

The scheme for finding frequent sets of activities is quite simple. The process starts with finding a set of all unique activities in an event log, let this set be Σ. Next, consider all single activities in Σ as activity sets of size 1. Next support is computed for all the sets and frequent ones are kept.

Activity sets of size $2..k$, where k is the largest amount of parameters any DECLARE template in consideration accepts, can be found by considering the following iterative procedure. Given frequent activity sets of size $t-1$:

1. Create sets of size *t* by joining relevant frequent activity sets of size *t-1*;
2. Prune set by applying property that a frequent candidate set cannot contain an infrequent subset;
3. Keep frequent sets of size *t* (activity sets with the support above $supp_{\min}$).

The output of the procedure is the union of the activity sets of sizes $1..k$,

The constraints are discovered by considering DECLARE templates together with found activity sets. A DECLARE template with n parameters is instantiated with all permutations of each frequent set of size n. The generation of permutations of frequent sets is required, because for most of the templates the ordering of activities used as parameters matters. As an example, given a frequent activity set $\{a, b\}$ and template *response*(a,b), two constraints are generated: $\Box(a \rightarrow \Diamond b)$ and $\Box(b \rightarrow \Diamond a)$.

Finally, we check if each candidate constraint is satisfied in a percentage of either deviant or non-deviant traces that is above a given minimum constraint support threshold (usually set as $supp_{\min}$).

5.1.2 Feature Selection

The number of features generated from the previous step can turn out to be too large. Therefore, it becomes important to remove the features that do not give much value for training the explanatory model. Having too many features contributes to long training times, overfitting and too complex classifiers. For this reason there is a need to perform feature selection. In this step, first, all duplicate and constant features are removed. Then, the features are selected by using two different methods both using Fisher score [DHS01] as basis (refer to Equation 1). The methods differ in the way the features are chosen. In the first method, the first k features ranked according to the Fisher score are selected. In the second one (called coverage method, described in [Jar16]), a number of features is selected by first ranking them according to the Fisher score. Going by the order of ranking, features are selected until every trace is covered by at least a fixed number (coverage threshold) of features. Feature is only chosen if it covers at least one of the remaining traces. Note that for the hybrid encoding the feature selection is carried out by putting sequential and declarative features together in the same ranking.

Fisher score for *j-th* feature is computed as following:

$$F_j = \frac{\sum_{i=1}^{c} n_i(\mu_i - \mu)^2}{\sum_{i=1}^{c} n_i \sigma_i^2} \tag{1}$$

, where n_i denotes number of data points in class *i*, μ_i and σ_i^2 denote mean and variance of class i corresponding to *j-th* feature. μ and σ are mean and variance of all data point corresponding to j-th feature.

5.1.3 Trace Encoding

In this step, the input log is transformed into a numeric dataset, which can then be used to train a decision tree.

Sequential Encoding. Each trace in the log is transformed into a numeric feature vector where each element of the vector corresponds to the number of occurrences of a pattern (taken from the list of selected features) in the trace. This is done for TR, TRA, MR and MRA. In our evaluation, we included in this encoding also the frequency of each individual activity (in the log alphabet Σ) in the trace. At the end of this step, the event log is transformed into a matrix of numerical values where each row corresponds to a trace and each column corresponds to a feature.

Declarative Encoding. Each trace in the log is transformed into a numeric feature vector as follows:

- a feature has value -1, if the corresponding DECLARE constraint is violated in the trace;
- a feature has value 0, if the corresponding DECLARE constraint is vacuously satisfied in the trace;
- a feature has value n, if the corresponding DECLARE constraint is satisfied and activated n times in the trace.

Also in this case, the event log is transformed into a matrix of numerical values where each row corresponds to a trace and each column corresponds to a feature.

To give an example of the above encoding scheme above consider the following example.

For instance, given a trace $T = \langle \mathsf{a}, \mathsf{b}, \mathsf{c}, \mathsf{a}, \mathsf{b}, \mathsf{c}, \mathsf{d}, \mathsf{a}, \mathsf{b} \rangle$:

- constraint Response(a,c) is unsatisfied, due to its third activation leading to a violation and is therefore encoded as -1;
- constraint Response(a,b) is satisfied and activated 3 times, hence it is encoded as 3;
- constraint Response(e,b) is satisfied vacuously and encoded as 0.

Hybrid Encoding. In the hybrid encoding, the features are selected from a common ranking of sequential and declarative features and each trace is encoded into a numerical feature vector as explained in the previous two paragraphs depending on whether the feature is sequential or declarative.

5.1.4 Model Training

For each type of encoding described, a decision tree is trained [1] to both classify new unseen traces (thus being able to evaluate the performance of the classification) and explain the classification with explicit rules. For this reason, decision trees, which are "white-box" classifiers are chosen.

5.1.5 Rule Extraction

From the decision tree classifier, the rules are extracted in a way that rules are corresponding to the conjunction of the atomic conditions encountered in each path from the root of the tree to the leaves labeled as deviant (or non-deviant). The conjunction of range conditions on the same feature is simplified by merging (when possible) the conditions and by removing subsumed conditions related to the same feature. In Fig. 8, the conjunction of the violation of a *not succession* constraint, the satisfaction of a *choice* constraint activated up to 3 times and a MR sequence not appearing in a trace (sequence encoded as dL8dU3_mr) characterize the trace as non-deviant. Changing the last condition with a condition requiring the encoded MR to exist in the trace labels the trace as deviant.

5.1.6 Application

Consider the motivating example presented in Section 4. By applying the business process deviance mining pipeline with sequential features, analysts would get as outcome that deviant cases are those for which the pattern $\langle \mathsf{ER}, \mathsf{RCR}, \mathsf{NC} \rangle$ is repeated more than 2 times (i.e., for which the TR occurs). However, although many of the deviant cases seem to be captured by this explanation, it is not sufficient for characterizing all of them: the classification accuracy is not 100%.

The analysts can then try to apply the pipeline with declarative features. The outcome, in this setting, is different: deviant cases are those for which the activity Resolve Simple Requests is eventually followed by the activity Document Request (i.e., for which a `response`(RSR, DR) occurs). Still, this explanation is not sufficient for capturing all the deviant cases.

[1] In order to reduce overfitting, the max depth of the tree is fixed to 10 and the leaves of the decision tree are forced to be of at least size 5.

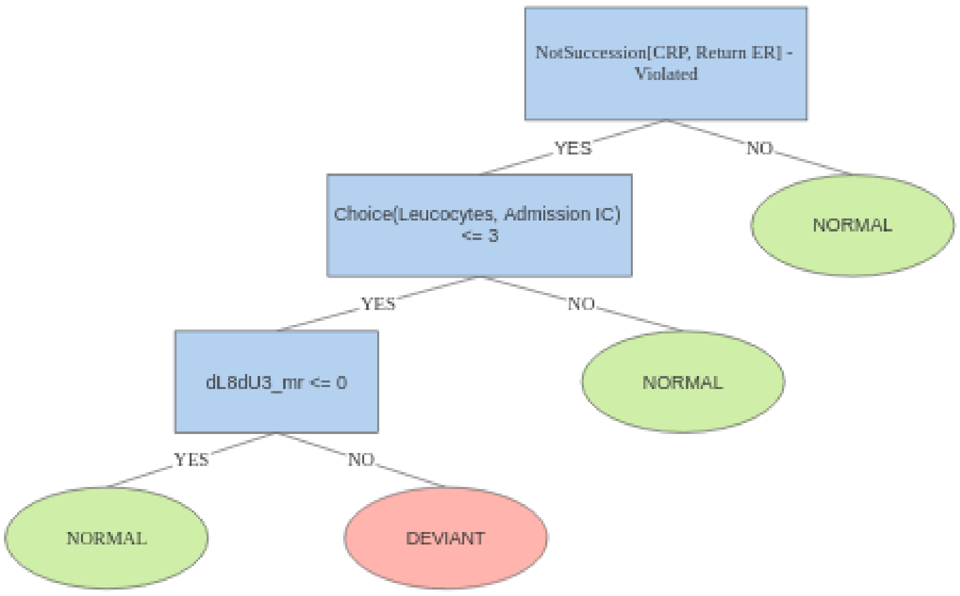

NotSuccession[CRP, Return ER] - Violated
YES
NO
Choice(Leucocytes, Admission IC) <= 3
NORMAL
YES
NO
dL8dU3_mr <= 0
NORMAL
YES
NO
NORMAL
DEVIANT

Figure 8. Example of a decision tree

By applying the pipeline with the hybrid features, the analysts are finally able to get the complete picture thanks to both the sequential and the declarative explanations.

The hybrid encoding allows hence analysts to realize that there are two situations in which the process executions take longer:

- when, in case of a difficult request, more than 2 iterations are carried out with the customer, until the customer is satisfied.
- when customer support operators procrastinate writing reports related to easy request resolutions. Indeed, differently from unknown complex requests, for which operators tend to complete the documentation immediately after the resolution, reports for easy requests tend to be delayed and the requests cannot be closed quickly.

5.2 Exploring Payload Features for Business Process Deviance Mining

In order to evaluate the use of payload features for deviance mining, two different methods for the extraction and the usage of data attributes of event logs are tested. The first method considers trace control-flow and data-attributes separately. The extracted features using the first method are referred to as *Pure data* features. The second method makes use of *Data-aware Declare constraints* (described in Section 3.2.3), in order to infuse data conditions into DECLARE constraints.

The pipeline starts with log preprocessing in order to remove ambiguity from having multiple copies of same data attributes in one set of trace or event attributes. The next step is to discover new payload features based on the methods described in the following subsections (i.e. pure data or data-aware DECLARE features). The found features are added to the set of features extracted in Section 5.1 before performing feature selection. The rest of the procedure is the same as in Section 5.1: resulting features are used to encode traces into numeric feature vectors, the feature vectors are used to train a decision tree and rules are extracted.

5.2.1 Log preprocessing

In some logs, it could happen that the same attribute occurs multiple times with different indexes. In order to use labelled logs for payload extraction, the problem of attributes appearing multiple times in the same event or in the same trace, with different attributes, has to be solved. In this case the attribute, with highest index is kept. For example, if a trace (or an event) has attribute in a form of *{age:1 = 1}, {age:2 = 2}*, then the one with largest index is kept and the other ones sharing the same prefix *age* are removed. In this case, what remains is *{age: = 2}*.

5.2.2 Pure Data method of payload feature creation

Among the two approaches proposed for enriching the set of features with data payload the *Pure data* one is the more straightforward one. The method creates new features, by either extracting trace attributes and event attributes from payload or by the use of meta-information (i.e. number of events in a trace), which are then encoded into numeric vectors. Since different events along the trace can share the same attribute with different values, i.e. the value of an attribute can evolve along events in the trace. In order to associate a single value to a subset of values of each event attribute, either one of the values of the attribute is picked based on its position of or an aggregated value of its values across different events is computed.

Feature extraction. The first step step is to extract attributes from event logs.

Extracting payload features from trace attributes. The extraction of trace attributes is simple - the created features correspond to attributes occuring as a trace attribute, with each of them having the same value as in the trace (e.g. if trace has attribute *Cost=30*, then the extracted feature is exactly the same). In this thesis, for payload extraction, types *String*, *Int*, *Float* and *Boolean* were considered

Extracting payload features from event attributes. The first way of extracting event attributes is by considering the value held by the attribute based on the position in the trace of the event in which it occurs. This type of (position-based) feature extraction can be applied for any type of attribute. In detail, two methods of extraction were considered:

- **Choose first** - The value of the feature is the first occurence of the attribute in the trace;
- **Choose last** - The value of the feature is the last occurence of the attribute in the trace.

For example consider the trace $\langle \mathsf{a}\{g = 1\}, \mathsf{a}\{g = 2\}, \mathsf{b}\{g = 3\}, \mathsf{c}\rangle$. For attribute *g* and extraction-method *Choose first*, the selected value for *g* is 1, as it is the value of its first occurence. With *Choose last*, the selected value for *g* is 3 as the value of its last occurence.

The second way of extracting payload from event attributes is by aggregating the values that each attribute assumes along the trace. For aggregation, four different methods have been considered:

- **Count** - A new aggregated feature is created for each value the attribute takes. Each feature has an integer value counting how many times the attribute holds that value in a trace. This is used for attributes of type *String*;
- **Choose max.** - A new aggregated feature is created by finding the maximum value the attribute holds in the trace. This type of extraction is used for attributes of type *Int* and *Float*;
- **Choose min.** - A new aggregated feature is created by finding the minimum value the attribute holds in the trace. This type of extraction is used for attributes of type *Int* and *Float*;
- **Compute avg.** - A new aggregated feature is created by computing the average of values the attribute holds in the trace. This type of extraction is used for attributes of type *Int* and *Float*.

To provide examples on aggregated features, consider again the previous trace $\langle \mathsf{a}\{g = 1\}, \mathsf{a}\{g = 2\}, \mathsf{b}\{g = 3\}, \mathsf{c}\rangle$ and attribute *g*. If methods *Choose max* and *Choose min* are used, then the new feature would be *g=3* and *g=1* respectively. If method

Compute avg is used, then the new feature is created by calculating the mean of all values that attribute *g* takes, in this case the new feature is $g = (1 + 2 + 3)/3 = 2$. In order to give an example on *Count* method, consider the trace $\langle \mathsf{a}\{\mathit{color} = \mathit{white}\}, \mathsf{a}\{\mathit{color} = \mathit{black}\}, \mathsf{b}\{\mathit{color} = \mathit{white}\}, \mathsf{c}\rangle$. In this case we have an attribute named *color* which takes two different values: *white* and *black*. A new feature is created for each of the unique value. In this situation 2 new features are created. The first feature is *color:white=1* and the second feature is *color:black=2*. The values associated to the features 1 and 2 respectively, corresponding to how many times *white* and *black* occured for the attribute *color*.

Extraction of meta-information. The third way of extracting data is by taking into consideration meta-information, which is not specifically contained within attributes themselves. Instead, they are created for example by calculating the total time length of a trace or the count of events in the trace. Currently one method of meta-information extraction was considered:

- **Trace length** - A new feature is created by finding the number of events in a trace.

For example, if we have a trace $\langle \mathsf{a}\{g = 1\}, \mathsf{a}\{g = 2\}, \mathsf{b}\{g = 3\}, \mathsf{c}\rangle$, then with method *Trace length*, the new feature would have a value of 4 corresponding to the event count in the trace.

Trace encoding. Given *Pure data* features, the traces are encoded as numeric vectors in the following way:

- resulting features created using aggregation methods (count, max, min and avg) are all numerical features and are encoded as they are;
- trace-attributes and attributes extracted with position-based extraction method of type *Float* or *Int* are encoded as they are;
- trace-attributes and attributes extracted with position-based extraction method of type *Boolean* are encoded as 1, if the attribute value is True and 0, if attribute value is False.
- meta-information features are numerical and are encoded as they are;
- trace-attributes and attributes extracted with position-based extraction method of type *String* are encoded into numerical features by using the one-hot encoding [2].

[2]One-hot encoding is a method to transform categorical data into a set of numerical features. For each unique value the categorical feature assumes, a new binary variable is created. Each binary variable is "1", if the original categorical value corresponds to the new binary variable and "0" otherwise. e.g. categorical feature *species* with unique values of "dog" and "cat", is transformed into two features *species:dog* and *species:cat*, if the original observation was *species=dog*, then new feature *species:dog = 1* and *species:cat = 0*.

5.2.3 Data-aware Declare constraints

The second method to make use of data-attributes leverages Data-aware DECLARE constraints described in Section 3.2.3. It is often the case, that the DECLARE constraints do not perfectly discriminate between deviant and non-deviant cases, therefore what is done here is to try and increase the discriminative capabilities by conditioning the constraints on data. The pipeline for data-aware constraints starts by discovering DECLARE constraints as discussed in Section 5.1.1, but on a more limited set of templates, where only one parameter acts as an activation (e.g. templates shown in Table 3). The next step is to select discriminating constraints by applying *coverage* feature selection described in Section 5.1.2. The last step focuses on increasing the discriminative capabilities of the selected DECLARE constraints by adding data conditions. The resulting Data-aware DECLARE constraints are then used as new features.

Discovering data-condition. For each selected DECLARE constraint, we create a data-aware constraint with the following procedure:

1. Collect the fulfilled activations for each trace;
2. Extract the data snapshot at every fulfilled activation;
3. Divide data snapshots into two sets:
    - Positive sample set, if the data snapshot belongs to an activation occurring in a deviant trace;
    - Negative sample set, if the data snapshot belongs to an activation occurring in a non-deviant trace;
4. Encode samples into numeric vectors;
5. Learn a decision tree by using numeric vectors based on data snapshots of positive and negative samples;
6. Create data-aware DECLARE constraint by adding data condition to the input constraint.

Data snapshot extraction procedure. Data snapshot is a collection of attributes. To extract data snapshot given a trace and an activation, the following is done:

1. Initialize the snapshot with trace attributes;
2. For each event in the trace (in the order of appearance) until the given activation is found:
    (a) For each event attribute:
        i. If attribute exists in current state of data snapshot then update the snapshot's attribute value to correspond to the current one;
        ii. If attribute does not exist in the data snapshot then add the attribute with the corresponding value to the snapshot;

(b) Return the current state of data snapshot.

To give an example of extraction of a data snapshot, consider the following set of trace-attributes *{cost=10, color=yellow}* with the following trace $\langle \mathsf{a}\{\mathit{color} = \mathit{white}\}, \mathsf{a}\{\mathit{color} = \mathit{black}\}, \mathsf{b}\{\mathit{color} = \mathit{white}\}, \mathsf{c}\{\mathit{cost} = 5\}\rangle$. Having a constraint `response`(a,b), the activations in this trace would be the first activity a at the first position and second activity a at the second position. Given the initial set of trace-attributes, the trace is traversed, encountering the first event $\mathsf{a}\{\mathit{color} = \mathit{white}\}$, the state of attributes is updated to *{cost=10, color=white}* and will be the resulting data snapshot for the first activation. Encountering the second activity $\mathsf{a}\{\mathit{color} = \mathit{black}\}$, the state of attributes is updated to *{cost=10, color=black}* for the second attribute and will be the resulting data snapshot for the second activation.

Encoding data-snapshots into numeric vectors. To train a decision tree classifier there is a need to turn data snapshots into numeric vectors. For each fulfilled activation, a numeric feature vector of attribute features is built, by following the procedure:

- Each attribute in the activation data snapshot is encoded as the value corresponding to the attribute value in the snapshot (categorical values are one-hot encoded as was done for *Pure data* method);
- The encoded vector has a positive label, if the activation belonged to a deviant trace, and a negative label, if it belonged to a normal trace.

Creation of a data-aware declare constraint. Using the set of positive and negative samples generated by encoding data-snapshots into numeric vectors, a decision tree is trained. The trained decision tree will be the data-condition for a new data-aware DECLARE constraint, where the initial DECLARE constraint forms the declare part of the constraint. For example, if the initial constraint is `response`(a,b) and the trained decision tree is T, then the new data-aware DECLARE constraint is `response`(a,b,T). Given the decision tree, an activity a is an activation according to the new constraint `response`(a,b,T), if its data snapshot encoded to a numeric vector, given as an input to decision tree T, classifies as positive.

Encoding trace as a numeric vector using Data-aware declare constraint. Traces are encoded as numeric vectors by checking whether the data-aware constraints extracted from previous teps are satisfied within traces. The trace encoding scheme using a data-aware DECLARE constraint is similar as was for DECLARE constraint, which was described in Section 5.1.3. The difference is in that an activity is considered to be an activation, only, if it follows the data condition.

Each trace in the log is transformed into a numeric feature vector as follows:

- a feature has value -1, if the corresponding data-aware DECLARE constraint is violated in the trace;
- a feature has value 0, if the corresponding data-aware DECLARE constraint is vacuously satisfied in the trace;
- a feature has value n, if the corresponding data-aware DECLARE constraint is satisfied and activated n times in the trace.

For instance, given trace $T = \langle \mathsf{a}\textit{\{color:white\}}, \mathsf{b}, \mathsf{c}, \mathsf{a}, \mathsf{b}\textit{\{color:black\}}, \mathsf{c}, \mathsf{d}, \mathsf{a}, \mathsf{b}\rangle$, with no initial trace attributes (data snapshot is empty at start) and a decision tree T, which has one path, with a rule equivalent of *IF (color=white) => 1*, leading to a positive label, then:

- constraint Response(a,c,T) is satisfied and activated once, and is hence encoded as 1. First (no attribute *color* within snapshot) and third a (snapshot has attribute *color:black*) do not follow the data condition and are therefore not activations;
- constraint Response(a,e,T) is unsatisfied, due to first a not eventually being followed by e and is encoded as -1;
- constraint Response(d,f,T) is satisfied vacuously and encoded as 0 (d is not an activation, because the data snapshot for it is *color:black*).

With a data-aware DECLARE constraint, some traces, for which the original DECLARE constraint was not satisfied, can end up to be vacuously or non-vacuously satisfied. Indeed, if all previously violated activations, do not satisfy the data-condition on the activation, while at least one fulfilled activation satisfies the data-condition, the constraint ends up being satisfied for the trace. If, instead, all the (previous activations) do not satisfy the data condition, the constraint ends up being vacuously satisfied for the trace.

5.2.4 Application

In Section 5.1.6, by applying the process deviance mining pipeline with hybrid encodings, analysts found that the deviant cases are the ones for which:

- the pattern $\langle \mathsf{ER}, \mathsf{RCR}, \mathsf{NC}\rangle$ was repeated more than 2 times; or
- the activity RSR is eventually followed by DR.

Let us assume now that differently from from the previous scenario in Section 5.1.6 the discerned patterns are not able to completely separate deviant from non-deviant cases.

Analysts decide to try and add additional features based on data-attributes. After running the pipeline with the addition of data-based features, analysts found as first cause of deviance, the same as they found without data: pattern $\langle \mathsf{ER}, \mathsf{RCR}, \mathsf{NC}\rangle$ was repeated more than 2 times. With addition *Pure data* features, analysts found that process executions, where customer service language was not English, took more time, i.e. are the deviant cases. When adding features from data-aware DECLARE constraints, analysts found that data-aware DECLARE constraint response(RSR,DR, *resource=D*) characterizes a deviant case better than the initial DECLARE constraint without the data condition, which states that if RSR was performed by a specific resource group *D* and it was eventually followed by DR, the execution times were longer.

Compared to only considering control-flow features, the addition of features based on data, helped analysts to enrich and refine the deviance causes previously identified. In total analysts were able to find 3 situations, where process executions take considerably longer:

- when the customer service language is not English. This can be due to the lack of specific language speaking customer service agents;
- when a customer support operators belonging to a specific resource group (resource group D) procrastinate writing reports related to easy request resolutions;
- when, in a case of a difficult request, more than 2 iterations are carried out with the customer.

5.3 Extraction of Results of Deviance Mining

It is important to provide analysts a *good* set of rules, describing the differences between deviant and non-deviant traces. One way get the characterizing rules is to extract decision rules from a 'white-box' classifier like a decision tree, for which the extraction process was described in Section 5.1.5. Extraction of rules from a decision tree is an indirect method of rule extraction, as it first requires an already trained classifier. There are also direct methods, which extract rules direct from the data. One of the direct methods is RIPPER, which was introduced in Section 3.3.2.

5.3.1 Extraction of Rules

The approach for building a decision tree and extracting rules is the same as was described in Section 5.1.

For RIPPER, the feature discovery, selection and encoding parts remain exactly the same. In the step of model training, instead of training a decision tree classifier, the encodings are given as an input to RIPPER. As a result of RIPPER a set of decision rules is got as an output.

6 Evaluation

In this section evaluations are provided for each of the previously described experimentation approaches. All the code and results for the thesis can be found at: `https://github.com/Abercus/deviancеminingthesis`. The full set of results can be found within the same repository in *ExperimentResults* folder, where also included are results of measuring the metrics on training data and standard deviations for each of the metrics.

For decision trees used in the experiments, the implementation *DecisionTreeClassifier* from *scikit-learn* [PVG+11] was used. As an implementation for RIPPER, *JRip* within Weka software [HFH+09] was chosen.

6.1 Exploring Business Process Deviance with Sequential and Declarative Patterns

In order to investigate the impact of different types of features (sequential, declarative and hybrid) on their ability to accurately discriminate between non-deviant and deviant executions of a process, an extensive evaluation was carried out. In detail the following research questions were considered:

RQ 1. What is the best (combination of) feature type(s) on synthetic logs generated starting from a procedural model and labeled based on control flow information?
RQ 2. What is the best (combination of) feature type(s) on synthetic logs generated starting from a declarative model and labeled based on control flow information?
RQ 3. What is the best (combination of) feature type(s) on real-life logs labeled based on information not related to control flow?
RQ 4. What is the best (combination of) feature type(s) on real-life logs labeled based on control flow information?

RQ1 evaluates different types of features on synthetic logs generated starting from a procedural model and labeled using conditions on the presence of sequential patterns in each trace. Symmetrically, **RQ2** aims at evaluating the different types of features on synthetic logs in which a declarative model has been used for generating the log and declarative constraints used for defining deviant and non-deviant cases. **RQ3** and **RQ4**, differently from the previous ones, focus on real-life logs in which deviant and non-deviant cases have been defined based on information not related to control flow and on the presence of a control flow pattern.

6.1.1 Datasets

The evaluation has been carried out using 3 different synthetic datasets[3] and 7 real-life datasets.

The first one of the 3 synthetic datesets, the X-ray dataset (*Synth*$_{XRAY}$), has been generated starting from a declarative model by using the MP-Declare Log Generator [SDFGM18]. Moreover, the disjunction of three Declare constraints has been used to label each case of the log as non-deviant (if at least one of three constraints is satisfied) or as deviant (if all the three of them are violated).

The second ($Synth_{MR/TR}$) and the third ($Synth_{MRA/TRA}$) synthetic datasets were instead generated starting from a BPMN procedural model by using the procedural log generator PLG2 [Bur16]. The datasets were labeled using procedural constraints. For $Synth_{MR/TR}$, a trace was labeled as deviant, if a sequence of events (abc) occurred at least twice exactly in the same order (MR/TR condition). For $Synth_{MRA/TRA}$, a trace was labeled as deviant, if a sequence of events occurred 3 times in the log, but this time the requirement of being in specific order was not required (MRA/TRA condition), e.g., abc, bca.

Concerning the real-life event logs, the BPI Challenge 2011 [Van11] and the Sepsis Cases event logs [Man16] were used. The BPI Challenge 2011 log contains data about a Dutch Academic Hospital. Each case represents the clinical history of a patient and events in the log are enriched with data attributes describing diagnosis and treatment received by the patient. For this log 4 different labeling criteria based on attribute values (provided in [NDLR$^+$14]) were used:

- case attribute *Diagnosis* is "cervix cancer" (*BPIC2011*$_{dCC}$);
- case attribute *Treatment code* is "101" (*BPIC2011*$_{t101}$);
- case attribute *Diagnosis code* is "M13" (*BPIC2011*$_{m13}$);
- case attribute *Diagnosis code* is "M16" (*BPIC2011*$_{m16}$).

The Sepsis event log collects cases of patients with symptoms of a sepsis condition from a Dutch hospital. The Sepsis log was labeled based on 3 different criteria: one based on procedural patterns (*Sepsis*$_{Proc}$), one based on declarative patterns (*Sepsis*$_{Decl}$) and a third one based on temporal aspects (*Sepsis*$_{ER}$)

- event sequence *NC-Leucocytes-CRP* exists in trace (*Sepsis*$_{Proc}$);
- constraints *Response:(IV Antibiotics, Leucocytes)*, *Response:(LacticAcid, IV Antibiotics)* and *Response:(ER Triage, CRP)* are non-vacuously fulfilled at the same time (*Sepsis*$_{Decl}$);
- the patient returned to emergency room within 28 days after being released (*Sepsis*$_{ER}$).

[3]The models used for generating the synthetic event logs and the criteria used for labeling them in terms of deviant or non-deviant cases are available at `https://www.dropbox.com/s/wbx9vjlzdro00ej/devianceMining2019Models.zip?dl=0`.

Table 4. Overview of the datasets used in the evaluation

Dataset	**Number of cases**	**Number of deviant cases**	**Number of non-deviant cases**	**Avg. trace length**	**Labeling type**
$Synth_{XRAY}$	1000	150	850	17	Declarative
$Synth_{MR/TR}$	2517	839	1678	12	Procedural
$Synth_{MRA/TRA}$	1452	484	968	19	Procedural
$BPIC2011_{dCC}$	1143	277	866	131	Attribute
$BPIC2011_{t101}$	1143	368	775	131	Attribute
$BPIC2011_{m13}$	1143	235	908	131	Attribute
$BPIC2011_{m16}$	1143	476	667	131	Attribute
$Sepsis_{ER}$	1050	113	937	14	Temporal
$Sepsis_{Proc}$	1050	318	732	14	Procedural
$Sepsis_{Decl}$	1050	365	685	14	Declarative

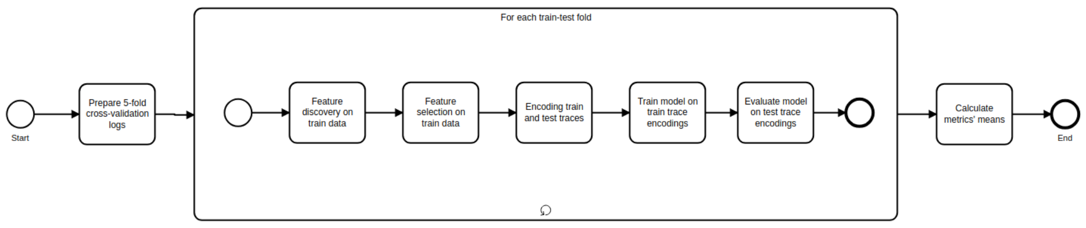

Start
Prepare 5-fold cross-validation logs
For each train-test fold
Feature discovery on train data
Feature selection on train data
Encoding train and test traces
Train model on train trace encodings
Evaluate model on test trace encodings
Calculate metrics' means
End

Figure 9. Cross-validation pipeline

Table 4 provides an overview of the size of all the datasets used in the evaluation. For each event log and each labeling criteria, we report the total number of cases in the log, the number of deviant and non-deviant cases based on the labeling criteria, the average trace length and the type of labeling used, i.e., declarative, procedural, temporal or attribute-based.

6.1.2 Procedure

In order to evaluate and compare the different feature types on their ability to discriminate between deviant and non-deviant cases, we measured and compared the performance of a classifier, trained with labeled data encoded based on different feature types, in classifying new unseen execution traces as deviant or non deviant. The encodings we used are Individual Activities (IA), Declare, IA+TR, IA+TRA, IA+MR, IA+MRA and Hybrid. Figure 9 summarizes the procedure carried out. For each event log and labeling criteria, we carried out a 5-fold cross-validation [4] on the labeled event log so that, for each step of the cross-validation, for each encoding and for each coverage thresholds (used are 5, 15 and 25), we apply the following procedure:

1. event log is split into two sets: 80% training data and 20% testing data;

[4]cross-validation is a technique for assessing how classifiers perform on previously unseen samples.

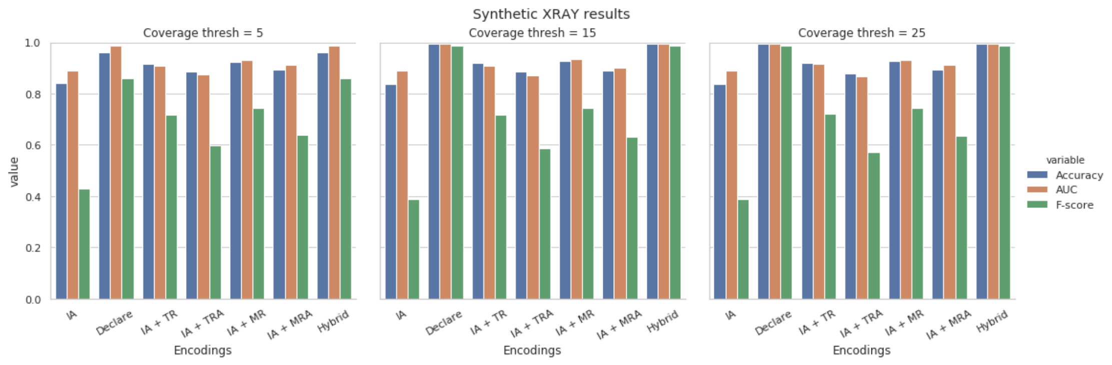


Figure 10. Synthetic dataset X-ray results

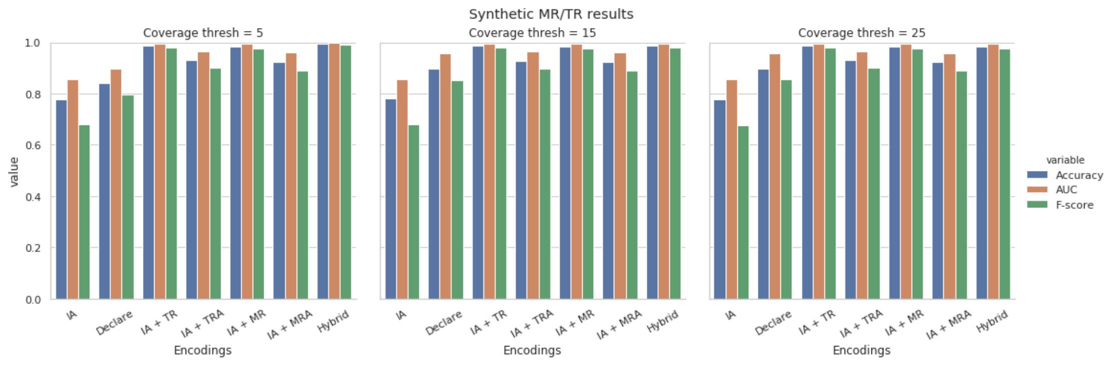
2. features are discovered and selected (for selection coverage method was used) on training data;
3. training data and test data is encoded according to the feature type (for declare the original encoding scheme was used);
4. classifier is trained with (the decision tree) encoded training data;
5. three commonly used metrics aiming at evaluating of the classifier on the testing data are computed.

At the end, for each of the three metrics, the average of the 5-fold iterations is computed.

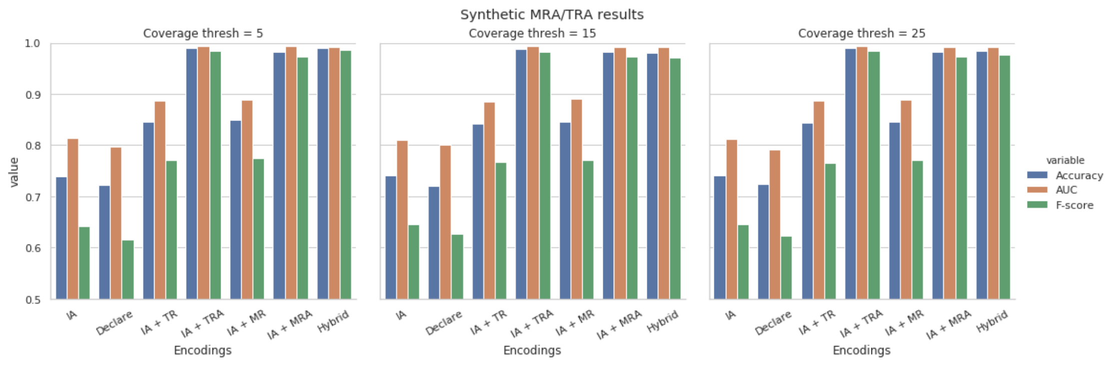
In detail, we measured accuracy, AUC (Area Under The Curve) ROC (Receiving Operating Characteristics) curve and F1 score. For accuracy the following standard formula was used $\frac{tp+tn}{tp+tn+fp+fn}$, where *tp* is the number of true positives, *tn* is the number of true negatives and *fp* and *fn* respectively stand for false positives and false negatives. F1-score, which is the harmonic mean of precision and recall, is defined as $F_1 = 2 \cdot \frac{precision \cdot recall}{precision+recall}$, where recall is defined as $recall = \frac{tp}{tp+fn}$ and precision as $precision = \frac{tp}{tp+fp}$. Precision corresponds to the fraction of all positively classified samples, for which the ground truth is also positive. Recall corresponds to the fraction of how many positive samples were classified as positive over the set of all positive samples. AUC ROC curve, when using normalized units, is equal to the probability that the classifier ranks randomly chosen positive observation higher than a randomly chosen negative observation [Faw06].

6.1.3 Results

Figure 10 plots the results related to the experiments carried out on the $Synth_{XRAY}$ dataset, i.e., the dataset built and labeled starting from a declarative specification, with different (combinations of) feature types and for different coverage threshold values. The plot shows that, as expected, the declarative and the hybrid features overcome the individual activity and the sequential encodings on all the three metrics and for all



(a) Synthetic MR/TR dataset results



(b) Synthetic MRA/TRA dataset results

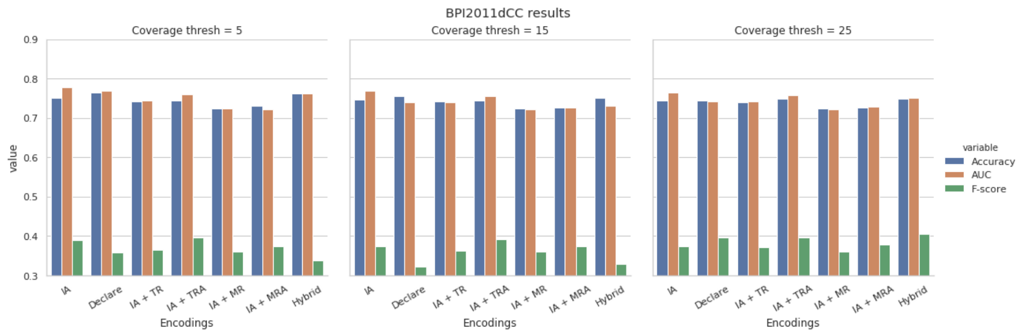
Figure 11. Synthetic procedural dataset results

the coverage threshold values. This analysis suggests that for "declarative" event logs, i.e., event logs generated from a declarative model and in which deviant cases can be described in terms of non-sequential patterns, the best types of features to be used for encoding traces are those belonging to the declarative family or including also declarative patterns, i.e., hybrid ones (**RQ1**). Moreover, by inspecting the plot, we can also observe that for higher coverage threshold values, i.e., 15 and 25, we overall get more accurate results.

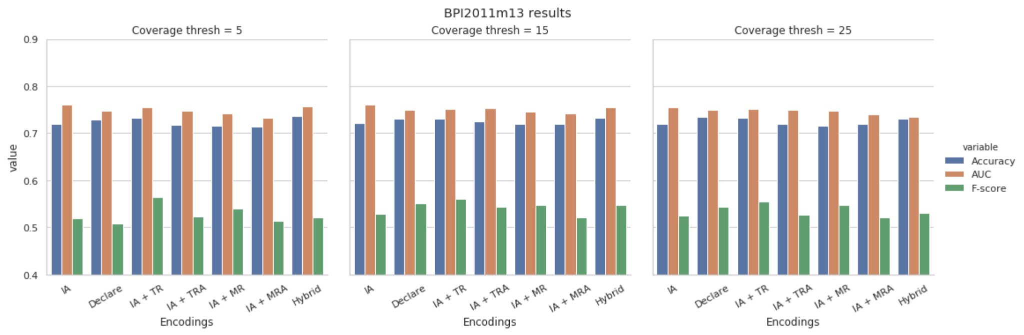
Figure 11a and 11b report the plot of the classification results related to the $Synth_{MR/TR}$ and the $Synth_{MRA/TRA}$ dataset, respectively, i.e., the two synthetic datasets built and labeled starting from a procedural model. As expected, in both cases the sequential and the hybrid approaches outperform the IA and the declarative ones. In detail, in the $Synth_{MR/TR}$, all the sequential features (IA+TR, IA+TRA, IA+MR and IA+MRA) are able to achieve very good results (with accuracy and F1-measure close to 1) for all the coverage threshold values. For $Synth_{MRA/TRA}$, instead, the best results are achieved by the alphabet-based (and the hybrid) feature types. This is reasonable by taking into account how the dataset has been built. It is important to note that also with procedural-generated and labeled synthetic event logs, hybrid feature type is able to best discriminate between deviant and non-deviant cases (**RQ2**).

Figure 12 reports the plots related to the *BPI2011* dataset, where an attribute-based labeling had been applied. For *BPI2011*$_{dcc}$ (Fig. 12a) all the accuracy and AUC measures were between 0.7-0.8, but F1-measure was close to or under 0.4, which indicates that labeling does not necessarily depend on the control-flow. The best results for *BPI2011*$_{dcc}$ results were achieved by using hybrid encoding with coverage threshold of 25. Individually declarative and IA+TRA with threshold of 25 had the best results. For log *BPI2011*$_{m13}$ (Fig. 12b) sequential encoding IA + TR with coverage threshold 15 gave the best results, with hybrid encoding with same coverage close to it. In results for log *BPI2011*$_{m16}$ (Fig. 12c), the results using all encodings were pretty close, with IA+MRA and Declare encodings with a coverage threshold 25 giving best results and the results for hybrid encoding closely behind. For log *BPI2011*$_{t101}$ (Fig. 12d), the deviance cause was grasped by hybrid and declarative encodings. With accuracy around 0.84, AUC 0.87 and F1-measure around 0.86, it suggests that for *BPI2011*$_{t101}$ the attribute labeling is pretty well explainable by the use of control-flow features. Overall, however, we can observe that with attribute-based labelling on real-life logs the hybrid encoding mostly provides the best or close to the best results compared with other encodings (**RQ3**).

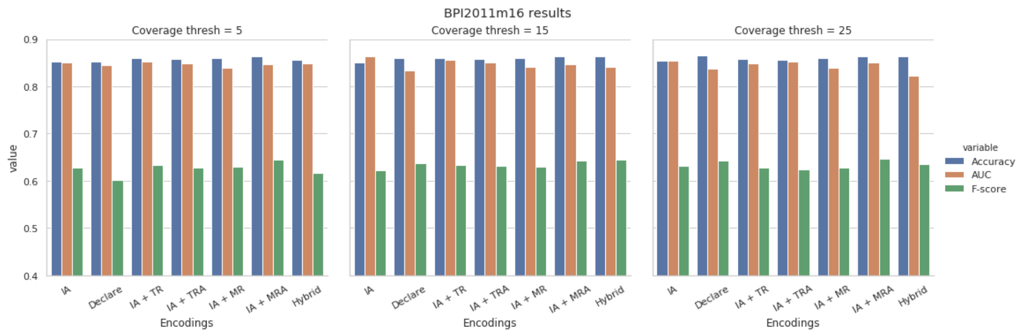
Finally, Figure 13 reports the plot with the results related to the *Sepsis* dataset labeled with procedural, declarative and temporal labeling. The figures show that with a sequential labeling, the sequential and hybrid features work better than declarative. Viceversa, with a declarative labeling declarative and hybrid features perform better than sequential. When, instead, the log is labeled with temporal conditions, there is no clear trend. However, on all labelings, for at least one of the coverage thresholds, the models built using the hybrid encoding were able to perform as well as the models built with the best one of the other encodings (**RQ4**).



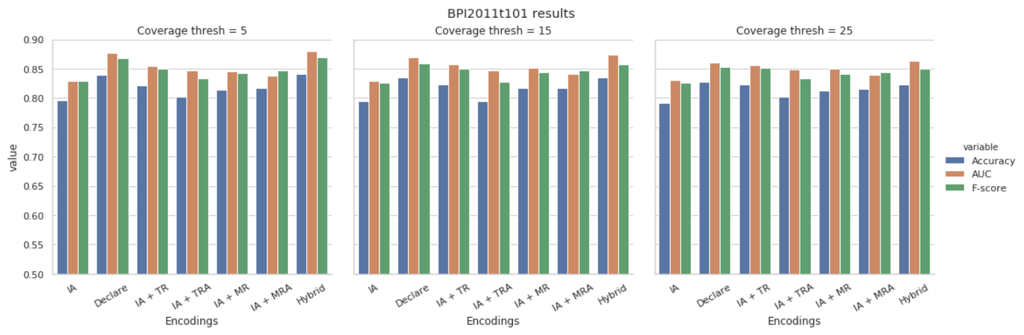
(a) Results of experiments on $BPI2011_{dCC}$ dataset.



(b) Results of experiments on $BPI2011_{m13}$ dataset

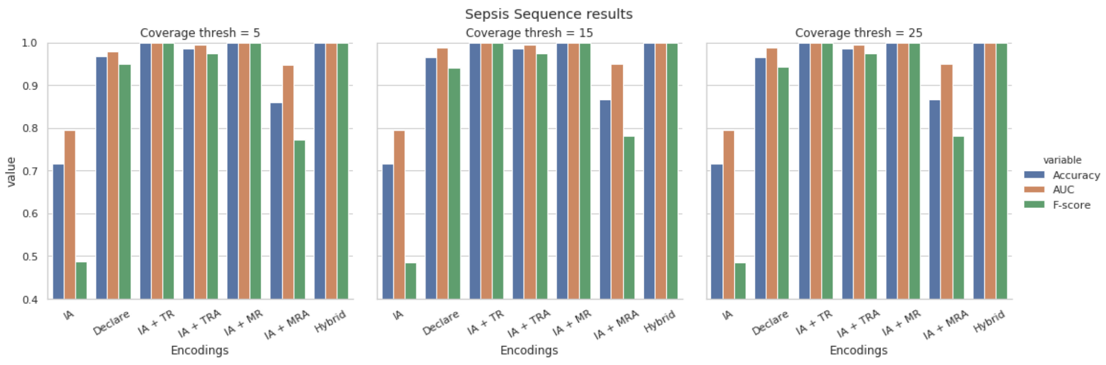


(c) Results of experiments on $BPI2011_{m16}$ dataset.

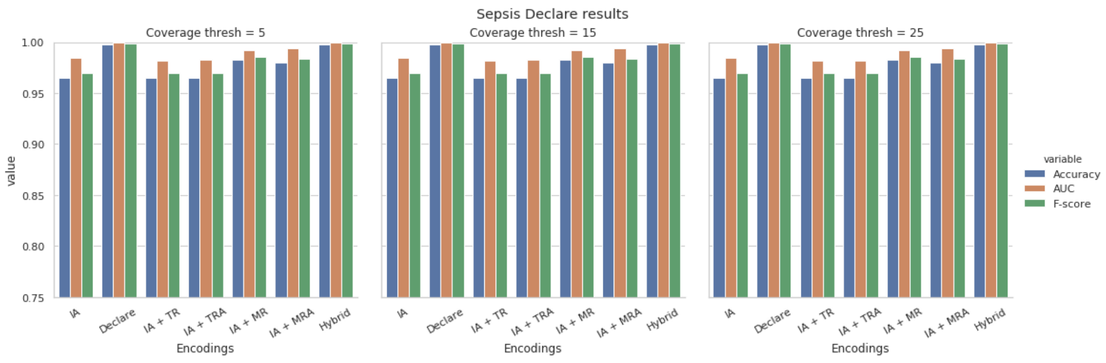


(d) Results of experiments on $BPI2011_{t101}$ dataset.

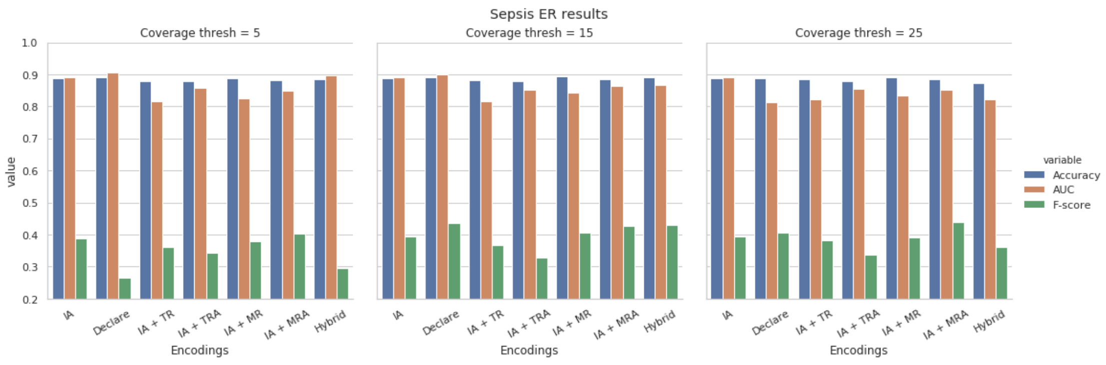
Figure 12. BPI2011 dataset results



(a) Results of experiments on $Sepsis_{Proc}$ dataset



(b) Results of experiments on $Sepsis_{Decl}$ dataset



(c) Results of experiments on $Sepsis_{ER}$ dataset

Figure 13. Sepsis dataset results

6.2 Exploring Payload Features for Business Process Deviance Mining

In order to explore whether the approaches for extracting and encoding data-attributes improve classifier's performance, a set of evaluations was performed and performance measures collected before and after the addition of features based on payload.

RQ 5. Does adding payload features improve the performance of deviance mining with sequential encodings on real-life logs?
RQ 6. Does adding payload features improve the performance of deviance mining with declarative encodings on real life logs?
RQ 7. Does adding payload features improve the performance of deviance mining with declarative encodings on real life logs?
RQ 8. What is the best combination of encodings amongst sequential (TR, MR, MRA, TRA), Declare and Hybrid with added payload features?

6.2.1 Datasets

The event logs used for experimentation on payload features were the same as in Section 6.1.1, but with synthetic event logs excluded. Synthetic logs are left out, because those logs do not contain informative data payload for investigation. The logs were *BPIC2011*$_{dCC}$, *BPIC2011*$_{m13}$, *BPIC2011*$_{m16}$, *BPIC2011*$_{t101}$ and *Sepsis*$_{ER}$.

Before extracting payload features from the event logs, all data-attributes used in the process of labeling the traces were removed. For *BPI2011* datasets this includes the removal of all *Diagnosis*, *Diagnosis code*, *Treatment code* and similar attributes, based on which the traces were labeled on. Since *Sepsis*$_{ER}$ was labeled on a temporal condition, it did not require the removal of any data-attributes.

6.2.2 Procedure

The procedure carried out to see, if the addition of features based on *Pure data* or data-aware DECLARE constraints improves the capability of discriminating between deviant and non-deviant cases, is done similarly as was done in Section 6.1.2. The main difference comes from the addition of features based on data-attributes. For experimenting with *Pure data* features, the following encodings were used Individual Activities (IA) + Pure Data (PD), Declare+PD, IA+TR+PD, IA+TRA+PD, IA+MR+PD, IA+MRA+PD, Hybrid+PD. For experimenting with features on Data-aware DECLARE constraints (DAC) the encodings used were Declare+DAC, Hybrid+DAC. Additionally, experiments were carried out by adding both payload feature types and using a combination of the encodings: Declare+PD+DAC and Hybrid+PD+DAC. For the results obtained without using data-payload attributes, used for comparison purposes, the same set of experiments was carried out using encodings: IA, Declare, IA+TR, IA+TRA, IA+MR, IA+MRA and Hybrid.

Choice of attributes. In the feature extraction phase, attributes *concept:name*, *lifecycle:transition* and *Label* were ignored. The attribute *concept:name* was ignored due to each trace having a unique name and using *concept:name* would be the same as using encoding IA as it is the name of an activity. The attribute *Label* was ignored, because it is a standard attribute used for naming traces and events. Missing values were handled in both logs by setting missing *String* value as "missing", *Int* and *Float* values as "0" and *Boolean* as "false".

Pure data extraction methods. The following *Pure data* payload extraction methods (see Section 5.2.2) were used for *BPI2011* logs:

- on event attributes *org:group*, *org:section*, *Producer code* and *Specialism code* aggregated features were created by using method *Count*;
- the position-based method *Choose first* was used for all other event attributes (encodings depend on the attribute type);
- trace attributes, which were not ignored (see above in *Choice of attributes*), were extracted as they were (e.g age) and the way of encoding depended on the attribute type;
- a meta-information feature was created by using the method *Length*.

For $Sepsis_{ER}$ the following *Pure data* feature extraction methods were used:

- on event attribute *org:group* an aggregated feature was created by using method *Count*;
- position-based method *Choose first* was used for all other event attributes (encodings depend on the attribute type);
- trace attributes, which were not ignored (see above in *Choice of attributes*), were extracted as they were and the way of encoding depended on the attribute type;
- a meta-information feature was created by using the method *Length*.

Data-aware declare constraint feature extraction. Creating features using data-aware DECLARE constraints does not require as much configuration as when using features based on *Pure data*. What suffices is the set of ignored attributes as given previously in *Choice of attributes*.

As done in Section 6.1.2, a 5-fold cross validation on the labeled logs was carried out and the same set of metrics were computed.

6.2.3 Results

Following, we report the results on experiments with the addition of data payload features.

Sequential encoding.

The plots in Figures 14b, 15b, 16b, 17b and 18b report accuracy, AUC and F1 measures for the *BPI2011*$_{dCC}$, *BPI2011*$_{m13}$, *BPI2011*$_{dCC}$, *BPI2011*$_{m16}$, *BPI2011*$_{t101}$ and *Sepsis*$_{ER}$, on sequential features with and without the features of the *Pure data* payload, respectively.

Results plotted in Figure 14a show that with dataset *BPI2011*$_{dCC}$, the addition of *Pure data* (PD) features to sequential encodings brings a considerable increase in accuracy, AUC and F1 metrics. The same can also be seen to happen in Figure 15a for dataset *BPI2011*$_{m13}$ and in Figure 17a for *BPI2011*$_{t101}$.

For *BPI2011*$_{m16}$ dataset (Figure 16a), the addition of *Pure data* to sequential encodings did not have much of effect as for the other $BPI2011$ logs. However looking at a single metric at a time, it can be seen that the features leading to the best F1-measure were IA+TR+PD (coverages 15 and 25), for AUC metric the best combination was IA+MR+PD (coverage 5) and the best for the accuracy was IA+TR+PD (coverages 15 and 25).

On *Sepsis*$_{ER}$ log (Figure 18a), the addition of *Pure data* to sequential encodings, brought a small increase in AUC, but a decrease in F1-measure.

These experiments show that for some labelings, the addition of *Pure data* features to sequential features, can increase the performance metrics of the deviance mining approach. However, there are also logs, where the addition of data features to sequential brings no or only a small increase in one of the metrics, while a considerable decrease in one or several others. This can be due to a only a small or a lack of correlation between the labeling of a log and the data (**RQ 5**).

Declarative encoding.

The plots in Figures 14b, 15b, 16b, 17b and 18b report accuracy, AUC and F1 measures for the *BPI2011*$_{dCC}$, *BPI2011*$_{m13}$, *BPI2011*$_{dCC}$, *BPI2011*$_{m16}$, *BPI2011*$_{t101}$ and *Sepsis*$_{ER}$, on declarative features with and without the features of the *Pure data*, *Data-aware declare* or the combination, respectively.

For dataset *BPI2011*$_{dCC}$ (Fig. 14b), the highest accuracy and F1-measure was achieved with Declare+PD+DAC (coverage 25). Highest AUC was achieved with Declare+DAC on coverage 15.

On dataset *BPI2011*$_{m13}$ (Fig. 15b), the best results were achieved with Declare+PD to Declare. Declare+DAC and Declare+PD+DAC, also performed better than Declare, with the exception of coverage 5, where Declare had a higher AUC.

Results on *BPI2011*$_{m16}$ (Fig. 16b) are similar to the ones for *BPI2011*$_{m13}$, with Declare+PD giving the best results. Declare+DAC and Declare+PD+DAC also gave better results on all metrics compared to Declare, this time without any exception. For *BPI2011*$_{t101}$ (Fig. 17b), Declare+PD+DAC and Declare+DAC performed the best, with the best achieved with coverage threshold of 15.

The results obtained for *Sepsis*$_{ER}$ (Fig. 18b) for different coverages were not as stable as for *BPI2011* log, features Declare and Declare+PD for coverage 5, resulting in a considerably lower F1 compared to other coverage thresholds. With coverage 15, the best result in accuracy and F1 was with Declare+PD, highest AUC with Declare. On coverage 25, the best results were given with Declare+DAC.

In terms of adding payload features to declarative features, there is no clear answer in what always works the best but is dependent on the log. As an example, Declare+PD+DAC had highest F1-measure for *BPI2011*$_{dCC}$, while the highest AUC was achieve with Declare+DAC. For *BPI2011*$_{m13}$, the best results were achieved with Declare+DAC and for *BPI2011*$_{m13}$ by Declare+PD. In *Sepsis*$_{ER}$, the results were even more unstable, as with some coverage best result being Declare+DAC and in other Declare+PD. However, in most of the experimented cases the the performance of the deviance mining approach improved with at least one of the added payload features. As previously, the lack of improvement from data can be due to a lack of correlation between the data and the labeling (**RQ 6**).

Hybrid encoding.

The plots in Figures 14c, 15c, 16c, 17c and 18c report accuracy, AUC and F1 measures for the *BPI2011*$_{dCC}$, *BPI2011*$_{m13}$, *BPI2011*$_{dCC}$, *BPI2011*$_{m16}$, *BPI2011*$_{t101}$ and *Sepsis*$_{ER}$, on hybrid features with and without the features of the *Pure data*, *Data-aware declare* or the combination, respectively.

With dataset *BPI2011*$_{dCC}$ (Fig. 14c) best results in terms of F-measure were achieved by Hybrid+DAC and Hybrid+PD+DAC (coverages 15 and 25). Highest AUC was achieved with Hybrid+PD with coverage of 25 and best accuracies with Hybrid+Data with coverages 15 and 25.

On dataset *BPI2011*$_{m13}$ (Fig. 15c), highest F-measure, AUC and accuracy were all achieved with Hybrid+PD over all coverages, with coverage 25 giving the best.

Results on *BPI2011*$_{m16}$ (Fig. 16c) again, show results similar to those of *BPI2011*$_{m13}$, with Hybrid+PD giving the best results with all coverages and coverage 25 giving the best.

For *BPI2011*$_{t101}$ (Fig. 17c), Hybrid+DAC and Hybrid+PD+DAC gave the best results in terms of all metrics, with coverage 15 working the best.

On *Sepsis*$_{ER}$ (Fig. 18c), the addition of data payload features to hybrid encodings, did not increase the performance metrics. However, the obtained performance metrics are quite close with and without the data payload encoding. This can be due to a small or no correlation between the data payload and the labeling of a log. The best F1-measure is achieved with a coverage of 25 and with Hybrid, Hybrid+PD, Hybrid+PD+DAC, with Hybrid+DAC close. Best AUC and accuracy were achieved with a coverage of 5 and encodings Hybrid and Hybrid+PD.

From the results of experiments with the payload addition to Hybrid model, on *BPI2011* logs, we can assess that the payload addition increased the performance of the deviance mining approach, while on *Sepsis*$_{ER}$ log the addition of data to hybrid model did not add any extra value, which can be due to a lack of signal from data (**RQ 7**).

Comparison of all encodings.

The plots in Figures 14d, 15d, 16d, 17d and 18d report accuracy, AUC and F1 measures for the *BPI2011*$_{dCC}$, *BPI2011*$_{m13}$, *BPI2011*$_{dCC}$, *BPI2011*$_{m16}$, *BPI2011*$_{t101}$ and *Sepsis*$_{ER}$, on sequential features with added *Pure data* features, declarative and hybrid features with and without the features of the *Pure data*, *Data-aware declare* or the combination, respectively.

With dataset *BPI2011*$_{dCC}$ (Fig. 14d), the best combination of encodings was based on sequential enicoding and , more specifically IA+TR+PD and IA+TRA+PD. Both have a close results in terms of accuracy and AUC. The highest F1 resulted from IA+TRA+PD, with IA+TR+PD close to it.

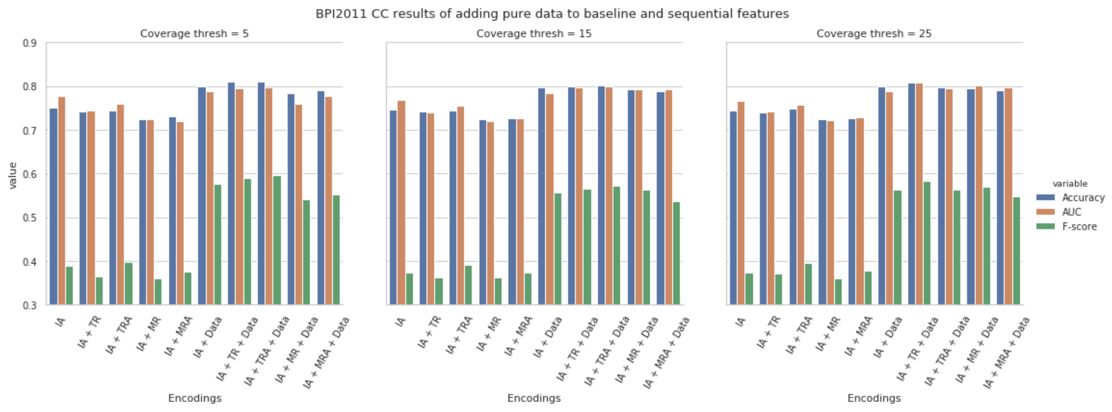
On dataset *BPI2011*$_{m13}$ (Fig. 15d), the best results in terms of all metrics, AUC, F-measure and accuracy, were achieved with Declare+PD and Hybrid+PD.

For dataset *BPI2011*$_{m16}$ (Fig. 16d), the best combination with added payload features, was based on sequential patterns. More specfically IA+TR+PD performed best in terms of accuracy and F1, while IA+MR+PD resulted in the highest AUC.

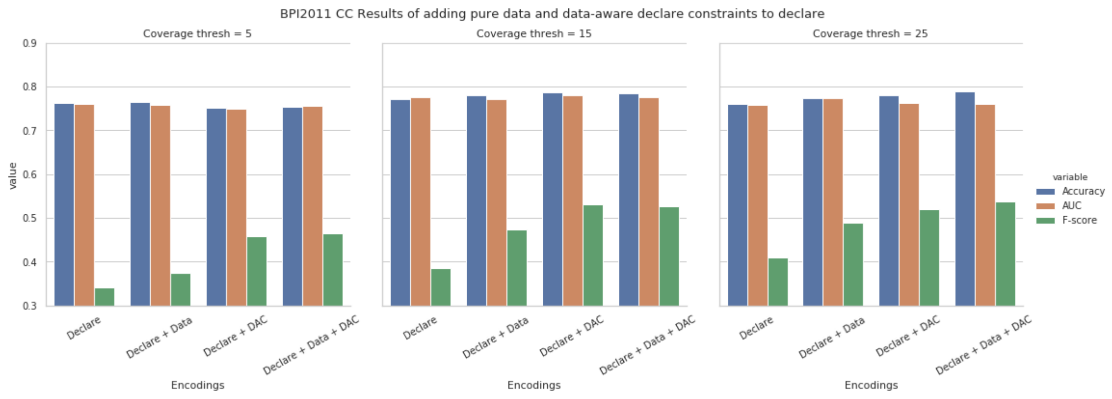
For *BPI2011*$_{t101}$ (Fig. 17d), the encodings based on sequential features together with payload features, got the highest results in all three measures.

On *Sepsis*$_{ER}$ (Fig. 18c), the highest results in terms of F-measure and accuracy, resulted from Declare+PD. Highest accuracy was got as a result of the sequential encoding IA+MR+PD.

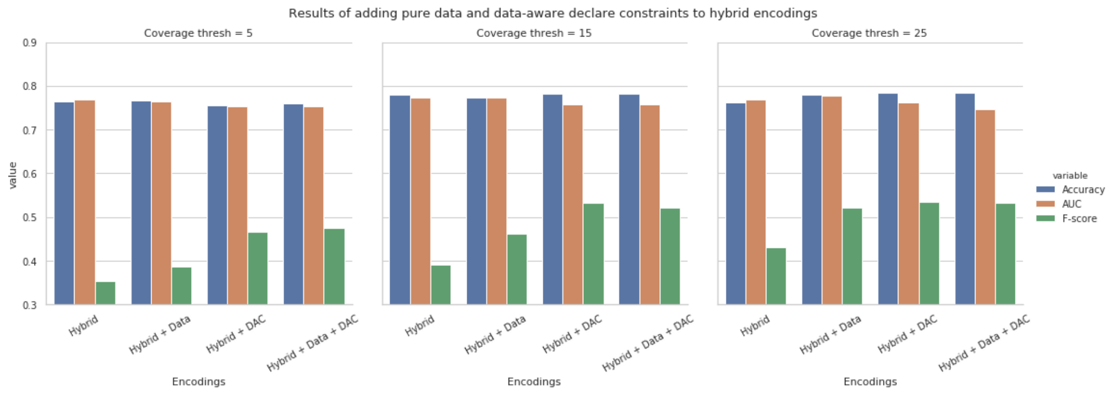
Having the results on all encodings and datasets, there is no clear winner in terms of the best combination of encoding with data payload features. The choice of the best combination varies depending on the dataset, labeling and on the chosen metrics. In general, the impact of the payload is related to the correlation between the data and the labeling of a log (**RQ 8**).



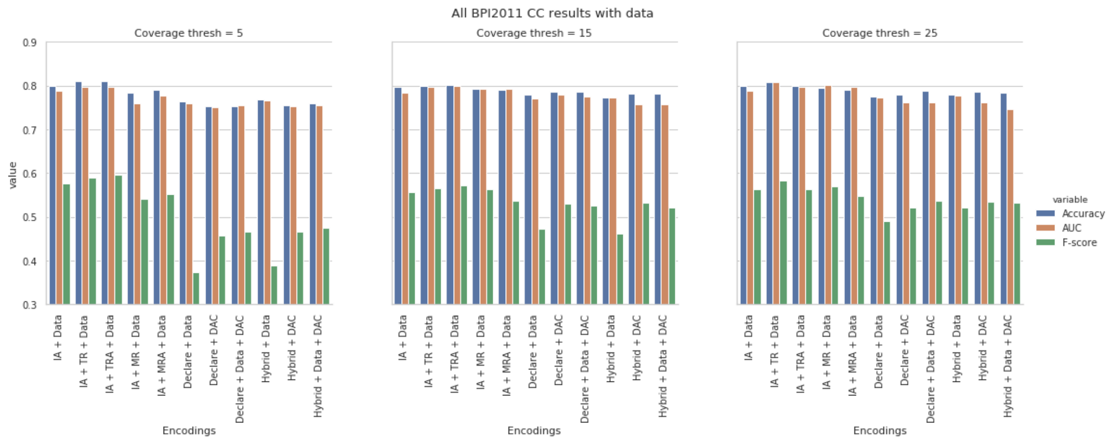
(a) Results with added payload features to baseline and sequential features on *BPI2011*$_{dCC}$ dataset.



(b) Results with added payload to declarative features on *BPI2011*$_{dCC}$ dataset.

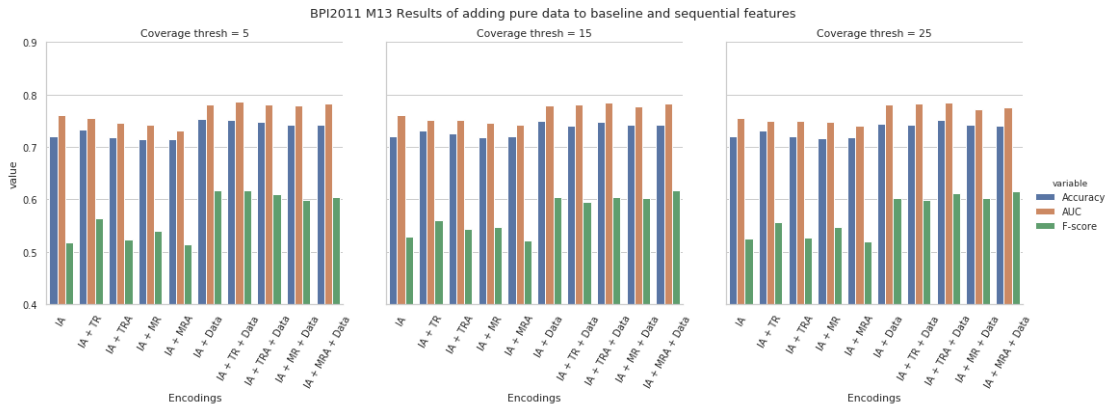


(c) Results with added payload to hybrid model on *BPI2011*$_{dCC}$ dataset.

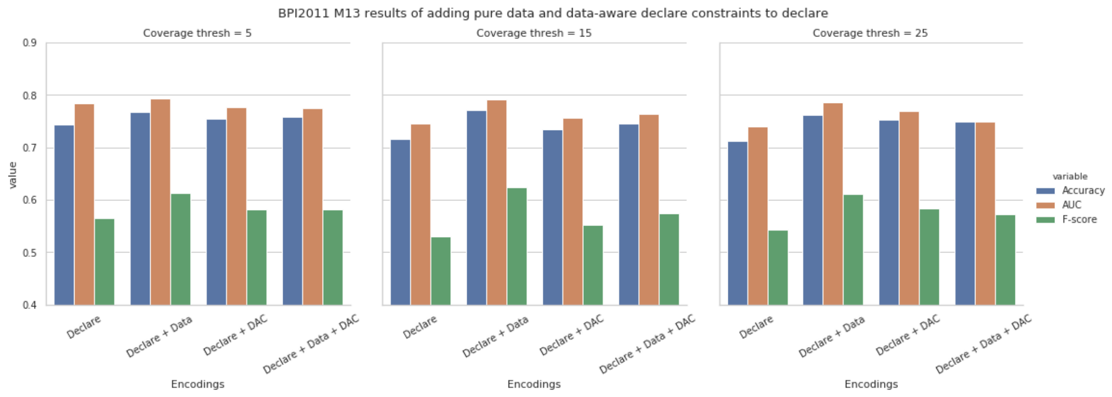


(d) Comparative results of all payload additions on $BPI2011_{dCC}$ dataset.

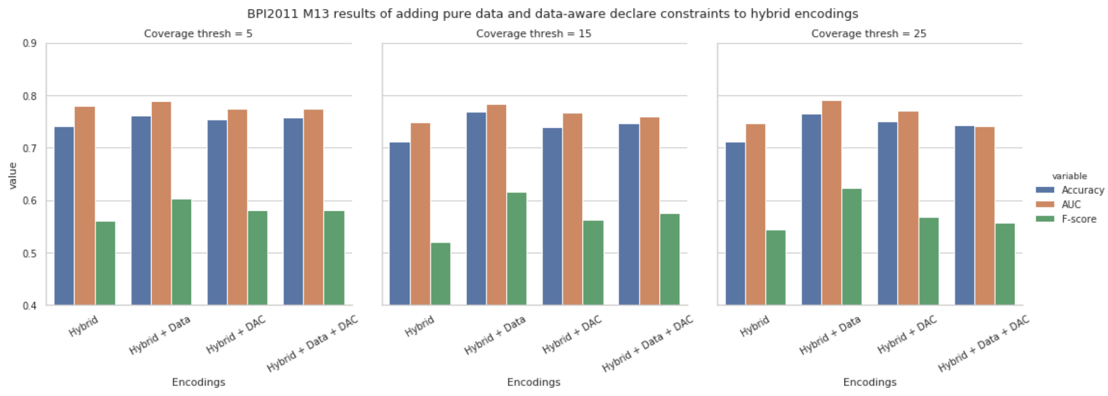
Figure 14. $BPI2011_{dcc}$ dataset payload results



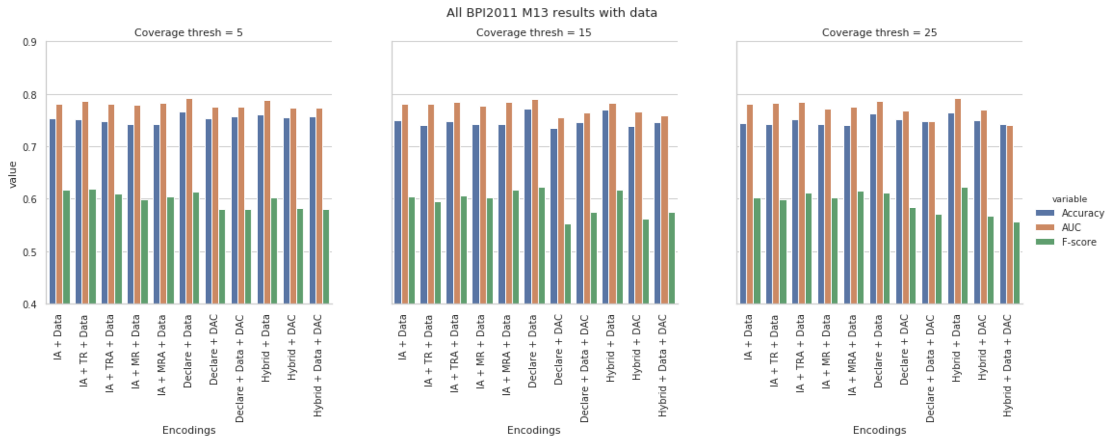
(a) Results with added payload features to baseline and sequential features on $BPI2011_{m13}$ dataset.



(b) Results with added payload to declarative features on $BPI2011_{m13}$ dataset.

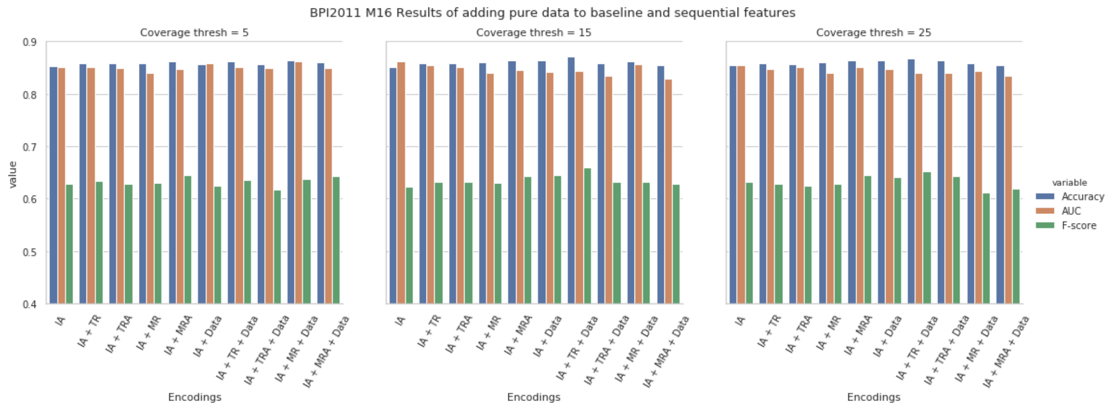


(c) Results with added payload to hybrid model on $BPI2011_{m13}$ dataset.

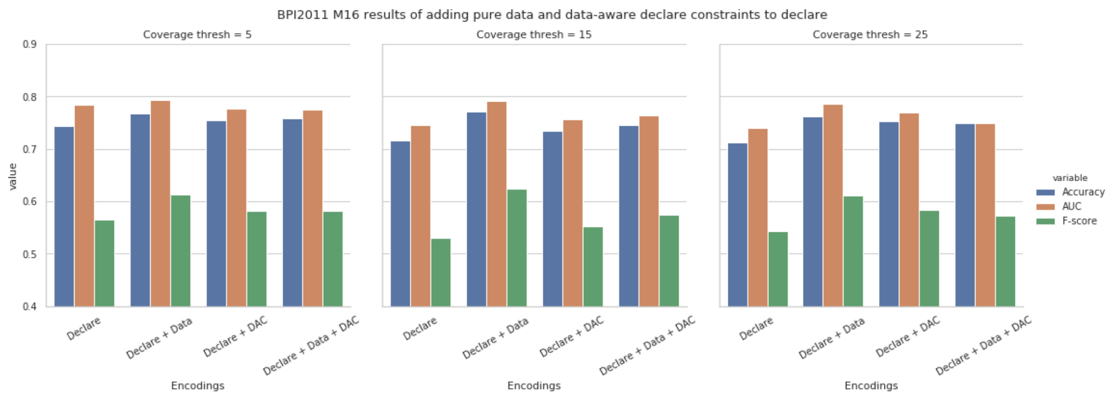


(d) Comparative results of all payload additions on $BPI2011_{m13}$ dataset.

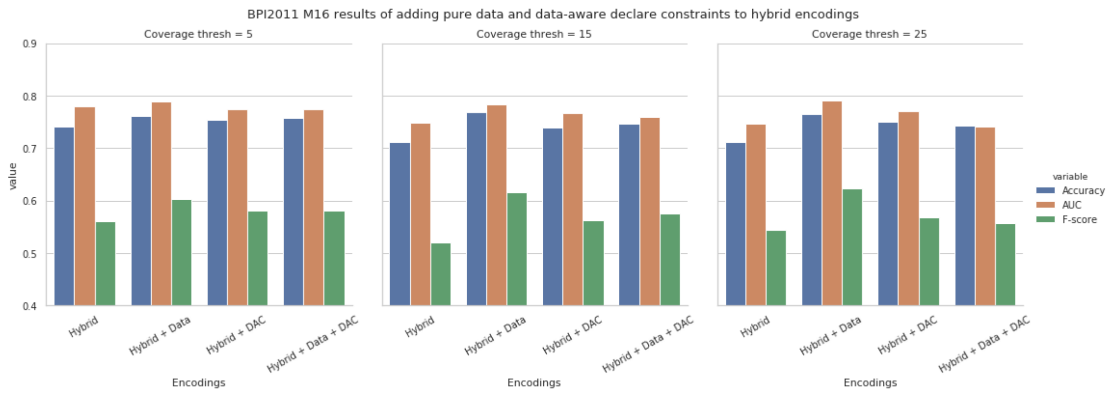
Figure 15. $BPI2011_{m13}$ dataset payload results



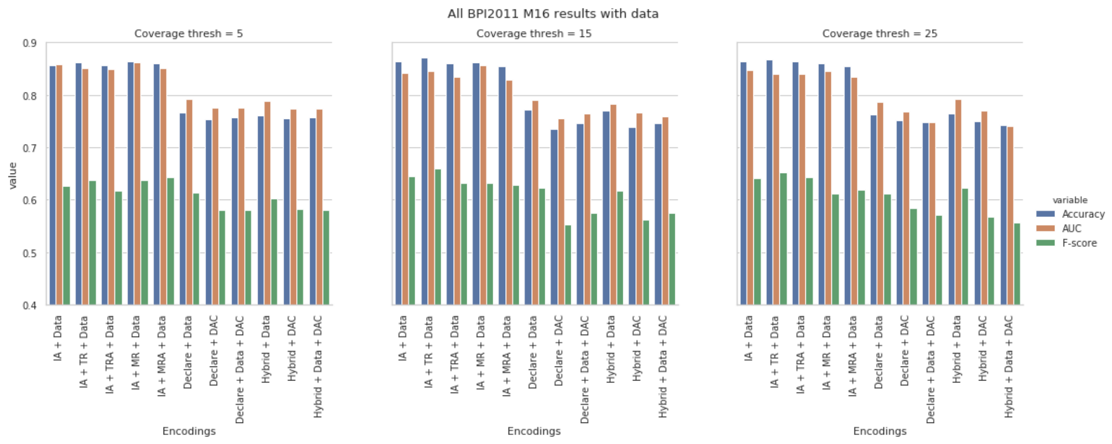
(a) Results with added payload features to baseline and sequential features on $BPI2011_{m16}$ dataset.



(b) Results with added payload to declarative features on *BPI2011*$_{m16}$ dataset.

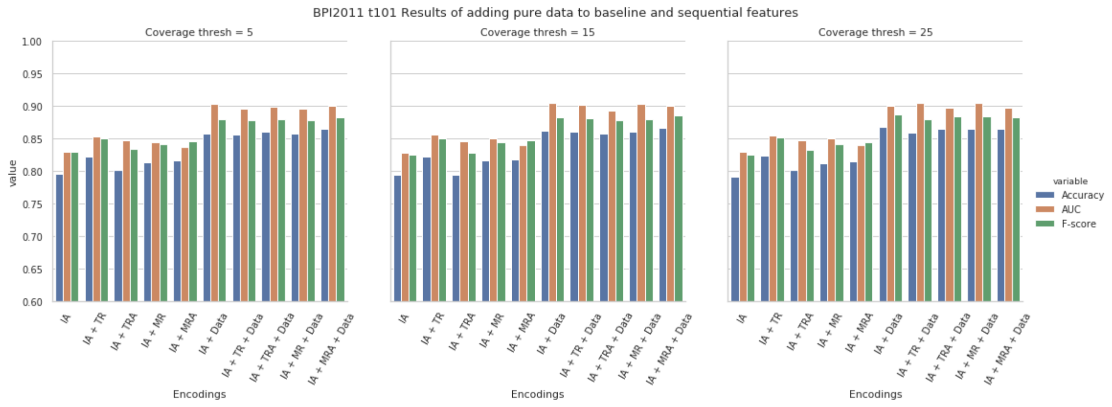


(c) Results with added payload to hybrid model on *BPI2011*$_{m16}$ dataset.

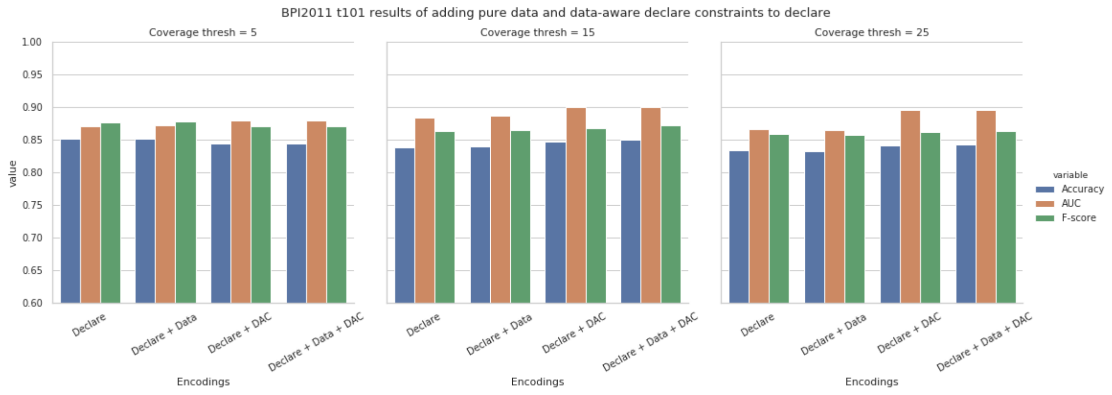


(d) Comparative results of all payload additions on *BPI2011*$_{m16}$ dataset.

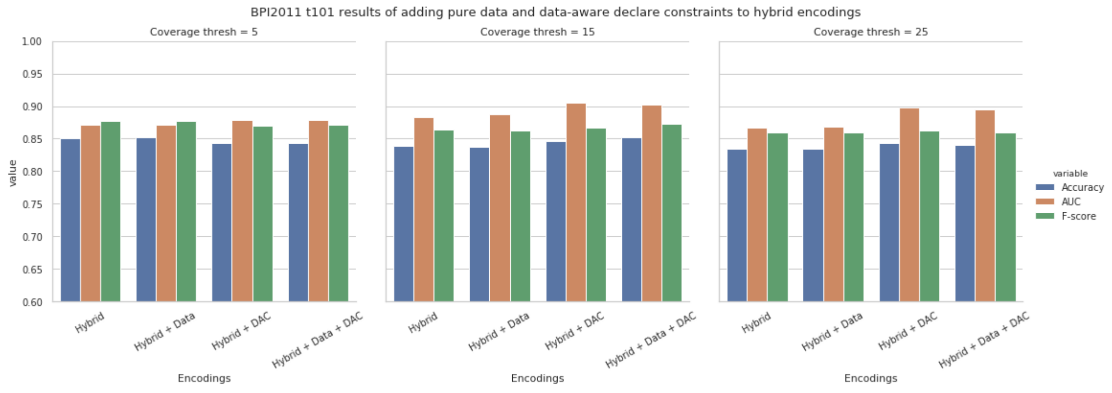
Figure 16. *BPI2011*$_{m16}$ dataset payload results



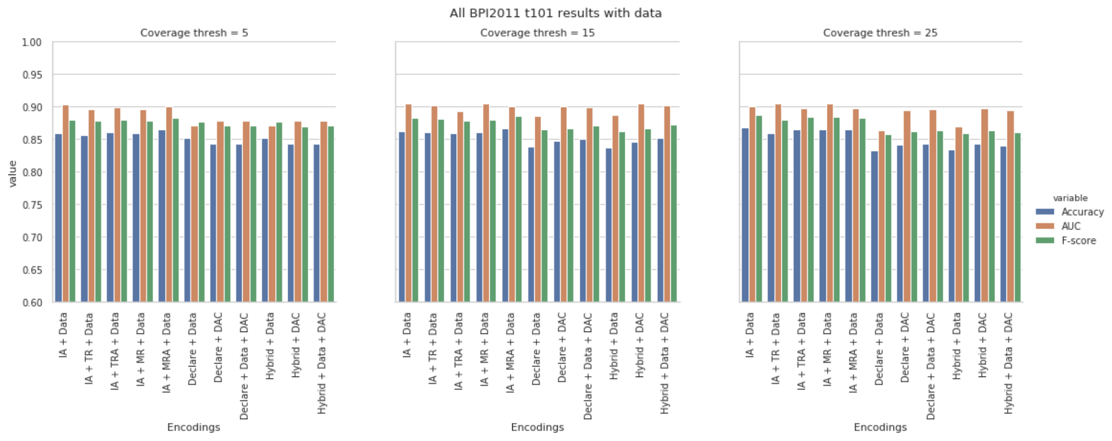
(a) Results with added payload features to baseline and sequential features on *BPI2011*$_{t101}$ dataset.



(b) Results with added payload to declarative features on *BPI2011*$_{t101}$ dataset.

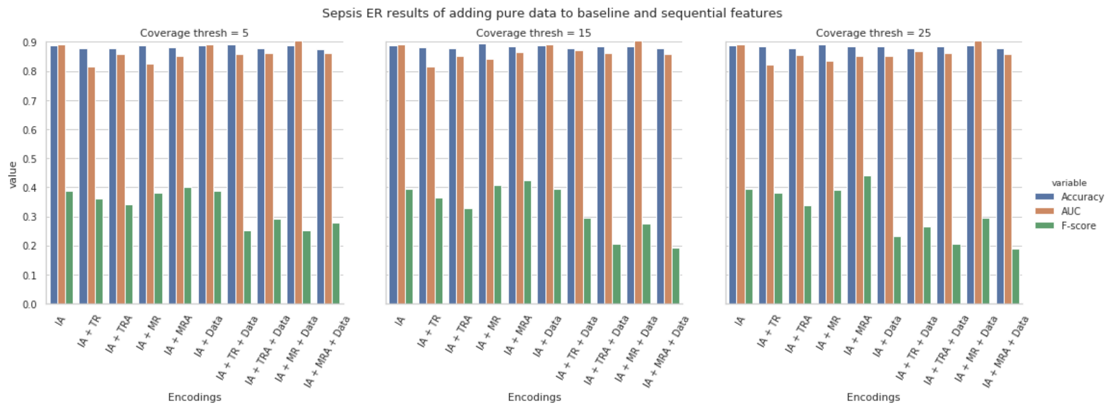


(c) Results with added payload to hybrid model on *BPI2011*$_{t101}$ dataset.

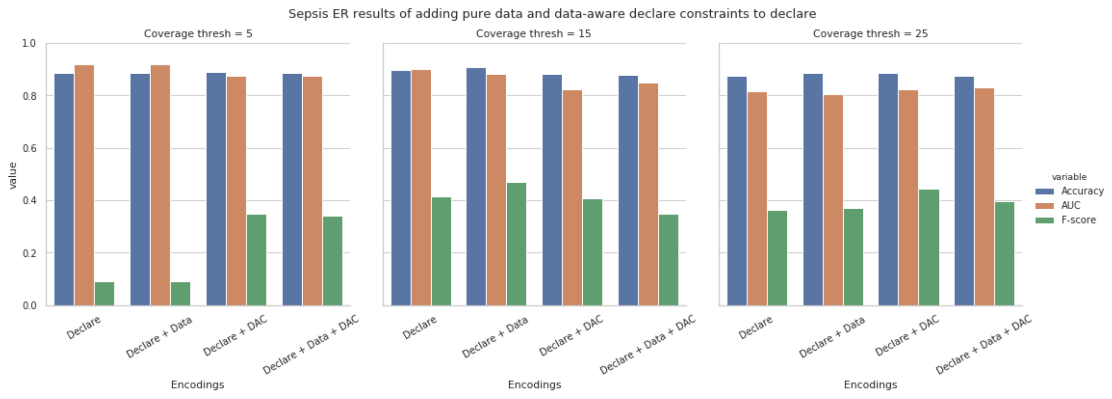


(d) Comparative results of all payload additions on $\mathit{BPI2011}_{t101}$ dataset.

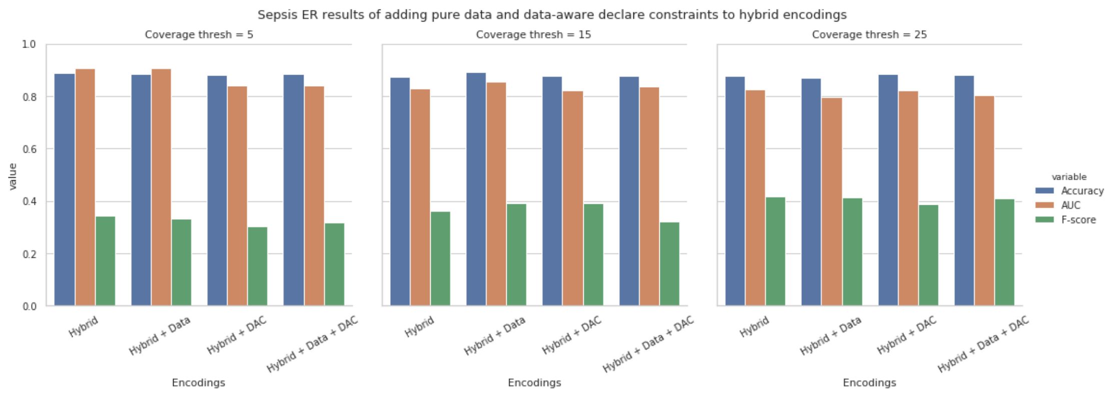
Figure 17. $\mathit{BPI2011}_{t101}$ dataset payload results



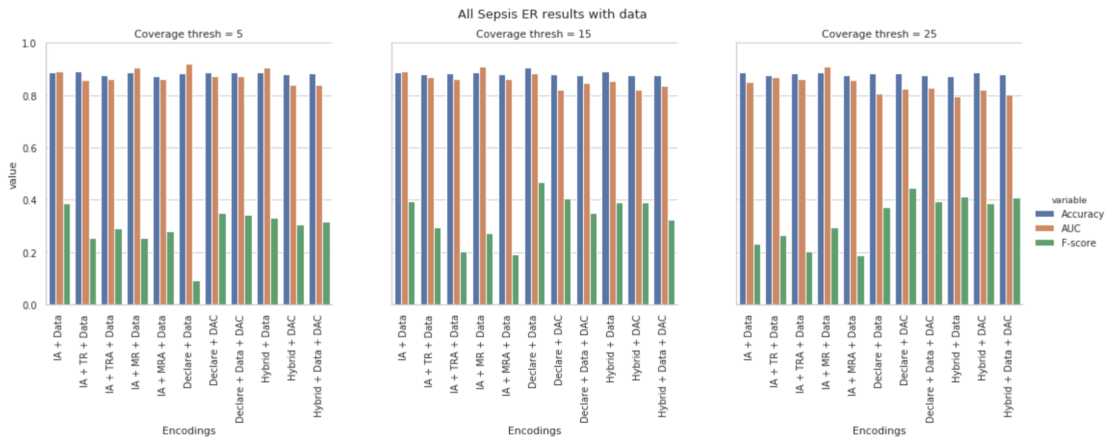
(a) Results with added payload features to baseline and sequential features on Sepsis_{ER} dataset.



(b) Results with added payload to declarative features on Sepsis_{ER} dataset.



(c) Results with added payload to hybrid model on $Sepsis_{ER}$ dataset.



(d) Comparative results of all payload additions on $Sepsis_{ER}$ dataset.

Figure 18. $Sepsis_{ER}$ dataset payload results

6.3 Extraction of Results of Deviance Mining

The goal of the experiments was to compare deviance characterization capabilities of decision trees and RIPPER by comparing them in performance measures of classification and by looking at the *complexity* of output rules. The *goodness* of the extracted rules was then measured by looking at different performance measures of the classifier, such as accuracy, recall, precision, F1-score and AUC. As a measure of *complexity* for the rules, what was measured and compared, was the size of the output ruleset and the average length of the rules within the ruleset.

In detail, we are interested to investigate the following two research questions:

RQ 9. Are RIPPER classification accuracy metrics in line with decision trees?

RQ 10. Does RIPPER provide more compact explantions (i.e. less and shorter decision rules) to explain deviances than the ones extracted from decision trees?

6.3.1 Datasets

Datasets $\mathit{BPIC2011}_{dCC}$, $\mathit{BPIC2011}_{m13}$, $\mathit{BPIC2011}_{m16}$ and $\mathit{BPIC2011}_{t101}$ were used. For the selected datasets, same attributes were removed as was done for payload experiments in Section 6.2.1.

6.3.2 Procedure

The chosen set of encodings for the experiments were control-flow based encodings with *Pure data* features. The experimented encodings were IA+PD, MR+PD, MRA+PD, TR+PD, TRA+PD and Hybrid+PD.

The experiments were performed similarly as in Section 6.1.2 using decision trees[5] and other using RIPPER[6] as classifier. Differently from previous experiments, the resulting rules from decision trees and RIPPER were collected.

Therefore, on labeled event log, for each classification method (decision tree and RIPPER), for each encoding, for each coverage threshold (5, 15, 25) and for each step of 5-fold cross-validation, the following procedure was carried out:

1. event log is split into two sets: 80% training data and 20% testing data;
2. features are discovered and selected (using coverage method for selection) on training data;
3. training data and test data is encoded according to the feature type (for declare the original encoding scheme was used);
4. classifier is trained with encoded training data;

[5]The max depth of the tree was fixed to 10 and the leaves of the decision tree were forced to be of at least size 5.

[6]The number of optimization steps is set to 2.

5. five metrics aiming at evaluating the classifier on the testing data are computed;
6. decision rules are extracted from the classifier and two metrics computed on the extracted rules.

At the end, the average of the 5-fold iterations is computed for all metrics separately for each classification method, encoding and coverage threshold.

Classification performance is compared by using accuracy, AUC, F1-score, recall and precision, which were described in Section 5.1.5. The compactness of the deviance explanations was instead measured by means of two metrics: *rule length* and *ruleset size*. *Rule length* is number of conditions in a specific rule. For example if a decision rule is *(A>0)&(B<0) => 1* (if A is greater than 0 and B is smaller than 0, then classify as 1), then the rule length is 2. *Ruleset size* is the number of rules in resulting decision rule set. If there are 3 rules returned after extracting them from a decision tree or as a result from RIPPER, then the ruleset size is 3.

6.3.3 Results

The results of experiments carried out on dataset $\mathit{BPI2011}_{dCC}$ are plotted in Figure 19, showing that with some exceptions (declare and hybrid with coverage 5, IA with coverage 25, IA+TRA with coverage 15), decision trees (DT) and RIPPER are similar in terms of classification performance metrics. In some cases RIPPER did better in terms of F-measure, but in a few exceptional cases failed in finding a good set of rules (listed as exceptions before), resulting in lower averages. In the case of AUC, RIPPER has slightly worse results compared to DT. For several encodings, RIPPER and DT had close F1, but with RIPPER having higher precision and lower recall.

Figure 20 plots the results on dataset $\mathit{BPI2011}_{m13}$, where DT and RIPPER again have similar classification metrics. In this case RIPPER also resulted in higher F1 for most of the encodings, but a smaller AUC. Is is also noticeable, that in most cases RIPPER has higher precision, but comparable or lower recall. For dataset $\mathit{BPI2011}_{m16}$, with results plotted in Figure 21, the results were analogous.

On dataset $\mathit{BPI2011}_{t101}$ (Figure 22), the results of RIPPER and DT were also pretty similar, with RIPPER having comparable or higher F1 and lower AUC compared to DT.

In comparison between RIPPER and decision trees, RIPPER often resulted in higher F1-measures but in a lower AUC. In some of the experiments, with some of the encodings, using RIPPER resulted with a higher precision, but with a similar or a lower recall, showing a probable tradeoff between recall and precision. It has to be noted though, that in the experiments, decision trees were fixed to a specific set of hyperparameters, and the results on decision trees might change with a change of parameters (**RQ 9**).

Figures 23a, 23b, 23c and 23d, plot results related to the extracted average length of rules and sizes of extracted rules for each encoding. In all the experiments, the RIPPER provided considerably less and shorter rules compared to those extracted from decision trees. (**RQ10**).

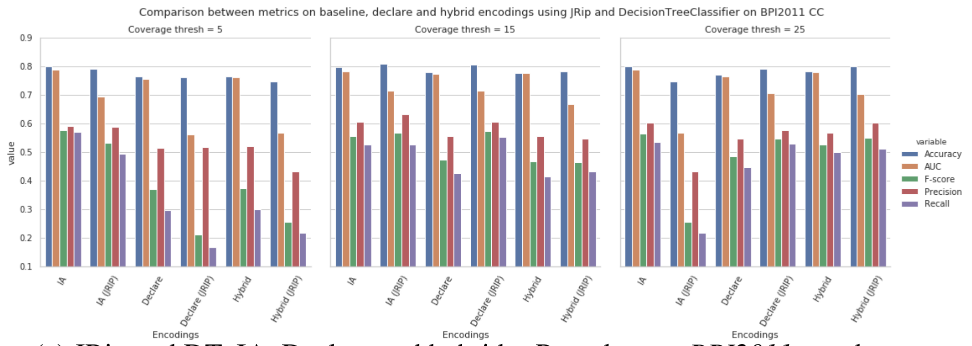
However, it is possible to get shorter and fewer rules by changing hyperparameters of decision trees. Limiting the maximum depth would also limit the length of rules and the count of output rules. Another explanation for the length differences of decision rules is due to differing natures of decision trees and RIPPER algorithm. Decision trees work in a greedy manner by recursively splitting observations into two sets, which at each decision split maximize some measure (e.g. information gain). Due to this, when extracting the rules, all the conditions, which are deeper in the decision tree, also include the conditions upwards the path to the root. RIPPER instead as an instance of sequential covering algorithms, builds one rule at a time, trying to cover as many positive observations and as little negative observations as possible. Having a fixed set of hyperparameters can therefore be seen as a threat to validity for this set of experiments.

As an example of rules retrieved using RIPPER and Hybrid+PD for $bpi2011_m13$ dataset:

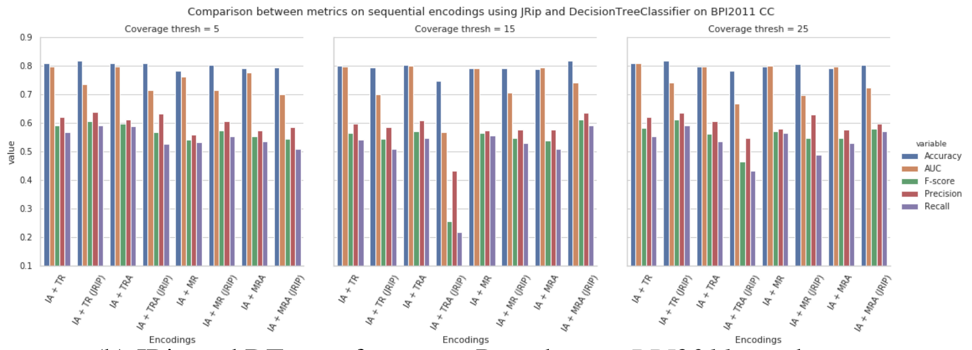
```
(trace:Age|first|discrete <= 56)
        and (response:(First outpatient consultation,
                        uterus - carc. cervical acc. werthei) >= 0)
        and (choice:(squamous cell carcinoma using eia,
                         outpatient follow-up consultation) <= 2)
        and (alternate_precedence:(administrative fee the first pol,
                        outpatient follow-up consultation) >= 0)
        => Label=1 (69.0/4.0)

(choice:(squamous cell carcinoma using eia, assumption laboratory) <= 1)
        and (trace:Age|first|discrete <= 56)
         => Label=1 (224.0/99.0)

 => Label=0 (621.0/104.0)
```

An example of a single rule extracted from a decision tree with the same dataset and encodings:

```
(trace:Age|first|discrete <= 54.5)
        and (responded_existence:(ca-125 using meia,
                squamous cell carcinoma using eia) > -0.5)
        and (responded_existence:(administrative fee - the first pol,
               order rate) <= 3.5)
        and (trace:Age|first|discrete > 45.5)
        and (not_succession:(thorax. echo kidney-urinary tract) <= 0.5)
        and (responded_existence:(squamous cell carcinoma using eia,
                histological examination - biopsies nno) <= -0.5)
        => Label=1 (15.0/0.0)
```

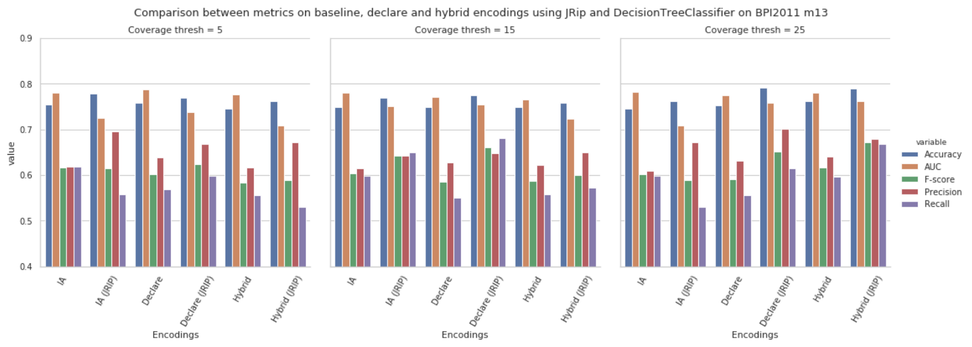


(a) JRip and DT: IA, Declare and hybrid + Pure data on *BPI2011*$_{dCC}$ dataset.

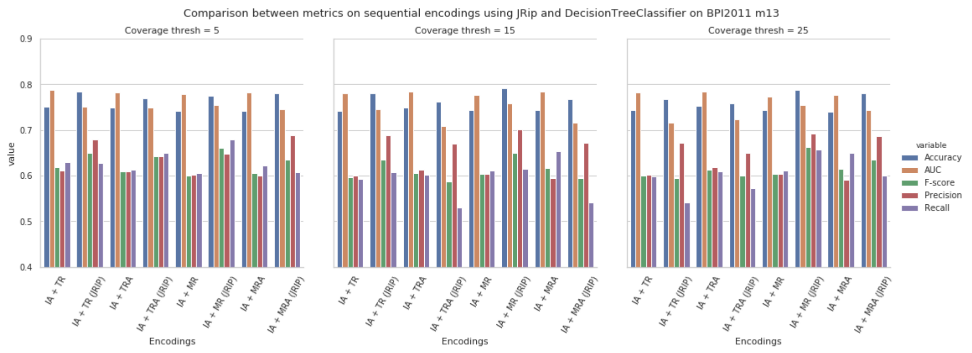


(b) JRip and DT: seq. features + Pure data on *BPI2011*$_{dCC}$ dataset.

Figure 19. JRip and DecisionTree results on *BPI2011*$_{dCC}$ dataset.

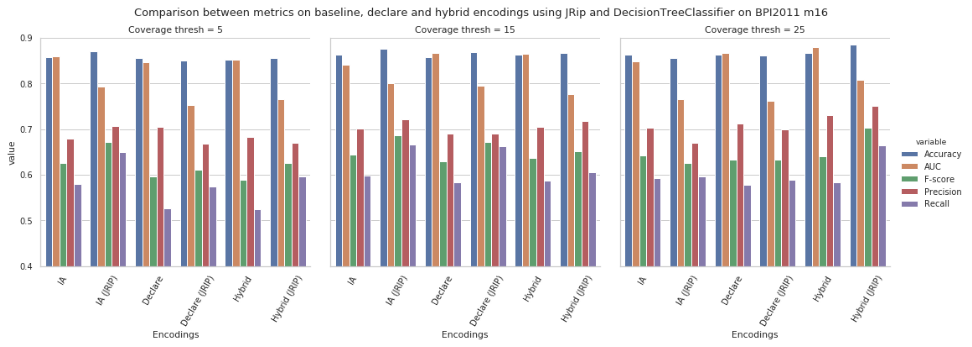


(a) JRip and DT: IA, Declare and hybrid + Pure data on *BPI2011*$_{m13}$ dataset.

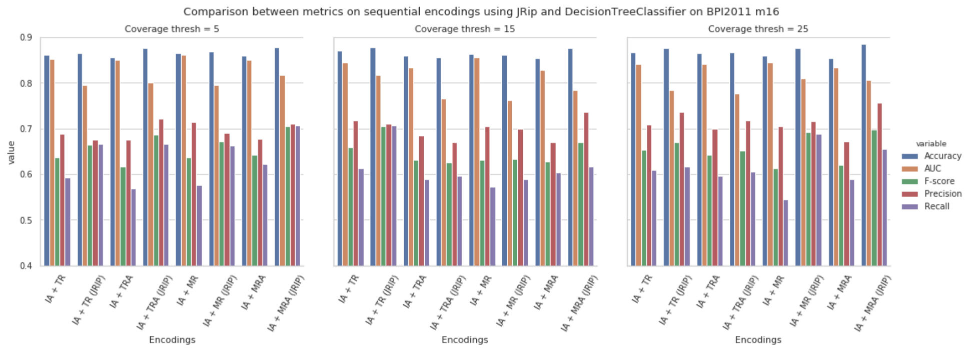


(b) JRip and DT: seq. features + Pure data on *BPI2011*$_{m13}$ dataset.

Figure 20. JRip and DecisionTree results on *BPI2011*$_{m13}$ dataset.

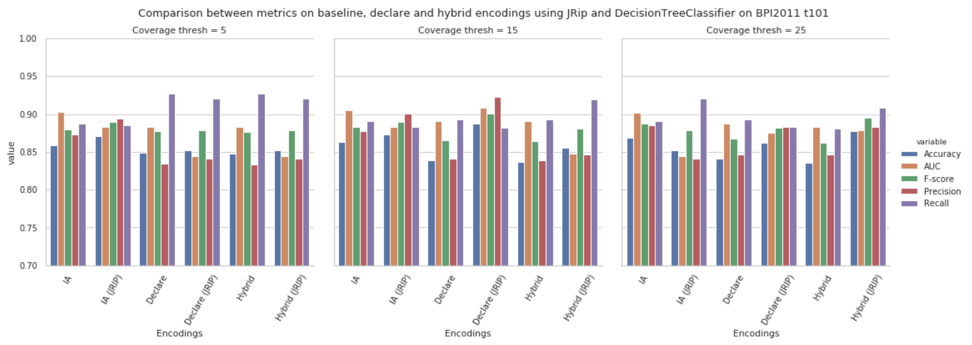


(a) JRip and DT: IA, Declare and hybrid + Pure data on *BPI2011*$_{m16}$ dataset.

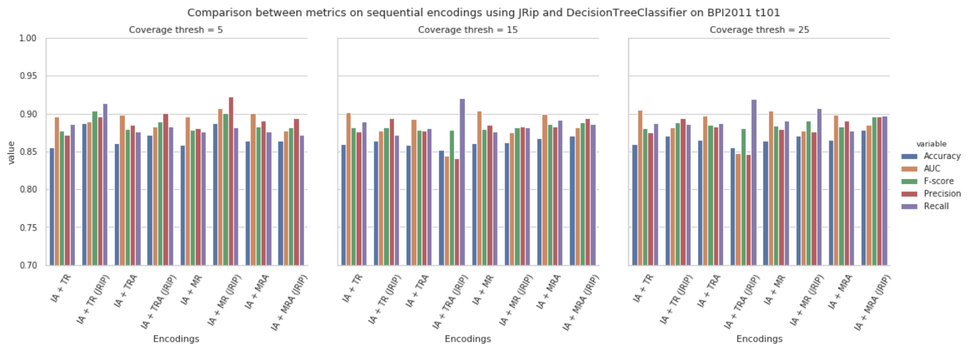


(b) JRip and DT: seq. features + Pure data on *BPI2011*$_{m16}$ dataset.

Figure 21. JRip and DecisionTreeClassifier results on *BPI2011*$_{m16}$ dataset.

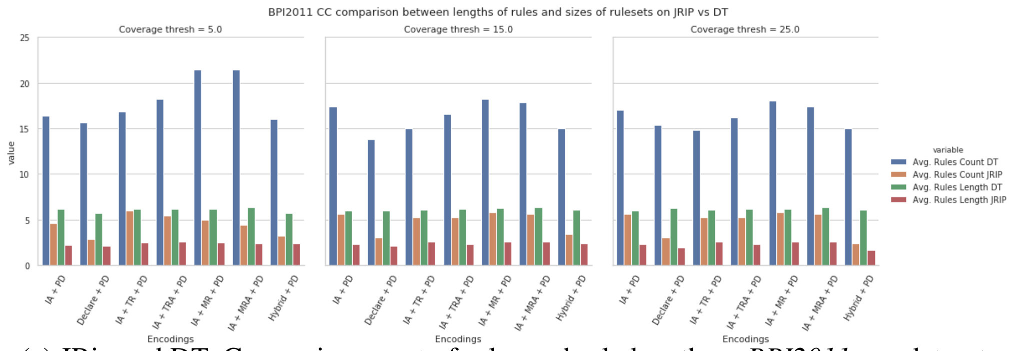


(a) JRip and DT: IA, Declare and hybrid + Pure data on *BPI2011*$_{t101}$ dataset.

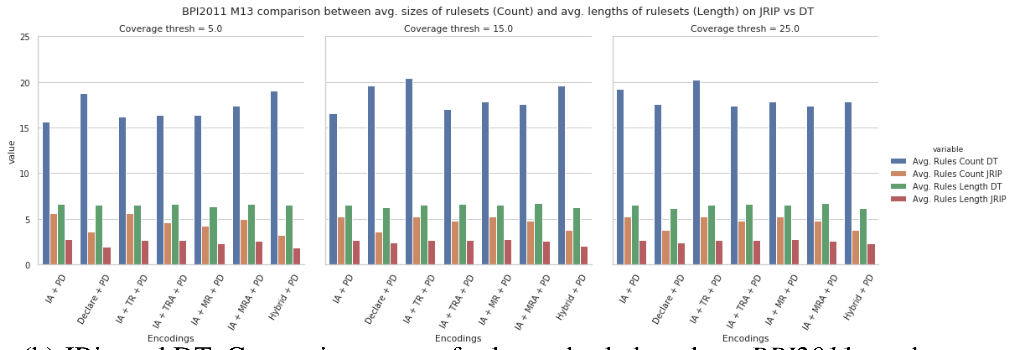


(b) JRip and DT: seq. features + Pure data on *BPI2011*$_{t101}$ dataset.

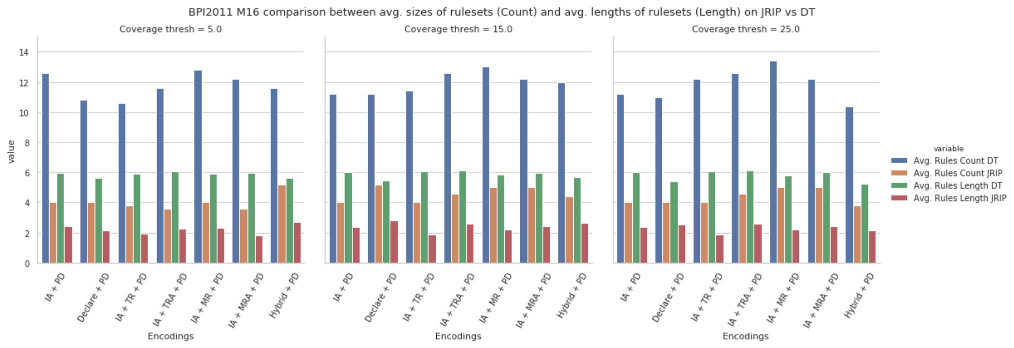
Figure 22. JRip and DecisionTreeClassifier results on *BPI2011*$_{t101}$ dataset.



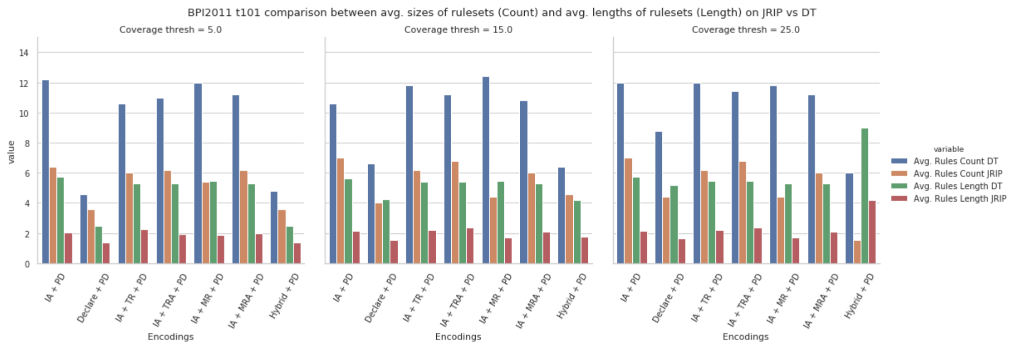
(a) JRip and DT: Comparing count of rules and rule length on *BPI2011*$_{dCC}$ dataset.



(b) JRip and DT: Comparing count of rules and rule length on *BPI2011*$_{m13}$ dataset.



(c) JRip and DT: Comparing count of rules and rule length on *BPI2011*$_{m16}$ dataset.



(d) JRip and DT: Comparing count of rules and rule length on *BPI2011*$_{t101}$ dataset.

Figure 23. Comparison of rule lengths and ruleset sizes of JRip and DecisionTree on *BPI2011* dataset.

7 Conclusions

This thesis has focused on approaches for uncovering and explaining (positive and negative) deviances in business process execution traces. In detail, three different aspects have been investigated.

Firstly, different types of features were explored, ranging from sequential to declarative, up to hybrid ones for explaining deviances. Proposed features were applied to different synthetic and real-life logs, by showing advantages and limits of each of them. Overall, the conclusion was that hybrid encoding is preferable independently of the nature of the process and of the labeling.

Secondly, the thesis investigated the impact of data (together with declarative and sequential features) on the capability of existing techniques to explain process execution deviances. Payload was included by a more straightforward attribute extraction and aggregation method called *Pure data* method and by discovery of data-aware DECLARE constraints. Results showed that the addition of payload features can increase the performance of deviance mining approach. However, the impact of different payload features depends on the characteristics of the log and its labeling. The lack of improvement in performance can be due to a small or no correlation between the data and the labeling of a log. If there is no correlation between the data and the labeling of a log, then the addition of features based on data will not help in increasing the performance.

Thirdly, this thesis took a look at different final outcomes of business process deviance mining process returned to the user. More concretely, two different classifiers were evaluated (RIPPER and decision trees) in the pipeline of business process deviance mining approaches and compared them using classification performance metrics such as accuracy, AUC, F1-measure, and in terms of amount and length of returned decision rules. Results show that the classification performance results of RIPPER and decision trees are in line, while RIPPER provided shorter and less rules compared to those extracted from decision trees. The results suggest that using RIPPER, a variant of sequential covering methods, as a classifier for business process deviance mining is a good alternative to decision trees when requiring more compact explanations in the form of decision rules.

For future work, there are several possible directions to go and ways to improve the present work. One of the possible improvements would be to try out more feature selection methods to find better alternatives to the currently used *coverage* method. To further the effectiveness of data payload, the *Pure data* approach could be improved by considering more features, especially features considering meta-information, with one possibility being a feature corresponding to the total time length of a trace. In order to assess the understandability of the returned rules, an empirical study could be carried out on human subjects for assessing whether more compact rules are actually more understandable.

In order to ease the finding of the best possible explanations, the hyperparameter search should be done automatically (e.g. search of coverage, maximum tree depths) to get the best explanations for a desired metric. For improvement to research in business process deviance mining, the creation of more concrete benchmarks for comparison of approaches is important.

Another possible direction would be to try out model-agnostic methods for describing the important features in classification process, which could allow the use of more complex and powerful models.

References

[Alp14] E. Alpaydin. *Introduction to Machine Learning*. Adaptive Computation and Machine Learning series. MIT Press, 2014.

[AS] R. Agrawal and R. Srikant. Fast Algorithms for Mining Association Rules. In *VLDB 1994*, pages 487–499.

[BCDFM14] Mario Bernardi, Marta Cimitile, Chiara Di Francescomarino, and Fabrizio Maggi. Using discriminative rule mining to discover declarative process models with non-atomic activities. volume 8620, 08 2014.

[Bur16] Andrea Burattin. PLG2: multiperspective process randomization with online and offline simulations. In *BPM Demo Track 2016*, pages 1–6, 2016.

[Bv13] R. P. J. C. Bose and W. M. P. van der Aalst. Discovering signature patterns from event logs. In *2013 IEEE Symposium on Computational Intelligence and Data Mining (CIDM)*, pages 111–118, April 2013.

[BvdA09] R. P. Jagadeesh Chandra Bose and Wil M. P. van der Aalst. Abstractions in process mining: A taxonomy of patterns. In *BPM*, volume 5701 of *Lecture Notes in Computer Science*, pages 159–175. Springer, 2009.

[BvdA13] R. P. Jagadeesh Chandra Bose and Wil M. P. van der Aalst. Discovering signature patterns from event logs. In *CIDM*, pages 111–118. IEEE, 2013.

[CFGP15] Alfredo Cuzzocrea, Francesco Folino, Massimo Guarascio, and Luigi Pontieri. A multi-view learning approach to the discovery of deviant process instances. In *OTM 2015 Conferences*, pages 146–165. Springer International Publishing, 2015.

[CFGP16] Alfredo Cuzzocrea, Francesco Folino, Massimo Guarascio, and Luigi Pontieri. A multi-view multi-dimensional ensemble learning approach to mining business process deviances. In *IJCNN 2016*, pages 3809–3816, 2016.

[CFGP17] Alfredo Cuzzocrea, Francesco Folino, Massimo Guarascio, and Luigi Pontieri. Extensions, analysis and experimental assessment of a probabilistic ensemble-learning framework for detecting deviances in business process instances. In *ICEIS 2017*, pages 162–173, 2017.

[CHX11] Ning Chen, Steven C. H. Hoi, and Xiaokui Xiao. Software process evaluation: A machine learning approach. In *Proceedings of the 2011 26th*

IEEE/ACM International Conference on Automated Software Engineering, ASE '11, pages 333–342. IEEE Computer Society, 2011.

[Coh95] William W. Cohen. Fast effective rule induction. In *In Proceedings of the Twelfth International Conference on Machine Learning*, pages 115–123. Morgan Kaufmann, 1995.

[DHS01] Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification*. Wiley, New York, 2 edition, 2001.

[Faw06] Tom Fawcett. An introduction to roc analysis. *Pattern Recognition Letters*, 27(8):861 – 874, 2006. ROC Analysis in Pattern Recognition.

[HFH$^+$09] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The WEKA data mining software: an update. *SIGKDD Explorations*, 11(1):10–18, 2009.

[Jar16] J. Jarabek. Exploring business process Deviance with Declare. Master's thesis, UT, 2016.

[KV99] Orna Kupferman and Moshe Y. Vardi. Vacuity detection in temporal model checking. In *CHARME 1999*, volume 1703, pages 82–96, 1999.

[LKL07] David Lo, Siau-Cheng Khoo, and Chao Liu. Efficient mining of iterative patterns for software specification discovery. In *13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '07, pages 460–469, 2007.

[LRW13] Geetika T. Lakshmanan, Szabolcs Rozsnyai, and Fei Wang. Investigating clinical care pathways correlated with outcomes. In Florian Daniel, Jianmin Wang, and Barbara Weber, editors, *Business Process Management*, pages 323–338, 2013.

[Man16] Mannhardt, F. (Felix). Sepsis cases - event log, 2016.

[MBvdA12] Fabrizio M. Maggi, R. P. Jagadeesh Chandra Bose, and Wil M. P. van der Aalst. Efficient discovery of understandable declarative process models from event logs. In Jolita Ralyté, Xavier Franch, Sjaak Brinkkemper, and Stanislaw Wrycza, editors, *Advanced Information Systems Engineering*, pages 270–285, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[MCMM13] Marco Montali, Federico Chesani, Paola Mello, and Fabrizio M. Maggi. Towards data-aware constraints in declare. In *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, SAC '13, pages 1391–1396, New York, NY, USA, 2013. ACM.

[MDGBM13] Fabrizio Maria Maggi, Marlon Dumas, Luciano García-Bañuelos, and Marco Montali. Discovering data-aware declarative process models from event logs. In Florian Daniel, Jianmin Wang, and Barbara Weber, editors, *Business Process Management*, pages 81–96, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[MMv11] F. M. Maggi, A. J. Mooij, and W. M. P. van der Aalst. User-guided discovery of declarative process models. In *2011 IEEE Symposium on Computational Intelligence and Data Mining (CIDM)*, pages 192–199, April 2011.

[Mol19] Christoph Molnar. *Interpretable Machine Learning*. 2019. `https://christophm.github.io/interpretable-ml-book/`.

[MPv+10] Marco Montali, Maja Pesic, Wil M. P. van der Aalst, Federico Chesani, Paola Mello, and Sergio Storari. Declarative Specification and Verification of Service Choreographies. *ACM Transactions on the Web*, 4(1), 2010.

[NDLR+14] Hoang Nguyen, Marlon Dumas, Marcello La Rosa, Fabrizio Maria Maggi, and Suriadi Suriadi. Mining business process deviance: A quest for accuracy. In *On the Move to Meaningful Internet Systems: OTM 2014 Conferences*, pages 436–445. Springer Berlin Heidelberg, 2014.

[NDR+16] Hoang Nguyen, Marlon Dumas, Marcello La Rosa, Fabrizio Maria Maggi, and Suriadi Suriadi. Business process deviance mining: Review and evaluation. *CoRR*, abs/1608.08252, 2016.

[Pes08] M. Pesic. *Constraint-Based Workflow Management Systems: Shifting Control to Users*. PhD thesis, TU/e, 2008.

[PSv07] M. Pesic, H. Schonenberg, and W. M. P. van der Aalst. Declare: Full support for loosely-structured processes. In *11th IEEE International Enterprise Distributed Object Computing Conference (EDOC 2007)*, pages 287–287, Oct 2007.

[PVG+11] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[PWS+15] Andrew Partington, Moe Thandar Wynn, Suriadi Suriadi, Chun Ouyang, and Jonathan Karnon. Process mining for clinical processes: A comparative analysis of four australian hospitals. *ACM Trans. Management Inf. Syst.*, 5(4):19:1–19:18, 2015.

[Qui93] J. Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.

[Qui95] J.R. Quinlan. Mdl and categorical theories (continued). In Armand Prieditis and Stuart Russell, editors, *Machine Learning Proceedings 1995*, pages 464 – 470. Morgan Kaufmann, San Francisco (CA), 1995.

[RJB04] James Rumbaugh, Ivar Jacobson, and Grady Booch. *Unified Modeling Language Reference Manual, The (2nd Edition)*. Pearson Higher Education, 2004.

[RM08] Lior Rokach and Oded Maimon. *Data Mining with Decision Trees: Theory and Applications*. World Scientific Publishing Co., Inc., River Edge, NJ, USA, 2008.

[SDFGM18] Vasyl Skydanienko, Chiara Di Francescomarino, Chiara Ghidini, and Fabrizio Maria Maggi. A Tool for Generating Event Logs from Multi-Perspective Declare Models. page 5, 2018.

[SMW$^+$14] Suriadi Suriadi, Ronny S. Mans, Moe T. Wynn, Andrew Partington, and Jonathan Karnon. Measuring patient flow variations: A cross-organisational process mining approach. In Chun Ouyang and Jae-Yoon Jung, editors, *Asia Pacific Business Process Management*, pages 43–58, Cham, 2014. Springer International Publishing.

[SWO$^+$13] Suriadi Suriadi, Moe Thandar Wynn, Chun Ouyang, Arthur H. M. ter Hofstede, and Nienke J. van Dijk. Understanding process behaviours in a large insurance company in australia: A case study. In *CAiSE 2013, Valencia, Spain, June 17-21, 2013. Proceedings*, pages 449–464, 2013.

[Van11] Van Dongen, B.F. (Boudewijn). Real-life event logs - hospital log, 2011.

[WGV14] C W Günther and E Verbeek. Xes standard definition. 03 2014.

II. Licence