Process Mining and Management 2024-2025

Project 1: How much variable is my event log?

In this project we aim at measuring how much variable - in terms of variety of behaviour - are event logs. Computing the variability of a log can indeed be rather useful, for instance, to decide for (a procedural or a declarative) model discovery, or to decide which prediction technique to apply. Variability can be due to small variations in the performed behaviour in different executions or it can be due to concept drifts, i.e., data that evolves over time. In BPM it can happen indeed that the process that we are executing is changed at a certain point in time (e.g., some of the activities are no more executed, new ones are executed, two parts of the process are swapped). The process execution data will hence change starting from that point in time.

A possible way to measure the variability of an event log is counting the number of variants that it contains. A second possibility is to average the edit distance between each pair of traces in the event log.

Write three Python functions, each taking as input an event log and returning a measure of its variability. In detail, you should write the following three functions:

(i) *compute_variant_variablity*: returning the number of variants of the event log;
(ii) *compute_edit_distance_variability*: returning the average edit distance between pairs of traces;
(iii) *compute_my_variability*: returning a measure of the variability of the log that you can define by yourself. You can take into account the only control flow or also look at the variability of the event payloads.

In order to inspect the log, please use the PM4Py library (https://processintelligence.solutions/pm4py) and compute the variability measures (i) on the whole BPI Challenge 2011 log (BPIChallenge2011.xes.zip); (ii) on the whole concept drift log (concept_drift.xes.zip); (iii) on the traces of the concept drift log of type1 (concept_drift_type1.xes.zip); (iv) on the traces of the concept drift log of type 2 (concept_drift_type2.xes.zip). The concept drift log, indeed, is a log composed synthetically putting together traces of type 1 and traces of type 2, which slightly differ the ones from the others, thus creating concept drifts in the event log.

If you are interested to a notion of entropy for event logs, please have a look at the following papers:

- Christoffer Olling Back, Søren Debois, Tijs Slaats: *Towards an Entropy-Based Analysis of Log Variability***.** Business Process Management Workshops 2017: 53-70

- Christoffer Olling Back, Søren Debois, Tijs Slaats: *Entropy as a Measure of Log Variability***.** J. Data Semantics 8(2): 129-156 (2019)

We expect to receive:

- the commented Python code of the required functions;
- a main function and the instructions to run the code;
- a report describing the defined metrics, the overall structure of the code and reporting and discussing the results of the three metrics on the BPI Challenge 2011 and on the three concept drift logs.

Project 2: Analysing real-life logs

In this project, we would like to support a Dutch Financial institute in the analysis of one of its event logs (DutchFinancialInstitute.xes.zip).

The event log contains 262.200 events in 13.087 cases. Apart from some anonymization, the log contains all data as it came from the financial institute. The process represented in the event log is an application process for a personal loan or overdraft within a global financing organization. The amount requested by the customer is indicated in the case attribute AMOUNT_REQ, which is global, i.e., every case contains this attribute. The event log is a merge of three intertwined sub processes. The first letter of each task name identifies from which sub process (source) it originated from. Feel free to run analyses on the process as a whole, on selections of the whole process and/or the individual sub processes.

The data owner is interested in all valuable information related to his process and especially on:

- an estimation of the cycle time of the process;
- the resources that generated the highest activation rate of applications;
- how the process model looks like;
- which decisions have higher influence on the process flow and where they are.

We expect to receive a report describing:

- the analysis carried out;
- the used tools (and settings);
- the considerations done;
- the final conclusions.

Description of the log

An application is submitted through a webpage. Then, some automatic checks are performed, after which the application is complemented with additional information. This information is obtained through contacting the customer by phone. If an applicant is eligible, an offer is sent to the client by mail. After this offer is received back, it is assessed. When it is incomplete, missing information is added by again contacting the customer. Then a final assessment is done, after which the application is approved and activated.

Event types

Event type	**Meaning**
States starting with ‘A_’	States of the application
States starting with ‘O_’	States of the offer belonging to the application
States starting with ‘W_’	States of the work item belonging to the application
COMPLETE	The task (of type ‘A_’ or ‘O_’) is completed
SCHEDULE	The work item (of type ‘W_’) is created in the queue (automatic step following manual actions)
START	The work item (of type ‘W_’) is obtained by the resource
COMPLETE	The work item (of type ‘W_’) is released by the resource and put back in the queue or transferred to another queue (SCHEDULE)

Event types

Dutch state name	**English translation**
W_Afhandelen leads	W_Fixing incoming lead
W_Completeren aanvraag	W_Filling in information for the application
W_Valideren aanvraag	W_Assessing the application
W_Nabellen offertes	W_Calling after sent offers
W_Nabellen incomplete dossiers	W_Calling to add missing information to the application

Project 3: Predictions with intercase feature encoding

In this project we aim at enriching classical control-flow encodings with intercase information. Having features capturing how many concurrent cases are running or the characteristics of the concurrent cases could improve the accuracy of the predictions, especially in case of time-related predictions.

Write the following Python functions:

- *encode_case_simple_index (case, prefix_length, label_encoder)*: a Python function that takes as input a case (an execution trace) *case* of a certain length *prefix_length* (a prefix of the execution trace), the length of the prefix (*prefix_length*) and an encoder of the labels (*label_encoder)* – which encodes and decodes activity labels in numbers - and returns as output a vector encoding the trace using the *simple index encoding*. In detail, in the simple index encoding a (prefix) trace of length *prefix_length* is encoded as a vector of length *prefix_length*, where the p-th item corresponds to the numeric encoding of the name of the p-th event. To this aim, you can leverage a dictionary mapping the activity labels to a numeric value, which is provided by the *label_encoder* – you do not necessarily need to use the one-hot encoding for encoding the categorical activity labels. Please note, that if the case is shorter than *prefix_length* you need to add zero-padding in order to get an encoded vector of length *prefix_length*.
- *count_concurrent_cases (case, log)*: a Python function that takes as input a case (an execution trace) and an event log and returns as output the number of cases in the log executed concurrently (in parallel) to *case*. In detail, to this aim, you need to identify the start timestamp of *case*, i.e., when the case starts, the end timestamp of *case*, that is when the case ends, and you need to count how many cases in the log are executed (have an intersection interval) in the same time range. Please, note that there exist functions in PM4Py that allow you to identify cases run in a certain timeframe.
- *count_avg_duration (case, log)*: a Python function that takes as input a case (an execution trace) and an event log and returns the average duration of the concurrent cases, that is of the cases in the log that are executed in parallel to *case*.
- *count_my_intercase_value*: a Python function that returns one or more intercase features you like most – e.g., the average number of resources used by the concurrent cases, the average overlapping duration of concurrent cases, for each event the number of cases executed concurrently to the event interval, …
- *simple_index_encode (log, prefix_length, label_encoder, conc_cases, avg_dur, my_int)*: a Python function that takes as input a log of traces of length *prefix_length*, an encoder of the labels (*label_encoder)* – which encodes and decodes activity labels in numbers – and

three boolean variables*, conc_cases, avg_dur* and and *my_int*. If one of the three variables is set to true, you should add the corresponding intercase feature to the encoding.

The Production log, which is a real-life log containing data from a manufacturing process, is labelled based on whether the traces were fast (cycle time < average cycle time) or slow (cycle time >= average cycle time): in the former case the label is set to *true*, in the latter to *false*. Train a decision tree model with the prefixes (of length 5) of the training set (80%) of the labelled log Production (Production_avg_mean_training_0-80.xes.zip) by using (i) the simple index encoding; (ii) the simple index and the number of concurrent cases as intercase feature; (iii) the simple index and the number and the average duration of the concurrent cases as intercase feature; (iii) the simple index and your intercase feature; (iv) a combination of the most promising intercase features (or even all of them). Compute the accuracy of the predictions on the prefixes of length 5 of the testing set (20%) of the labelled Production log (Production_avg_mean_testing_80-100.xes.zip) and compare the different intercase encodings. Use padding in case traces are shorter than 5.

We expect to receive:

- the commented Python code of the required functions;
- a main function calling the training and the testing functions and the instructions to run the code;
- a report describing the overall structure of the code and reporting and discussing the results obtained using the simple-index encoding without the intercase features and with (a combination of) the intercase features.