DSP – Assignment 4 (Groups of 2 students)

The objective of this assignment is to analyse an event log and prepare it for machine learning tasks.

PART 1 – Getting the dataset and filtering

The table “loans” in the database “mydb” on the server (the same used in the exercises in class) contains an event log of a process of loan requests by customers of a bank. Your first task is to fetch the data from this table and make an event log (it is strongly suggested to use a pandas dataframe).

Each row in the table ‘loans’ is an event and is characterised by the following attributes (columns):
Column(name='id'): a unique id of the event
Column(name='caseid'): the id of the case to which the event belongs
Column(name='activity'): the activity label
Column(name='ts'): timestamp (there is only 1 timestamp for each event in this log)
Column(name='goal'): the reason for the loan request *
Column(name='apptype'): whether the loan application is new or follows up a previous one *
Column(name='reqamount'): the amount requested *
Column(name='action'): more detail about the activity executed
Column(name='resource'): who executed the activity
Column(name='eventorigin'): the system from which the event originated

* Note that these are “case-level attributes”, i.e., their value is the same for all events belonging to the same case. The other attributes are “event-level” ones, i.e., their value may be different for each individual event.
The case “id” is, obviously, a case-level attribute.

1.1) Write a filter function

```
def filter_event_log(event_log, z):
```

In the analysis tasks described below, you are always interested in considering only the first Z events of a case (that is, the Z events with earlier timestamp for any given case). Therefore, your first task is to write a filtering function that, given an event log dataframe and an integer number Z>0, filters out all the cases that have less than Z events and, for all remaining cases, retains only the first Z events (therefore, note that after applying the filtering, the cases in the filtered log will all have the same number of events). Note also that the number of events per case in this log is comprised between 10 and 63.

PART 2: Create social network graphs

2.1) Social network of work handoffs

A work handoff in an event log happens every time two resources perform consecutive activities in a case. For instance, if in case with id “25” the activity “A” is executed by resource “Alice” and activity “B”, which directly follows “A” (i.e., it was executed after “A”, with no other activities in between executed in the same case), is executed by “Bob”, then an handoff between Alice and Bob has occurred (Alice >> Bob).

```
def get_social_network_handoffs(event_log, z, x):
```

Your task is to build a social network graph using work handoffs in an event log. A social network is a directed graph in which nodes are the resources in the log and there is an edge from resource I to resource J if there are at least X handoffs I >> J in the log. The weight of edges in the graph is the number of handoffs.

Write a function that creates the social network of work handoffs graph when Z and X are given as input.

For values of the input parameters 1: Z=40, X=30 and 2: Z=40, X=45 print the obtained social network on a file named “snw1” and “snw2”, respectively, and print on console the list of node pairs in each graph that are NOT connected (note that, since the graph is directed, node pair [I,J] is different from [J,I]).

PART 3: Event log preparation for machine learning

Your last task is to prepare the event log for machine learning, in particular association rule mining. Association rule mining is a type of machine learning that requires in input a set of “items”. Each item can only assume the value 0 or 1. In the case of event log, the idea is to create a dataset in which each row represents a case, and columns represent the “items” describing a case.

3.1) Preprocessing function

```
def preprocess_event_log(event_log, z):
```

Write a function that, given a value Z comprised between 10 and 63, returns a dataframe containing the filtered event log (as in 1.1) prepared for association rule mining. Specifically, this function returns a dataframe containing one row per case id. Each row contains : (I) case id, (II) a number of items, which can assume only the values 0 or 1. Items are generated from events in a case following the instructions in the Appendix. When executed, this function also prints on console the first 10 row of the dataframe returned.

3.2) Items for a specific case id

```
def get_item(event_log, z, case_id, attr_name, attr_value,
event_pos):
```

Write a function that, given an event log, a case id, an integer Z comprised between 10 and 63 in input, an attribute name (e.g. “resource”), an attribute value (e.g., “user_45”), and event position (from 1 until Z) if the case identified by case id has at least Z events, then returns the items corresponding to attribute name and value for event in that position after preprocessing. Note that the parameter position is not necessary for case-level attributes. An appropriate message is printed on console if the case_id has less than Z events or if the attribute name or value do not exist.
See detailed instructions in the Appendix. Note that for attribute “timestamp” the possible values are “SHORT”, “MEDIUM”, “LONG”. For “reqamount” are “SMALL”, “MEDIUM”, “LARGE”.

Submission Instructions:
You have to conduct this project in **the same group as assignment 1**.
Complete the implementation of the required functions in the file a04-solution.py
You have to **submit the code** that you develop **on blackboard** by the **deadline of December 11th at 10pm**. Please zip the folder "assignment04.zip".

A demo with TAs will be scheduled in due time in the exam week.

Late submission policy:
Up to 6 hours late: the grade will decrease of 5 points for each ½ hour (e.g., your submission is 1.5 hour late, your grade will decrease of 15 points)
More than 6 hours late: these cases are critical and a decision will be taken on a case-by-case basis
(note that all assignments are graded on a scale of 100 points)

Appendix – Generations of items

1) Categorical attributes (e.g., "event" or "apptype", where values are strings) are one-hot encoded
2) Timestamps and other numerical values ("reqamount") are discretised before being one-hot encoded

One-hot-encoding
Consider a generic attribute "at" that assumes categorical values and let {v1, v2, v3, v4} be the set of all values that "at" assumes in an event log. One-hot enconding of attribute "at" means to transform "at" into a set of items, one for each possible assumed value. For a given event (or case) *e* (*c*), all items generated will be set to 0 except the one corresponding to the value assumed by "at" in *e* (*c*), which is set to 1. See detailed example later.

Normalisation and discretisation of timestamps
1. For each event *e*, calculate the difference between its timestamp and the timestamp of the preceding event in the same case. For the first event of a case, this difference is set to 0 by default.
2. Calculate the median (not the mean!) of these differences in the log for each possible activity (I.e, each possible value assumed by the attribute "event" in the log that you are analysing). Given an event in which activity X is executed, assign the discrete values SHORT, MEDIUM, LONG to the difference as follows:
   If value of difference for X in *e* is < 0.4 * median(X) ==> SHORT
   If value of difference for X in *e* is >= 0.4 * median(X) and < 0.65 * median(X) ==> MEDIUM
   If value of difference for X in *e* is >= 0.65 * median(X) ===> LONG
3. Create items by one-hot encoding the discretised values

Discretisation of case-level attribute "reqamount"
1. Calculate the average value of "reqamount" across all cases in the log. For each case, given the requested amount X, assign discrete values as follows:
   If X in case is < 0.4 * avg(X) ==> SMALL
   If X in case is >= 0.4 * avg(X) and < 1.2 * avg(X) ==> MEDIUM
   If X in case is >=1.2* avg(X) ===> LARGE
2. Create items by one-hot encoding the discretised values

Note that: For case-level attributes, a number of items will be generated only once for each case; for event-level attributes, a number of items will be generated for each event in a case.

Detailed example:
Consider an event log in which A, B, C and D are the only possible activities; Alice, Bob, and Chris the only possible resources; and "gold" and "silver" the only possible values that the case-level attribute "Client Type" can assume. We consider Z = 3, so for each case only the first 3 events are considered. Below you can see the events of cases with id 111 and 112.

Event log:

Case ID	Client type	Activity	Resource	Timestamp
111	gold	A	Bob	20
112	silver	A	Bob	19
112	silver	C	Bob	25
111	gold	B	Alice	26
111	gold	C	Chris	40
112	silver	D	Alice	38

First let's calculate the timestamp differences (note that, for a given case, it is always necessary to order events by timestamp if they are not already ordered in the event log).

Case ID	Client type	Activity	Resource	Timestamp
111	gold	A	Bob	0
112	silver	A	Bob	0
112	silver	C	Bob	6 (25-19)
111	gold	B	Alice	6 (26-20)
111	gold	C	Chris	14 (40-26)
112	silver	D	Alice	13 (38-25)

Then, assuming that the median values of activities A, B, C, D in the log are 6, 7, 10, and 10, respectively, the timestamp differences will be normalised as follows:

Case ID	Client type	Activity	Resource	Timestamp
111	gold	A	Bob	SHORT
112	silver	A	Bob	SHORT
112	silver	C	Bob	MEDIUM (6 < 10*0.65)
111	gold	B	Alice	LONG (6 > 7*0.65)
111	gold	C	Chris	LONG (14 > 10*0.65)
112	silver	D	Alice	LONG (13 > 10*0.65)

Finally, now that all values are categorical, we can do one-hot encoding and generate items.

Below you can see the example of the dataframe returned by pre_process_event_log(). Note that the case-level attribute "Client type" generates one set of attributes per case, whereas the event-level attributes generate one set of attributes for each event in a case. Note that the example below only shows the items generated from case-level attribute and the event-level attributes of the first 2 events of each case.

Case id	ct_gold	ct_silv	A1	B1	C1	D1	Alice1	Bob1	Chris1	T1short	T1med	T1long	A2	B2	C2	D2	Alice2	Bob2	Chris2	T2short	T2med	T2long	...
111	1	0	1	0	0	0	0	1	0	1	0	0	0	1	0	0	1	0	0	0	0	1	...
112	0	1	1	0	0	0	0	1	0	1	0	0	0	0	1	0	0	1	0	0	1	0	...

For instance, the function to solve problem (3.2), given the input "111" (the case id), should return the following:
for attribute "activity", value "A", event position "1": 1 (see "A1" in the table above)
for attribute "activity", value "A", event position "2": 0
for attribute "activity", value "B", "event position "2": 1
for attribute "resource", value "Alice", event position "1": 0
for attribute "timestamp", value "medium", event position"2": 0
for attribute "client type", value "gold": 1
etc.