Refining Event Labels

Jul 04, 2019

CONTENTS:

The source code is available under: https://github.com/NicoleVentsch/Refining-Event-Labels.

CHAPTER

ONE

MODULE 1 - FILE CONVERTER

class eventLogConverter.concreteImplementation.**FileUtility**(*defaultDirectory*)

class eventLogConverter.concreteImplementation.**XESFileConverter**

class eventLogConverter.concreteImplementation.**CSVFileConverter**

class eventLogConverter.concreteImplementation.**XESFileCreator**

class eventLogConverter.defaultFileHandlersFactory.**FileConverterFactory**

class eventLogConverter.defaultFileHandlersFactory.**FileCreatorFactory**

class eventLogConverter.fileHandlerBase.**FileHandlerFactory**

register(*fileType*, *fileHandler*)

Register a virtual subclass of an ABC.

Returns the subclass, to allow usage as a class decorator.

class eventLogConverter.fileHandlerBase.**FileCreator**

class eventLogConverter.fileHandlerBase.**FileConverter**

class eventLogConverter.fileUtility.**FileUtilityBase**(*defaultDirectory*)

Return Home

CHAPTER

TWO

MODULE 2 - PREPROCESSING LOG

class eventLogProcessing.DBTool.**DBTool**(*eventLog*)

data base class containing the main preprocessing steps and tools used to access the database

getVariants()

get all the variants from the variantTable

Returns a list of list of Strings representig all the variants

getVariantByID(*vID*)

get a variant given a variantID

Parameters **vID** – a variantID (integer)

Returns a list of Strings representig a variant

getVariantByEventID(*eID*)

get a variant given an eventID

Parameters **eID** – an eventID (integer)

Returns a list of Strings representig a variant

getTracesByVariantID(*vID*)

get all traces within a variant given a variantID

Parameters **vID** – an variantID (integer)

Returns a list of integers representig the traces within a variant

getEventByID(*eID*)

get an event given its eID

Parameters **eID** – an eventID (integer)

Returns an Object representig an event (containing: EventID, VariantID, Position and Event)

getVariantTable()

get the variantTable

Returns a pandas DataFrame representing the variantTable

getEventVariantTable()

get the eventVariantTable

Returns a pandas DataFrame representing the eventVariantTable

Return Home

CHAPTER THREE

MODULE 3 - COST FUNCTION

`costFunction.mappings.`**`createEventIDs`**`(`*variants=[]*`)`

assigns a unique ID to each event of a variant given a list of variants

Parameters **`variants`** – list of variants, i.e., a list of lists

Returns a list of list of tuples (ID,Event), where each ID is unique

`costFunction.mappings.`**`commonLabels`**`(`*variant1*, *variant2*`)`

creates a list of common event labels between two variants

Parameters

- **`variant1`** – first variant as a list of tuples (eventID, event label)
- **`variant2`** – second variant as a list of tuples (eventID, event label)

Returns a list of common event labels (without IDs) between the two variants

`costFunction.mappings.`**`getNumberOfCommonLabels`**`(`*variant1=[]*, *variant2=[]*`)`

gives the number of common event labels between two variants

Parameters

- **`variant1`** – first variant as a list of tuples (eventID, event label)
- **`variant2`** – second variant as a list of tuples (eventID, event label)

Returns number of common event labels of the two variants

`costFunction.mappings.`**`getPositionsLabel`**`(`*string*, *variant*`)`

gives a list of all IDs corresponding to a given event label within a given variant

Parameters

- **`string`** – event label
- **`variant`** – variant as a list of tuples (eventID, event label)

Returns a list of all IDs corresponding to the given event label within the variant

`costFunction.mappings.`**`possibleMappings`**`(`*variant1=[]*, *variant2=[]*`)`

gives a list of all possible mappings between two given variants

Parameters

- **`variant1`** – first variant as a list of tuples (eventID, event label)
- **`variant2`** – second variant as a list of tuples (eventID, event label)

Returns a list of all possible mappings between the two variants where a mapping is a set of matched pairs (ID1,ID2), where the event label corresponding to ID1 is the same as that corresponding to ID2; ID1 is from the first variant and ID2 from the second variant

`costFunction.mappings.`**`positionsOfCandidates`**(*candidates*, *variants*)

gives a list of all IDs referring to some candidate label

Parameters

- **`candidates`** – set of candidate labels for refinement chosen by the user
- **`variants`** – list of all trace variants in event log where each label has unique ID

Returns a list with all event IDs whose label is in the candidate set

`costFunction.cost.`**`costStructure`**(*variant1*, *variant2*, *mapping*)

get the sum of the differences in the distances between each matched pair and other matches pairs

Parameters

- **`variant1`** – the first variant
- **`variant2`** – the second variant
- **`mapping`** – the mapping of the actions from the first to the second variant

Returns sum of the differences in the distances

`costFunction.cost.`**`context`**(*variant*)

gives a two list (x,y) for the variant, the first one containing the set of predecessors of each action in the variant and the second one containing the set of successors of each action in the variant

Parameters **`variant`** – the variant as a list of tuples (eventID, event label) of which we get the list of predecessors and successors

Returns a tuple (x,y) of lists of sets, where x[i] is the set of predecessors of label on position i and y[i] the set of successors of label on position i

`costFunction.cost.`**`costNoMatch`**(*variant1*, *variant2*, *context1*, *context2*, *mapping*)

calculates the cost for labels that are not matched. This cost is given as the sum of the number of their predecessors and successors.

Parameters

- **`variant1`** – the first variant as a list of tuples (eventID, event label)
- **`variant2`** – the second variant as a list of tuples (eventID, event label)
- **`context1`** – predecessors,successors of variant1
- **`context2`** – predecessors,successors of variant2
- **`mapping`** – the mapping for which the costs for the non-matched labels are calculated

Returns the cost for the non-matched labels

`costFunction.cost.`**`costMatched`**(*variant1*, *variant2*, *context1*, *context2*, *mapping*)

calculates the cost for labels that are matched. This cost is given as the sum of the differences in the direct/indirect neighbors of the matched pairs.

Parameters

- **`variant1`** – the first variant as a list of tuples (eventID, event label)
- **`variant2`** – the second variant as a list of tuples (eventID, event label)
- **`context1`** – predecessors,successors of variant1

- **`context2`** – predecessors,successors of variant2
- **`mapping`** – the mapping for which the costs for the matched labels are calculated

Returns the cost for the matched labels

`costFunction.cost.`**`costMapping`**(*cp*, *variant1*, *variant2*, *context1*, *context2*, *mapping*)

gives the total cost of a mapping between two variants based on a weighted sum of the structural costs and the costs for matched and non-matched labels

Parameters

- **`cp`** – custom parameters object
- **`variant1`** – the first variant as a list of tuples (eventID, event label)
- **`variant2`** – the second variant as a list of tuples (eventID, event label)
- **`context1`** – predecessors,successors of variant1
- **`context2`** – predecessors,successors of variant2
- **`mapping`** – the mapping for which the total cost is calculated

Returns the total cost of the mapping

`costFunction.cost.`**`optimalMapping`**(*variant_i*, *variant_j*, *i*, *j*, *context_i*, *context_j*, *matrixx*, *cp*)

given two variants the mapping with the lowest total cost together with the value of this cost will be returned

Parameters

- **`variant_i`** – the first variant as a list of tuples (eventID, event label)
- **`variant_j`** – the second variant as a list of tuples (eventID, event label)
- **`i`** – index of variant_i in variants
- **`j`** – index of variant_j in variants
- **`context_i`** – predecessors,successors of variant_i
- **`context_j`** – predecessors,successors of variant_j
- **`matrixx`** – matrix that should containing the cost of the mappings (after the function was called)
- **`cp`** – custom parameters object

Returns a tuple (mapping, cost) of the mapping with the lowest total cost and the corresponding cost value; a mapping is a set of matched pairs (ID1,ID2), where the event label corresponding to ID1 is the same as that corresponding to ID2; ID1 is from the first variant and ID2 from the second variant

`costFunction.cost.`**`bestMappings`**(*cp*, *variants*, *C*)

get the best mappings for the given variants and update the cost matrix, so that it contains the cost for each optimal mapping

Parameters

- **`cp`** – custom parameters object
- **`variants`** – a list of variants
- **`C`** – the cost matrix that should be updated, so that it contains the costs of the optimal mappings

Returns a list of the best mappings between all combinations of two variants from the given variants `costFunction.cost.`**`context2`**`(`*variant*, *k*`)`

creates a list of k predecessors and successors of all events of a given variant

Parameters

- **`variant`** – variant as a list of tuples (eventID, event label)
- **`k`** – integer specifying the number of predecessors and successors we consider

Returns a list of sets of predecessors and successors of each event within a variant

Return Home

CHAPTER FOUR

MODULE 4 - LABEL REFINEMENT

`refinement.labelRefinement.`**`connectedComponents`**`(`*G*, *candidateLabels*`)`

computes the connected components given a subgraph :param G: a graph object created from the networkx library :param candidateLabels: a list of Strings representing the candidate lables :return: a dictionary containing {candidateLabel: [[comp1],[comp2],. . .]}

`refinement.labelRefinement.`**`sizelargestComponent`**`(`*connectedComponents*`)`

computes the size of the largest components for each candidateLabel :param connectedComponents: a dictionary containing the connected components created from the method connectedComponents() :return: a dictionary with the form {candidateLabel: maxSize([[comp1],[comp2],. . .])}

`refinement.labelRefinement.`**`averagePosition`**`(`*Gi*, *db*`)`

computes the average position of the events for a given connected component, i.e., #Gi :param Gi: a list representing the connected component for a given event [[comp1],[comp2],. . .] :param db: a DBTool object :return: a list with the average position [[avgPosComp1],[avgPosComp2],. . .]

`refinement.labelRefinement.`**`getPosition`**`(`*eID*, *db*`)`

get the position of an event given its eventID :param eID: an eventID (integer) :param db: a DBTool object :return: an integer representig the position of an event within a trace

`refinement.labelRefinement.`**`sortConectedComponents`**`(`*connectedComponents*, *db*`)`

sort the connected components in ascending order w.r.t. their average position :param connectedComponents: a dictionary containing the connected components created from the method connectedComponents() :param db: a DBTool object :return: a dictionary containing the sorted components, i.e., {candidateLabel: [[comp1],[comp2],. . .]}

`refinement.labelRefinement.`**`horizontalRefinement`**`(`*cp*, *graphList*`)`

perform the horizontal relabeling according to the paper :param cp: a customParameters object :param graphList: a list of graphs created from the networkx library :return: the same list of graphs but with relebaled event nodes

`refinement.labelRefinement.`**`verticalRefinement`**`(`*cp*, *graphList*, *db*`)`

perform the vertical relabeling according to the paper :param cp: a customParameters object :param graphList: a list of graphs created from the networkx library :param db: a DBTool object :return: the same list of graphs but with relebaled event nodes

Return Homgit statuse

CHAPTER

FIVE

MODULE 5 - POST-PROCESSING

eventLogProcessing.postProcessing.**eventLogRenaming**(*cp*, *subgraphList*, *db*, *eventLog*)

function that renames the original event log based on the results of the refinement algorithm

Parameters

- **cp** – a customParameters object
- **subgraphList** – a list of graphs created from the networkx library
- **db** – a DBTool object
- **eventLog** – the original event log provided by the user

Returns the refined event log based on the results of the refinement algorithm

Return Home

CHAPTER SIX

GRAPH CREATION

class graph.graphTool.**graphTool**

graph class containing the main functionalities we need for the algorithm

initialization of a graph

createEdgeList(*edges=[]*, *weight=-1*)

creates a list of tuples of edges with the corresponding weights given a list of edges and weights

Parameters

- **edges** – edges given as a list of tuples (eventID1,eventID2)
- **weight** – a weight

Returns a list of tuples (eventID1, eventID2, weight) of edges together with their weight

createGraphFromVariants(*variants=[]*)

updates an empty graph, such that it becomes a weighted graph containing vertices of the form (eventID, event label) and edges of the form (eventID1, eventID2, weight) based on a given list of variants

Parameters **variants** – list of variants, where a variant is given as a list of tuples (eventID, event label), i.e., a list of lists of tuples

clusterDetection(*customParams*)

clusters the variants based on a given threshold; to do so, edges with a weight above the threshold are deleted from the given graph respresenting the optimal mappings

Parameters **customParams** – custom parameter object containing the threshold the algorithm should use

Returns list of subgraphs where each subgraph represents a cluster of variants

getGraph()

function that returns the graph object

Returns nx.Graph() object

addOptimalMappings(*bestMappingsList*, *maxCost*, *candidate_positions*)

updates the graph by assigning new weights to edges between mapped pairs of candidate labels given a list of all optimal mappings between all variants, the max cost for normalization and the positions of the candidate labels :param bestMappingsList: a list containing all best mappings and their costs as tuples (best mapping, cost) :param maxCost: the cost of the best mapping with the highest cost out of all best mappings

Parameters **`candidate_positions`** – a list with all IDs corresponding to all candidate labels

Return Home

CHAPTER SEVEN

CUSTOM PARAMETERS

class objects.customParameters.**customParameters**(*candidateLabels*, *horizontalThreshold*, *verticalThreshold*, *weightStructure*, *weightMatch*, *weightNoMatch*)

class for the custom parameters

initialization function

getCandidateLabels()
function that returns the candidate labels

getHorizontalThreshold()
function that returns the horizontal threshold

getVerticalThreshold()
function that returns the vertical threshold

getStructureWeight()
function that returns the weight structure

getNoMatchWeight()
function that returns the weight for not matched pairs

getMatchWeight()
function that returns the weight for matched pairs

setcandidateLabels(*candidateLabels*)
function that sets the candidate labels

setHorizontalThreshold(*horizontalThreshold*)
function that sets the horizontal threshold

setVerticalThreshold(*verticalThreshold*)
function that sets the vertical threshold

setStructureWeight(*weightStructure*)
function that sets the weight structure

setNoMatchWeight(*weightNoMatch*)
function that sets the weight for not matched pairs

setMatchWeight(*weightMatch*)
function that sets the weight for matched pairs

Return Home

CHAPTER

EIGHT

INDICES AND TABLES

- genindex
- search