Package 'deduplication'

October 30, 2020

Type Package

Title R package for computing the duplicity probility for each mobile device.

Version 0.1.0

Author Bogdan Oancea <bogdan.oancea@gmail.com>, David Salgado <david.salgado.fernandez@ine.es> Sandra Barragan <sandra.barragan.andres@ine.es>

Maintainer Bogdan Oancea <bogdan.oancea@gmail.com>

Description R package for computing the duplicity probility for each mobile device, i.e. the probability of a device to be in a 2-to-1 correspondence with its owner.

License GPL3, EUPL

Imports data.table,
destim,
Matrix,
doParallel,
parallel,
rgeos,
stringr,
xml2,
Rsolnp

Encoding UTF-8

LazyData true

RoxygenNote 7.1.1

Suggests knitr,
rmarkdown

VignetteBuilder knitr

Collate 'antennaNeighbours.R'
'aprioriDuplicityProb.R'
'aprioriOneDevice.R'
'buildCentroidProbs.R'
'buildCentroids.R'
'buildCluster.R'
'buildDeltaProb.R'
'buildDeltaProb2.R'
'buildDuplicityTable1to1.R'
'buildDuplicityTablePairs.R'
'centerOfProbabilities.R'

'checkConnections_step1.R'
'tileEquivalence.R'
'distance_coverArea.R'
'checkConnections_step2.R'
'computeDuplicity.R'
'computeDuplicityBayesian.R'
'dispersionRadius.R'
'readPostLocProb.R'
'computeDuplicityTrajectory.R'
'computePairs.R'
'deduplication.R'
'example1.R'
'example2.R'
'example3.R'
'fitModels.R'
'getDeviceIDs.R'
'getConnections.R'
'getEmissionProbs.R'
'getEmissionProbsJointModel.R'
'getGenericModel.R'
'getJointModel.R'
'modeDelta.R'
'readCells.R'
'readEvents.R'
'readGridParams.R'
'readSimulationParams.R'
'splitReverse.R'

R topics documented:

`antennaNeighbours` *Computes a list of neighbouring antennas.*

Description

Computes a list of pairs antennaID1-antennaID2 with neighbouring antennas. Two antennas are considered neighbours if their coverage areas overlap (i.e. their intersection is not void).

Usage

```
antennaNeighbours(coverarea)
```

Arguments

`coverarea`	a data.table object with two columns: `antennaID` and `cell`. The first column contains the ID of each antenna while the second one contains a `sp` object that represents the coverage area of the corresponding antenna. The coverage area is obtained by calling `readCells()` function.

Value

A data.table object with a single column called `nei`. Each element of this column is a pair of the form antennaID1-antennaID2 where the two antennas are considered neighbours.

`aprioriDuplicityProb` *Apriori probability of duplicity.*

Description

Apriori probability of duplicity, i.e. the probability of a device to be in a 2-to-1 correspondence with the person who holds it. It is computed as $2 * (Ndev - Ndev2)/(Ndev * (Ndev - 1))$ where $Ndev$ is the total number of devices and $Ndev2$ is the number of devices that are in a 1-to-1 correspondence with the persons that hold them.

Usage

```
aprioriDuplicityProb(prob2Devices, ndevices)
```

Arguments

`prob2Devices`	The probability of a person to have 2 devices. In case of using simulated data, this parameter is read from the simulation configuration file.
`ndevices`	The number of devices detected by the network during the time horizon under consideration.

Value

Apriori probability of duplicity, i.e. the probability of a device to be in a 2-to-1 correspondence with the person who holds it.

`apriori0neDeviceProb` *Apriori probability of a device to be in a 1-to-1 correspondence with its holder.*

Description

Apriori probability of a device to be in a 1-to-1 correspondence with the holder. It is computed simply as $(2 * Ndev1 - Ndev)/Ndev1$ where $Ndev1$ is the number of devices that are in a 1-to-1 correspondence with the persons that hold them and $Ndev$ is the total number of devices.

Usage

```
apriori0neDeviceProb(prob2Devices, ndevices)
```

Arguments

`prob2Devices`	The probability of a person to have 2 devices. In case of using simulated data, this parameter is read from the simulation configuration file.
`ndevices`	The number of devices detected by the network during the time horizon under consideration.

Value

The apriori probability of a device to be in a 1-to-1 correspondence with its holder.

`checkConnections_step1`

Make the first step of the checks between the connections and the emission model.

Description

This function obtains the results of the first step of checks for the compatibility between the connections observed and the emission model. It checks if each individual connection is coherent with the probabilities in the emission matrix. In those cases when that connection is not compatible, the connection is imputed with a missing value.

Usage

```
checkConnections_step1(connections, emissionProbs)
```

Arguments

`connections`	A matrix with the connections, a row per device and a column per time.
`emissionProbs`	A matrix of emission probabilities resulting from the function `getEmissionProbs` in **deduplication** package.

Value

A `list` with two elements: one is the information about the checks and the second is the matrix of connections with imputation in the observations not compatible with the model in `emissionProbs`.

`checkConnections_step2` *Make the second step of the checks between the connections and the emission model.*

Description

This function obtains the results of the second step of checks for the compatibility between the connections observed and the emission model. It checks that the transitions from one connections to the following is coherent with the probabilities in the emission matrix. In those cases when the transition is not compatible, a solution to fit the model is given with the needed time padding coefficient.

Usage

```
checkConnections_step2(emissionProbs, connections, gridParams)
```

Arguments

`emissionProbs`	The matrix of emission probabilities.
`connections`	A matrix with the connections, a row per device and a column per time.
`gridParams`	A list with the parameters of the grid: `nrow,ncol,tileX,tileY`.

Value

A `list` with two elements" one is the information about the checks with a vector containing all the time padding coefficients and the second is the matrix of connections with the time padding done by using the maximum coefficient.

`computeDuplicity` *Computes the duplicity probability for each device.*

Description

Computes the duplicity probability for each device using three different methods: pairs, 1-to-1 and trajectory. The theory behind these methods is described in detail in WPI Deliverable 3 and in the paper *An end-to-end statistical process with mobile network data for Official Statistics*.

Usage

```
computeDuplicity(
  method,
  gridFileName,
  eventsFileName,
  signalFileName,
  antennaCellsFileName = NULL,
  simulatedData = TRUE,
  simulationFileName,
  netParams = NULL,
  path = NULL,
  gamma = 0.5,
  aprioriProbModel = NULL,
  aprioriProbJointModel = NULL,
  lambda = NULL,
  handoverType = "strength",
  emissionModel = NULL,
  antennaFileName = NULL,
  prefix = NULL
)
```

Arguments

method	The method used to compute the duplicity probability. It could have one of the following values: "pairs", "1to1", "trajectory".
gridFileName	The name of the file with the grid parameters. This file could be the one generated by the simulation software or can be created with any text editor. The grid file generated by the simulation software has the following columns: Origin X,Origin Y,X Tile Dim,Y Tile Dim,No Tiles X,No Tiles Y. We are interested only in the number of rows and columns and the tile size on OX and OY axes. Therefore, the file provided as input to this function should have at least the following 4 columns: No Tiles X ,No Tiles Y, X Tile Dim and Y Tile Dim.
eventsFileName	The name of the file with the network events to be used. Depending on the parameter simulatedData it could be a .csv file coming from the simulation software or from a real MNO. In case the file comes from the simulation software it contains following columns: time,antennaID,eventCode,deviceID,x,y,tile Only the first 4 columns are used, the rest are ignored.
signalFileName	The name of the .csv file that contains the signal strength/quality for each tile in the grid. Depending on the parameter simulatedData it could be a .csv file coming from the simulation software or from a real MNO. In case the file comes from the simulation software the data are organized as a matrix with the number of rows equals to the number of antennas and the the following columns: Antenna ID,Tile 0,Tile 1,... Tile (N-1). On the first column there are the antenna IDs and on the rest of the columns the corresponding signal strength/quality for each tile in the grid.
simulatedData	If TRUE the input data are provided by the simulation software, otherwise real data is used.
simulationFileName	The name of the file used to define a simulation scenario. It is the file that was provided as input for the simulation software. This file is required only if simulatedData is TRUE.

`netParams` This parameter is required if simulatedData is FALSE, i.e. the **deduplication** package uses real mobile network data. In this case, `netParam` is a list with two elements: `conn_threshold` and `prob_sec_mobile_phone`. `conn_threshold` is the minimum value of the signal strength/quality that can be used to connect a mobile device to an antenna. If the signal in a region is below this value the region is considered out of the coverage area. `prob_sec_mobile_phone` is the probability of a person to have two mobile devices.

`path` The path where the files with the posterior location probabilities for each device are to be found. This parameter is needed only if the "trajectory" method is used.

`gamma` This value is used only for the "trajectory" method and is the factor used in the compariosn between the mode of Delta X and Delta Y and the dispersion radius. The default value is 0.5.

`aprioriProbModel`
This parameter is used to initialize the apriori probabilities for the HMM model for each device. The default value is NULL which means that by default the models are initialized with the steady state.

`lambda` This parameter is used in combination with the "1-to-1" method. If it is NULL (which is the default value) the computations follow the description given in WPI Deliverable 3, otherwise the computations proceed as they are described in the *An end-to-end statistical process with mobile network data for Official Statistics* paper.

`handoverType` The handover mechanism used by the mobile network. It could have two values: "strength" or "quality". The default value is "strength". This parameter is used to compute the emission probabilities for the HMM models using the values from the signal file. If this file contains the signal strength then the `handoverType` should be set to "strength" and if the file contains the values of the signal quality then the parameter should be set to "quality".

`emissionModel` This parameter dictates how the emission probabilities for the HMM models are computed. It can have two values: "RSS" and "SDM". Normally, the emission probabilities are computed using the signal values from the signal file assuming that if the `handoverType` is set to "strength" then the signal file contains the strength values and if the `handoverType` is set to "quality" the signal file contains the quality of the signal. For demonstrative purposes the package supports unusual combinations. If `handoverType` is set to "strength", the signal file contains the signal strength but the `emissionModel` is set to "SDM" the signal strength values are transformed to signal quality and then the computation of the emission probabilities uses these transformed values. If `handoverType` is set to "quality", the signal file contains the signal quality but the `emissionModel` is set to "RSS", the signal quality values are transformed to signal strength and then the computation of the emission probablities uses these transformed values.

`antennaFileName`
The name of the xml file containing the technical parameters of the antennas needed to transform the signal quality into signal strength and the other way around. This is an input file of the simulation software. An example file is included in this package. The default value is NULL because this file is only needed to make unusual combinations between `handoverType` and the way the emission probabilities are computed.

`prefix` The file name prefix for the files with posterior location probabilities.

`cellsFileName` The name of the file where the coverage areas of antennas are to be found. It is a should be a .csv file with two values on each row: the antenna ID and a WKT

string representing a polygon (i.e. it should start with the word POLYGON) which is the coverage area of the corresponding antenna. This area is also called the antenna cell.

`aprioriJointModel`
: This parameter is used to initialize the apriori probabilities for the joint HMM models for each pair of two devices. The default value is NULL which means that by default the models are initialized with the steady state.

Value

a data.table object with two columns: `deviceID` and `dupP`. On the first column there are deviceIDs and on the second column the corresponding duplicity probability, i.e. the probability that a device is in a 2-to-1 correspondence with its holder.

`computeDuplicityBayesian`
: *Computes the duplicity probabilities for each device using a Bayesian approach.*

Description

Computes the duplicity probabilities for each device using a Bayesian approach. It uses two methods: "pairs" and "1to1". The "pairs" method considers the possible pairs of two compatible devices. These devices were selected by `computePairs()` function taking into consideration the antennas where the devices are connected and the coverage areas of antennas. Two devices are considered compatible if they are connected to the same or to neighbouring antennas. Thus, the data set with pairs of devices will be considerable smaller than all possible combinations of two devices. The "1to1" method considers all pairs of two devices when computing the duplicity probability, the time complexity being much greater than that of the "pairs" method. Both methods uses parallel computations to speed up the execution. They build a cluster of working nodes and splits the pairs of devices equally among these nodes.

Usage

```
computeDuplicityBayesian(
  method,
  deviceIDs,
  pairs4dupl,
  modeljoin,
  llik,
  P1 = NULL,
  Pii = NULL,
  init = TRUE,
  lambda = NULL
)
```

Arguments

`method`
: Selects a method to compute the duplicity probabilities. It could have one of the two values: "pairs" or "1to1". When selecting "pairs" method, the `pairs4dupl` parameter contains only the compatible pairs of devices, i.e. the pairs that most

of the time are connected to the same or to neighbouring antennas. "1to1" method checks all possible combinations between devices to compute the duplicity probabilities.

`deviceIDs` A vector with the device IDs. It is obtained by calling the `getDevices()` function.

`pairs4dupl` A data.table object with pairs of devices and pairs of antennas where these devices are connected. It can be obtained by calling `computePairs()` function.

`modeljoin` The joint HMM model returned by `getJointModel()` function.

`llik` A vector with the values of the log likelihoods after the individual HMM models for each device were fitted. This vector can be obtained by calling `fitModels()` function.

`P1` The apriori duplicity probability as it is returned by `aprioriDuplicityProb()` function. It is used when "pairs" method is selected.

`Pii` Apriori probability of a device to be in a 1-to-1 correspondence with the holder as it is returned by `aprioriOneDeviceProb()` function. This parameter is used only when "1to1" method is selected.

`init` A logical value. If TRUE, the `fit()` function uses the stored steady state as fixed initialization, otherwise the steady state is computed at every call of `fit()` function.

`lambda` It is used only when "1to1" method is selected and a non NULL value mean that the computation of the duplicity probabilities is performed according to the method described in *An end-to-end statistical process with mobile network data for Official Statistics* paper.

Value

a data.table object with two columns: `deviceID` and dupP. On the first column there are deviceIDs and on the second column the corresponding duplicity probability, i.e. the probability that a device is in a 2-to-1 correspondence with the holder.

`computeDuplicityTrajectory` *Computes the duplicity probabilities for each device using the trajectory approach.*

Description

Computes the duplicity probabilities for each device using the trajectory approach described in WPI Deliverable 3.

Usage

```
computeDuplicityTrajectory(
  path,
  prefix,
  devices,
  gridParams,
  pairs,
  P1,
```

```
  T,
  gamma
)
```

Arguments

path	The path where the files with the posterior location probabilities for each device are to be found.
devices	A vector with device IDs.
gridParams	A list with the number of rows and columns of the grid and the tile dimensions on OX and OY axes. The items of the list are named nrow, ncol, tileX and tileY.
pairs	A data.table object containing pairs with the IDs of compatible devices. It is obtained calling buildPairs() function.
P1	The apriori probability for a device to be in 1-to-1 correspondence with its owner.
T	The sequence of time instants in the data set.
gamma	A coefficient needed to compute the duplicity probability. See WPI Deliverable 3.

Value

a data.table object with two columns: deviceID and dupP. On the first column there are deviceIDs and on the second column the corresponding duplicity probability, i.e. the probability that a device is in a 2-to-1 correspondence with the holder.

---

computePairs *Builds pairs of devices and corresponding connections.*

Description

Builds a data.table object that contains pairs of antenna IDs in the form "antennaID1-antennaID2", where antennaID1 corresponds to a device while antennaID2 corresponds to another device, for each time instant over a time period when the network events are registered. These pairs are build for each distinct combination "deviceID1-deviceID2" of devices. The rows of the data.table object corresponds to a combination of devices and the columns corresponds to different time instants. The first two columns contains the device IDs of each pair of devices and the rest of the columns correspond to a time instant and contains the pairs antennaID1-antennaID2 where the two devices are connected at that time instant.

Usage

```
computePairs(
  connections,
  ndevices,
  oneToOne = TRUE,
  P1 = 0,
  limit = 0.05,
  antennaNeighbors = NULL
)
```

Arguments

`connections`	A matrix with the antenna IDs where the mobile devices are connected at every time instant. Each row corresponds to a device and each column to a time instant. This matrix is obtained by calling `getConnections()` function.
`ndevices`	The number of devices registered by the network. A vector with device IDs can be obtained by calling `getDevices()` function and the number of devices is simply the lenght of this vector.
`oneToOne`	If TRUE, the result is built to apply the method "1to1" to compute the duplicity probability for each device. This means that the result will contain all combinations of devices. If FALSE, the result will consider the proximity of antennas through the parameter `antennaNeighbors` and remove all the combinations of devices that are impossible to belong to a single person. If most of the time instants two devices are connected to two antennas that are not neighbors (i.e. their cells don't overlap) we consider that these two devices belong to different persons and remove this combination of devices from the result. The term "most of the time instants" is implemented like this: we add how many times during the time horizon two devices are connected to neighboring antennas and than keep in the final result only those combinations of devices with this number greater than the quantile of the sequence 0...(Number of time instants) with probability 1-P1-limit. In this way we reduce the time complexity of the duplicity probability computation.
`P1`	The apriori probability of duplicity. It is obtained by calling `aprioriDuplicityProb()` function.
`limit`	A number that stands for the error in computing apriori probability of duplicity.
`antennaNeighbors`	A data.table object with a single column `nei` that contains pairs of antenna IDs that are neighbors in the form antennaID1-antennaID2. We consider that two antennas are neighbors if their coverage areas has a non void intersection.

Value

a data.table object. The first two columns contain the devices indices while the rest of the columns contains pairs antennaID1-antennaID2 with antenna IDs where the devices are connected. There is one column for each time instant for the whole time horizon.

`deduplication` *deduplication: A package for computing the duplicity probability for mobile device detected by the network.*

Description

This package contains functions to compute the duplicity probability for mobile device detected by the network. It has three methods for this purpose: pairs, 1-to-1 and trajectory. The theory behind these methods is described in detail in WPI Deliverable 3 and in the paper *An end-to-end statistical process with mobile network data for Official Statistics*. For an example on how to use this package please read example.

Details

deduplication: A package for computing the duplicity probability for mobile devices.

example1 *Example of using* **deduplication** *package - the simple way*

Description

This is just an example on how to compute duplicity probabilities using simulated data. All the files used in this example are supposed to be produced using the simulation software. The "simulation.xml" file is an exception and it is an input file for the simulation software. The files used in this example are provided with the **deduplication** package.

Usage

```
example1()
```

Details

This is just an example on how to compute duplicity probabilities using simulated data. All the files used in this example are supposed to be produced using the simulation software. The "simulation.xml" file is an exception and it is an input file for the simulation software. The files used in this example are provided with the **deduplication** package.This example showsthe simplest way of using this package. It reads the necessary input data and then calls `computeDuplicity` function.

References

https://github.com/MobilePhoneESSnetBigData

Examples

```
# set the folder where the necessary input files are stored
path_root      <- 'extdata'

# set the grid file name, i.e. the file where the grid parameters are found
gridfile <-system.file(path_root, 'grid.csv', package = 'deduplication')

# set the events file name, i.e. the file with network events registered during
# a simulation
eventsfile<-system.file(path_root, 'AntennaInfo_MNO_MNO1.csv',
package = 'deduplication')

# set the signal file name, i.e. the file where the signal strength/quality
# for each tile in the grid is stored
signalfile<-system.file(path_root, 'SignalMeasure_MNO1.csv',
package = 'deduplication')

# set the antenna cells file name, i.e. the file where the simulation software
# stored the coverage area for each antenna
# This file is needed only if the duplicity probabilities are computed using
# "pairs" method
antennacellsfile<-system.file(path_root, 'AntennaCells_MNO1.csv',
package = 'deduplication')

# set the simulation file name, i.e. the file with the simulation parameters
# used to produce the data set
```

```
simulationfile<-system.file(path_root, 'simulation.xml',
package = 'deduplication')

# compute the duplicity probabilities using the "pairs" method
out1<-computeDuplicity("pairs", gridFileName = gridfile,
eventsFileName = eventsfile, signalFileName = signalfile,
antennaCellsFileName = antennacellsfile, simulationFileName = simulationfile)

# compute the duplicity probabilities using the "1to1" method
out2<-computeDuplicity("1to1", gridFileName = gridfile,
eventsFileName = eventsfile, signalFileName = signalfile,
simulatedData = TRUE, simulationFileName = simulationfile)

# compute the duplicity probabilities using the "1to1" method with lambda
out2p<-computeDuplicity("1to1", gridFileName = gridfile,
eventsFileName = eventsfile, signalFileName = signalfile,
simulatedData = TRUE, simulationFileName = simulationfile, lambda = 0.67)

# compute the duplicity probabilities using the "trajectory" method
prefix <- 'postLocDevice'
out3<-computeDuplicity("trajectory", gridFileName = gridfile,
eventsFileName = eventsfile, signalFileName = signalfile,
antennaCellsFileName = antennacellsfile, simulationFileName = simulationfile,
path= system.file(path_root, package='deduplication'), prefix = prefix)
```

`example2` *Example of using* **deduplication** *package - the long way*

Description

This is just an example on how to compute duplicity probabilities using simulated data. All the files used in this example are supposed to be produced using the simulation software. The "simulation.xml" file is an exception and it is an input file for the simulation software. The files used in this example are provided with the **deduplication** package.

Usage

```
example2()
```

Details

This is detailed example on how to compute duplicity probabilities using simulated data. All the files used in this example are supposed to be produced using the simulation software. The "simulation.xml" file is an exception and it is an input file for the simulation software. The files used in this example are provided with the **deduplication** package. This example shows step by step all intermediate computations performed before calling one the functions that computes the duplicity probabilities. All three methods are used in this example: "1-to-1", "pairs" and "trajectory".

References

https://github.com/MobilePhoneESSnetBigData

Examples

```
# set the folder where the necessary input files are stored

path_root      <- 'extdata'

# 0. Read simulation params

simParams <-readSimulationParams(system.file(path_root, 'simulation.xml',
package = 'deduplication'))

 # 1. Read grid parameters

gridParams <-readGridParams(system.file(path_root, 'grid.csv',
package = 'deduplication'))

 # 2.Read network events

events <- readEvents(system.file(path_root, 'AntennaInfo_MNO_MNO1.csv',
package = 'deduplication'))

 # 3. Get a list of detected devices

devices <- getDeviceIDs(events)

#4. Get connections for each device

connections <- getConnections(events)

#5. Emission probabilities are computed from the signal strength/quality file

emissionProbs <- getEmissionProbs(gridParams$nrow, gridParams$ncol,
system.file(path_root, 'SignalMeasure_MNO1.csv', package = 'deduplication'),
simParams$conn_threshold)

#6. Build joint emission probabilities

jointEmissionProbs <- getEmissionProbsJointModel(emissionProbs)

#7. Build the generic model

model <- getGenericModel(gridParams$nrow, gridParams$ncol, emissionProbs)

#8. Fit models

ll <- fitModels(length(devices), model,connections)

#9. Build the joint model

modelJ <- getJointModel(gridParams$nrow, gridParams$ncol, jointEmissionProbs)

#10. Read antenna cells and build a matrix of neighboring antennas

coverarea <- readCells(system.file(path_root, 'AntennaCells_MNO1.csv',
package = 'deduplication'))

antennaNeigh <- antennaNeighbours(coverarea)
```

```
#11. Apriori probability of duplicity

P1 <- aprioriDuplicityProb(simParams$prob_sec_mobile_phone, length(devices))

#12. Build a matrix of pairs of devices to compute duplicity probability

pairs4dupP<-computePairs(connections, length(devices), oneToOne = FALSE, P1=P1,
limit = 0.05, antennaNeighbors = antennaNeigh)

#13. Compute duplicity probabilities using the "pairs" method (faster)

out1 <- computeDuplicityBayesian("pairs", devices, pairs4dupP, modelJ, ll, P1)

#14. Apriori probability of 2-to-1

Pii <- aprioriOneDeviceProb(simParams$prob_sec_mobile_phone, length(devices))

#15. Build a matrix of pairs of devices to compute duplicity probability

pairs4dupO<-computePairs(connections, length(devices), oneToOne = TRUE)

#16. Compute duplicity probabilities using "1to1" method

out2 <- computeDuplicityBayesian("1to1", devices, pairs4dupO, modelJ, ll,
P1 = NULL, Pii=Pii)

#17. Compute duplicity probabilities using "trajectory method"

T<-sort(unique(events[,1][[1]]))

out3 <-computeDuplicityTrajectory(path=system.file(path_root, package = 'deduplication'),
"postLocDevice", devices, gridParams, pairs4dupP, P1 = P1, T, gamma = 0.5)
```

`example3` *Example of using* **deduplication** *package - the case of incompatibility*

Description

Example of using **deduplication** package - showing the case of incompatibility between the type of connection in the network and the emission model.

Usage

```
example3()
```

Details

This is an example of how to force the use of an emission model that differs from the type of connection done by the network. Thanks to the information from the simulator and the flexibility of the functions of this package, it is possible to make the assumption of a kind of emission model different from the real type of connection. Obviously, this incoherence has consequences. It could cause to type of problems. Firstly there are events to antennas without signal under the assumption made to build the emission probability matrix. Secondly, there are transitions between pairs of

antennas which are not possible in consecutive times because there is some tile without signal between those antennas under the assumption made to build the emission probability matrix. These two problems can be detected by two functions in this package: `checkConnections_step1()` and `checkConnections_step2()`. In the following example, one of these kinds of cases is shown.

References

https://github.com/MobilePhoneESSnetBigData

Examples

```
###    READ DATA    ####

# Set the folder where the necessary input files are stored

path_root <- 'extdata'

 # 0. Read simulation params

simParams <- readSimulationParams(system.file(path_root, 'simulation.xml',
package = 'deduplication'))

 # 1. Read grid parameters

gridParams <- readGridParams(system.file(path_root, 'grid.csv',
package = 'deduplication'))

 # 2. Read network events

events <- readEvents(system.file(path_root, 'AntennaInfo_MNO_MNO1.csv',
package = 'deduplication'))

# 3. Set the signal file name, i.e. the file where the signal strength/quality
# for each tile in the grid is stored

signalFileName <- system.file(path_root, 'SignalMeasure_MNO1.csv')

 # 4. Set the antennas file name

antennasFileName <- system.file(path_root, 'antennas.xml')

####    PREPARE DATA    ####

 # 5. Initial state distribution (prior)

nTiles <- gridParams$ncol * gridParams$nrow
initialDistr_RSS_uniform.vec <- initialDistr_SDM_uniform.vec <- rep(1 /
nTiles, nTiles)

 # 6. Get a list of detected devices

 devices <- getDeviceIDs(events)

 # 7. Get connections for each device

connections <- getConnections(events)
```

```
# 8. Emission probabilities are computed from the signal strength/quality file.
# In this case handoverType is strength then the emission model should be RSS.

emissionProbs_RSS <- getEmissionProbs(nrows = gridParams$nrow,
ncols = gridParams$ncol, signalFileName = signalFileName,
sigMin = simParams$conn_threshold, handoverType = simParams$connection_type[[1]],
emissionModel = "RSS", antennaFileName = antennasFileName)

 # We also try to use the emission model with the SDM assumption.

emissionProbs_SDM <- getEmissionProbs(nrows = gridParams$nrow,
ncols = gridParams$ncol, signalFileName = signalFileName,
sigMin = simParams$conn_threshold, handoverType = simParams$connection_type[[1]],
emissionModel = "SDM", antennaFileName = antennasFileName)

 # 9. Build joint emission probabilities for both options: RSS and SDM.

jointEmissionProbs_RSS <- getEmissionProbsJointModel(emissionProbs_RSS)

jointEmissionProbs_SDM <- getEmissionProbsJointModel(emissionProbs_SDM)

 # 10. Build the generic model for both cases and the a priori
 # uniform (by default)

model_RSS_uniform <- getGenericModel( nrows = gridParams$nrow,
ncols = gridParams$ncol, emissionProbs_RSS, initSteady = FALSE,
aprioriProb = initialDistr_RSS_uniform.vec)

model_SDM_uniform <- getGenericModel( nrows = gridParams$nrow,
ncols = gridParams$ncol, emissionProbs_SDM, initSteady = FALSE,
aprioriProb = initialDistr_SDM_uniform.vec)

 # 11. Fit models.

ll_RSS_uniform <- fitModels(length(devices), model_RSS_uniform, connections)

ll_SDM_uniform <- fitModels(length(devices), model_SDM_uniform, connections)

# The log likelihood is infinity for those connections that are impossible
# under the model SDM (ll_SDM_uniform).


 # 12. Make the checks and corresponding imputations to force the fit of SDM model.

### check1: connections incompatibles with emissionProbs are imputed as NA

check1_SDM <- checkConnections_step1(connections, emissionProbs_SDM)

check1_SDM$infoCheck_step1 connections_ImpSDM0 <-check1_SDM$connectionsImp

### check2: jumps incompatibles with emissionProbs are avoid by making time padding.

check2_SDM <- checkConnections_step2(emissionProbs = emissionProbs_SDM,
connections = connections_ImpSDM0, gridParams = gridParams)

connections_ImpSDM <- check2_SDM$connections_pad
```

```
 # 13. Fit models.

ll_SDM_uniform <- fitModels(length(devices), model_SDM_uniform, connections_ImpSDM)

 # 14. Build the joint model.

modelJ_RSS_uniform <- getJointModel( nrows = gridParams$nrow,
ncols = gridParams$ncol, jointEmissionProbs = jointEmissionProbs_RSS,
initSteady = FALSE, aprioriJointProb = initialDistr_RSS_uniform.vec)

modelJ_SDM_uniform <- getJointModel( nrows = gridParams$nrow,
ncols = gridParams$ncol, jointEmissionProbs = jointEmissionProbs_SDM,
initSteady = FALSE, aprioriJointProb = initialDistr_SDM_uniform.vec)

 # 15. Apriori probability of 2-to-1

Pii <- aprioriOneDeviceProb(simParams$prob_sec_mobile_phone, length(devices))

 # 16. Build a matrix of pairs of devices to compute duplicity probability

pairs4dup_RSS <- computePairs(connections, length(devices), oneToOne = TRUE)

pairs4dup_SDM <- computePairs(connections_ImpSDM, length(devices), oneToOne = TRUE)


####    COMPUTE DUPLICITY    ####


# 17. Compute duplicity probabilities using "1to1" method

duplicity1to1_RSS_uniform.dt <- computeDuplicityBayesian("1to1", devices,
pairs4dup_RSS, modelJ_RSS_uniform, ll_RSS_uniform, P1 = NULL, Pii=Pii)

duplicity1to1_SDM_uniform.dt <- computeDuplicityBayesian("1to1", devices,
pairs4dup_SDM, modelJ_SDM_uniform, ll_SDM_uniform, P1 = NULL, Pii=Pii)
```

`fitModels` *Fits the HMM model for each device.*

Description

Fits the HMM model for each device using the `fit()` function from **destim** package. The computations are done in parallel to reduce the running time using all the available cores. This function creates a cluster of working nodes, splits the devices equally among the working nodes and assigns a partition of devices to each node in the cluster. For Unix-like operating systems, this functions uses a "FORK" cluster while for Windows it uses a "SOCK" cluster.

Usage

```
fitModels(ndevices, model, connections)
```

Arguments

ndevices	The number of devices.
model	The HMM model returned by getGenericModel() function. This model is fitted for each device.
connections	A matrix whose elements are the antenna ID where a device is connected at every time instant. This matrix is returned by getConnections() function.

Value

A vector of log likelihoods computed using the fitted model for each device.

---

getConnections *Builds a matrix object containing the IDs of the antennas to which devices are connected.*

Description

Builds a matrix object containing the IDs of the antennas to which devices are connected. The number of rows equals the number of devices and the number of columns equals the number of time instants when the network events were recorded. An element [i,j] in the returned matrix equals the ID of the antenna where the mobile device with index i in the ordered list of device IDs (returned by getDeviceIDs()) is connected at the time instant with index j in the sequence of time instants when the network events were recorded.

Usage

```
getConnections(events)
```

Arguments

events	A data.table object returned by readEvents() function.

Value

A matrix object with the antenna IDs where devices are connected for every time instants in the events file. If a device is not connected to any antenna at a time instant, the corresponding element in the matrix will have the value NA.

---

getDeviceIDs *Builds a vector with the IDs of the mobile devices.*

Description

Builds a vector with the IDs of the mobile devices by taking the unique values from the events data set.

Usage

```
getDeviceIDs(events)
```

Arguments

events	A data.table object that contains the events generated by the mobile network. It is returned by the readEvents() function. The device IDs are on the second column of this data.table object.

Value

A vector with the IDs of the mobile devices detected by the network.

---

getEmissionProbs *Builds the emission probabilities for the HMM used to estimate the posterior location probabilities.*

Description

Builds the emission probabilities needed for the HMM used to estimate the posterior location probabilities. In case of using simulated data, these probabilities are build using the signal strength or signal quality saved by the simulation software for each tile in the grid.

Usage

```
getEmissionProbs(
  nrows,
  ncols,
  signalFileName,
  sigMin,
  handoverType = "strength",
  simulatedData = TRUE,
  emissionModel = NULL,
  antennaFileName = NULL
)
```

Arguments

signalFileName	The name of the .csv file that contains the signal strength/quality for each tile in the grid. This file is one of the outputs of the data simulator. The data are organized as a matrix with the number of rows equals to the number of antennas and the the following columns: Antenna ID,Tile 0,Tile 1,... Tile (N-1). On the first column there are the antenna IDs and on the rest of the columns the corresponding signal strength/quality for each tile in the grid.
sigMin	The minimum value of the signal strength/quality that allow a connection between a device and an antenna.
handoverType	The handover mechanism used by the mobile network. It could have two values: "strength" or "quality". It should match the types of the values in the signal file, otherwise the results are impredictible.
simulatedData	If TRUE, the input data provided to this function come from the simulator otherwise the data come from a real mobile network.

`emissionModel` A parameter that can take two values: "RSS" or "SDM". It indicates how the emission probabilities are computed. This parameter is needed to force computing the emission probabilities with a "wrong" model. For example, the signal file contains the values of the signal strength, the `handoverType` parameter is set to 'strength' but the `emissionModel` is set to "SDM", the values of the signal strength are transformed in signal quality and the emission probabilities are computed using the signal quality. Such a combination should never be used in practice but it is allowed only for demonstrative purposes: it can be use to demonstrate that if the emission probabilities are not correctly computed then the resulted duplicity probabilities are wrong.

`antennaFileName` This parameter is needed to read the technical parameters of antennas.These parameters are used to transform the signal strength in signal quality and the other way around. They are needed only in the case the emission probabilities are computed using the signal quality when the `handoverType` is "strength" or when they are computed using signal quality when the `handoverType` is "quality".

`nrow` the number of rows in the grid. It can be obtained by calling `readGridParams()`.

`ncol` the number of columns in the grid. It can be obtained by calling `readGridParams()`.

Value

Returns a Matrix object with the emission probabilities for the HMM. The number of rows equals the number of tiles in the grid and the number of columns equals the number of antennas. An element (i,j) of this matrix corresponds to the probability of a device being in tile i to be connected to antenna j. The row names of the matrix are the tile indexes and the column names are the antenna IDs.

`getEmissionProbsJointModel` *Builds the emission probabilities for the joint HMM.*

Description

Builds the emissions probabilities needed for the joint HMM used to estimate the posterior location probabilities.

Usage

```
getEmissionProbsJointModel(emissionProbs)
```

Arguments

`emissionProbs` the emission probabilities (the location probabilities) computed by calling `getEmissionProbs()` for each individual device.

Value

Returns a matrix with the joint emission probabilities for the HMM. The number of rows equals the number tiles and the number of columns equals the number of combinations between antenna IDs. Before the combination between antenna IDs are build, the NA value is added to the list of antenna IDs. An element in this matrix represents the transition probability from an antenna to another, computed for each tile in the grid.

---

getGenericModel *Builds the generic HMM model.*

Description

Builds the generic HMM model using the emission probabilities given by `getEmissionProbs()`.

Usage

```
getGenericModel(
  nrows,
  ncols,
  emissionProbs,
  initSteady = TRUE,
  aprioriProb = NULL
)
```

Arguments

`nrows`	Number of rows in the grid.
`ncols`	Number of columns in the grid.
`emissionProbs`	A matrix with the event location probabilities. The number of rows equals the number of tiles in the grid and the number of columns equals the number of antennas. This matrix is obtained by calling `getEmissionProbs()` function.
`initSteady`	If TRUE the initial apriori distribution is set to the steady state of the transition matrix, if FALSE the apriori distribution should be given as a parameter.
`aprioriProb`	The apriori distribution for the HMM model. It is needed only if `initSteady` is FALSE.

Value

Returns an HMM model with the initial apriori distribution set to the steady state of the transition matrix or to the value given by `aprioriProb` parameter.

getJointModel *Builds the joint HMM model.*

Description

Builds the joint HMM model using the emmission probabilities given by `getEmissionProbsJointModel()`.

Usage

```
getJointModel(
  nrows,
  ncols,
  jointEmissionProbs,
  initSteady = TRUE,
  aprioriJointProb = NULL
)
```

Arguments

`nrows`	Number of rows in the grid.
`ncols`	Number of columns in the grid.
`jointEmissionProbs`	A (sparse) matrix with the joint event location probabilities. The number of rows equals the number of tiles in the grid and the number of columns equals the number of antennas. This matrix is obtained by calling `getEmissionProbsJointModel`.
`initSteady`	If TRUE the initial apriori distribution is set to the steady state of the transition matrix, if FALSE the apriori distribution should be given as a parameter.
`aprioriJointProb`	The apriori distribution for the HMM model. It is needed only if initSteady is FALSE.

Value

Returns an HMM model with the initial apriori distribution set to the steady state of the transition matrix or to the value given by `aprioriJointProb` parameter.

modeDelta *Returns the mode of delta distribution.*

Description

Returns the mode of the deltaX or deltaY distribution.

Usage

```
modeDelta(deltaDistribution)
```

Arguments

`deltaDistribution`
: A data.table object that could be Delta X or Delta Y distribution. The table has two columns: delta and p. It is obtainded from calling `buildDeltaProb()` function.

Value

Rhe mode of the delta distribution.

---

readCells *Reads the coverage areas of antennas.*

Description

Reads the coverage areas of antennas from a .csv file.

Usage

```
readCells(cellsFileName, simulatedData = TRUE)
```

Arguments

`cellsFileName`
: It is the name of the file where the coverage areas of antennas are to be found. The data have two columns, the first one is the antenna ID and the second one is a WKT string representing a polygon (i.e. it should start with the word POLYGON) which is the coverage area of the corresponding antenna. This area is also called the antenna cell.

`simulatedData`
: If TRUE it means that the file with the coverage areas is produced by the data simulator

Value

A data.table object with 2 columns: the antenna ID and an `sp` geometry object which is the coverage area of the corresponding antenna.

---

readEvents *Reads the network events file.*

Description

Reads the network events file. This file can come from the network simulator or it ca be a file with real mobile network events provided by an MNO.

Usage

```
readEvents(eventsFileName, simulatedData = TRUE)
```

Arguments

eventsFileName	The name of the file with the network events to be used. Depending on the parameter simulatedData it could be a .csv file coming from the simulation software or from a real MNO. In case the file comes from the simulation software it should contain following columns: time,antennaID,eventCode,deviceID,x,y,tile. Only the first 4 columns are used, the rest are ignored.
simulatedData	If TRUE it means that the input data are simulated data, otherwise the data come from a real MNO.

Value

Returns a data.table object that contains the events generated by the mobile network. The number of rows equals the number of connection events recorded by the network. The returned object has the following columns: time,deviceID,eventCode,antennaID,x,y,tile,obsVar. obsVar stands for observed variable and is a concatenation between the antenna ID and the event code.

---

readGridParams *Reads the parameters of the grid.*

Description

Reads the parameters of the grid overlapped on the geographical of interest from a .csv file.

Usage

```
readGridParams(gridFileName)
```

Arguments

gridFileName	The name of the file with the grid parameters. This file could be the one generated by the simulation software or can be created with any text editor. The grid file generated by the simulation software has the following columns: Origin X,Origin Y,X Tile Dim,Y Tile Dim,No Tiles X,No Tiles Y. We are interested only in the number of rows and columns and the tile size on OX and OY axes. Therefore, the file provided as input to this function should have at least the following 4 columns: No Tiles X ,No Tiles Y,X Tile Dim,Y Tile Dim.

Value

Returns a list with the following items: nrow - the number of rows, i.e. the number of tiles in a column of the grid, ncol - the number of columns, i.e. the number of tiles in a row of the grid, tileX - the dimension of a tile on OX axis, tileY - the dimension of a tile on OY axis.

readPostLocProb *Reads a file with the posterior location probabilities.*

Description

Reads a .csv file with the posterior location probabilities. Each row of the file corresponds to a tile and each column corresponds to a time instant.

Usage

```
readPostLocProb(path, prefixName, deviceID)
```

Arguments

path	The path to the location where the posterior location probabilities are stored. The file with the location probabilities should have the name postLocDevice_ID.csv where ID is replaced with the device ID.
prefixName	The file name prefix. The whole file name is composed by a concatenation of prefixName, _ and deviceID.
deviceID	The device ID for which the posterior location probabilities are read.

Value

A Matrix object with the posterior location probabilities for the device with ID equals to deviceID. A row corresponds to a tile and a column corresponds to a time instant.

readSimulationParams *Reads the parameters of the simulation used to generate a data set.*

Description

Reads the parameters of the simulation used to generate a data set from an .xml file used by the simulation software. The following parameters are needed by this package: the connection threshold which is the minimum signal strength/quality that can be used by a mobile device to connect to an antenna and the probability of having a two mobile devices.

Usage

```
readSimulationParams(simFileName)
```

Arguments

simFileName	The name of the file used to define a simulation scenario. It is the file that was provided as an input for the simulation software.

Value

A list with all the parameters read from the file: start_time, end_time, time_increment, time_stay, interval_between_stays, prob_sec_mobile_phone, conn_threshold.

`tileEquivalence` *Transforms the tiles indices from the notation used by the simulation software to the one used by the* **raster** *package.*

Description

In order to perform the population estimations, the area of interest is overlapped with a rectangular grid of tiles. Each tile is a rectangle with predefined dimensions. This function is a utility function which transform the tiles indexes from the numbering system used by the simulation software to the one used by the **raster** package. The simulation software uses a notation where the tile with index 0 is the bottom left tile while the **raster** package uses another way to number the tiles, tiles being numbered starting with 1 for the upper left tile.

Usage

```
tileEquivalence(nrows, ncols)
```

Arguments

`nrow`	Number of rows in the grid overlapping the area of interest.
`ncol`	Number of columns in the grid overlapping the area of interest.

Value

Returns a data.frame object with two columns: on the first column are the tile indexes according to the **raster** package numbering and on the second column are the equivalent tile indexes according to the simulation software numbering.

Index