Package 'ltbiScreenLite'

January 17, 2019

Type Package

Title ltbiScreenLite

Version 0.1.0

Author N Green

Maintainer Nathan Green <ngreen1@ic.ac.uk>

Description Stripped-down version of LTBIscreeningproject package.

License What license is it under?

Encoding UTF-8

LazyData true

RoxygenNote 6.1.1

R topics documented:

`activetb_qaly_cost` *activetb_qaly_cost*

Description

For the Population model, calculate various QALYs and costs accounting for active TB progression in non-cured cohort subset.

Usage

```
activetb_qaly_cost(dectree_res, interv, cohort, folders = NA)
```

Arguments

`dectree_res`	Output of `parallel_decision_tree()`. This contains the probability of being cured of LTBI via screening.
`interv`	list of fixed model run parameter values
`cohort`	dataframe of individual level data
`folders`	list of strings locations for data and plots

Value

- QALY.statusquo: For each scenario a vector of total QALYs without screening programme, length number of sims. These are all the same because population QALYs are not varied for the cohort.
- QALY.screened: For each scenario a vector of total QALYs with screening programme, length number of sims.
- E_cost_screened: For each scenario single expected cost with screening programme.

- cost.screened_person: For each scenario a vector of QALYs per person with screening programme, length number of sims.
- cost.statusquo_person: For each scenario a vector of costs per person without screening programme, length number of sims.
- cost_incur: For each scenario a vector of incurred costs by screening programme, length number of sims.
- cost.statusquo: For each scenario a vector of total costs without screening programme, length number of sims. The are not identical because TB costs are randomly sampled.
- cost.screened: For each scenario a vector of total costs with screening programme, length number of sims.
- E_QALY_screened: For each scenario single expected QALYs with screening programme.
- QALY.screened_person: For each scenario a vector of QALYs per person with screening programme, length number of sims.
- QALY.statusquo_person: For each scenario a vector of QALYs per person without screening programme, length number of sims. These are all the same.
- QALYgain: For each scenario a vector of total QALYs gained with screening programme as the difference between screening and status-quo, length number of sims.
- cost_incur_person: For each scenario a vector of total costs incured per person with screening programme as the difference between screening and status-quo, length number of sims.
- E_cost_incur: For each scenario the expectedd total cost incured with screening programme as the difference between screening and status-quo.
- E_cost_incur_person: For each scenario the expected total cost incured per person with screening programme as the difference between screening and status-quo.
- QALYgain_person: For each scenario a vector of total QALY gained per person with screening programme as the difference between screening and status-quo.
- E_QALYgain: For each scenario the expected total QALY gained with screening programme as the difference between screening and status-quo.
- E_QALYgain_person: For each scenario the expected total QALy gained per person with screening programme as the difference between screening and status-quo.

`branch_unif_params` *branch_unif_params constructor*

Description

Define properties and assign `branch_unif_params` class.

Usage

```
branch_unif_params(pmin, pmax, name)
```

Arguments

`pmin`	Minimum probability
`pmax`	Maximum probability
`name`	Node label

Value

list

`calc_QALY_tb` *Calculate QALYs for active TB cases*

Description

Calculate the QALYs for each active TB individuals.

Usage

```
calc_QALY_tb(intervals = NA, utility, age, start_delay = NA,
  discount_rate = 0.035, ...)
```

Arguments

`intervals`	Time intervals for each utility
`utility`	(list) Utility value of non-diseased individual e.g. 1. Utility value of diseased individual
`age`	Ages in years; vector numeric
`start_delay`	What time delay to time origin, to shift discounting to smaller values
`discount_rate`	default 3.5% per annum
`...`	Additional arguments

Details

For each of 3 alternatives:

- diseasefree: to all-cause death
- fatality: case-fatality 12 months from notification
- cured: successfully treated for LTBI

Assume that death if it happens is within the first year of active TB. Assume that active TB cases when treated and survive first year are ~~fully cured~~.

Consider person-perspective (death) or NHS-perspective (exit uk) by defining the particular time-to-event end point.

Value

list of diseasefree, death, cured QALYs

`ce_default` *ce_default*

Description

Uses the first column of the status-quo matrices for all status-quo comparisons.

Usage

```
ce_default(ce0, ce1)
```

Arguments

`ce1`

`combine_popmod_dectree_res` *combine_popmod_dectree_res*

Description

Combine cost and QALY outputs from decision tree model and population model for overall cost-effectiveness samples.

Usage

```
combine_popmod_dectree_res(cohort, interv, popmod_res, dectree_res,
  folders = NA)
```

Arguments

`cohort`	individual level data; dataframe
`interv`	fixed model run inputs; list
`popmod_res`	output of `activetb_qaly_cost()`; list
`dectree_res`	output of `parallel_decision_tree()`; list
`folders`	list of ouput folder locations

Value

list of cost-effective statistics:

- ce0: marginal status-quo. Costs and QALYs of each sim.
- ce1: marginal intervention. Costs and QALYs of each sim.
- ce_default: non-incremental cost-effectiveness i.e. dataframe with first column status-quo.
- ce_incr: incremental cost-effectivness i.e. dataframe with first column 0 and other screening cost minus status-quo.

See Also

`parallel_decision_tree`, `activetb_qaly_cost` `costeff_stats` *Cost-effectiveness Statistics*

Description

For a scenario the population model with active TB cases.

Usage

```
costeff_stats(scenario_dat, interv_QALY, interv_cost, pop_year)
```

Arguments

`scenario_dat`	list
`interv_QALY`	list of `scenario_QALY()` output
`interv_cost`	list of `scenario_cost()` output
`pop_year`	integer

Value

list

- QALY.statusquo
- QALY.screened
- E_cost_screened: mean average
- cost.screened_person
- cost.statusquo_person
- cost_incur
- cost.statusquo
- cost.screened
- E_QALY_screened: mean average
- QALY.screened_person
- QALY.statusquo_person
- QALYgain
- cost_incur_person
- E_cost_incur: mean average
- E_cost_incur_person: mean average
- QALYgain_person
- E_QALYgain: mean average
- E_QALYgain_person: mean average

`cp_in_data_to_out_dir` *Copy input data to output folder*

Description

Copy input data to output folder

Usage

```
cp_in_data_to_out_dir(file_names, to_dir)
```

Arguments

`file_names`	vector of text strings
`to_dir`	

`create_and_save_policies` *create_and_save_policies*

Description

Given the input argument creates a sort of grid array, i.e all permutations, version of an environment which behaves like a list object.

Usage

```
create_and_save_policies(incidence_list, endpoints, LTBI_test, treatment)
```

Arguments

`incidence_list`	WHO incidence in country of origin groups to target screening
`endpoints`	when to stop using costs and QALYs at time of exit or death
`LTBI_test`	TSPOT or QFT
`treatment`	6 months or 3 months LTBI regimens

Value

list of policies as inputs for the model.

`create_and_save_scenarios`
Create and save scenarios

Description

Read in an Excel workbook consisting of cost and probability (p) sheets. This is converted to a list of dataframes for model inputs and a long flat array to easily inspect by eye. Saved to file.

Usage

```
create_and_save_scenarios(file_tag)
```

Arguments

`file_tag`	Trailing part of the Excel file name to identify specific sets of scenarios.

Value

none (save to project data folder)

`decision_tree_cluster` *Decision tree*

Description

Calculate decision tree expected costs and QALY loss for each simulation.

Usage

```
decision_tree_cluster(params, N.mc = 2,
  cost_dectree = "data/osNode_cost_2009.Rds",
  health_dectree = "data/osNode_health_2009.Rds", out_datatree = FALSE)
```

Arguments

`params`	an element of a scenario list with probabilities and costs to substitue into decision tree; long format array
`N.mc`	number of simulations; integer
`cost_dectree`	data.tree saved as Rds file names (string); default to package folder
`health_dectree`	data.tree saved as Rds file names (string); default to package folder
`out_datatree`	Output full datatree object? This may be large. Will also save to csv for checking; default: FALSE

Value

list

- `mc_cost`: each simulation total expected cost
- `mc_health`: each simulation total expected QALY loss
- `subset_pop`: cohort population sizes and probabilities at specific node or groups of nodes. Specifically calculates for individuals with LTBI since these are the subset of particular interest in term of cure; dataframe headings are
  - LTBI_pre
  - tests
  - positive
  - startTx
  - completeTx
  - cured
  - LTBI_post
  - p_LTBI_to_cured
  - LTBI_tests
  - LTBI_positive
  - LTBI_startTx
  - LTBI_completeTx
- `osNode.cost`: data.tree object
- `osNode.health`: data.tree object
- `call`: original call with arguments
- `N.mc`: number of Monte-Carlo simulations

`diroutput` *form name of output folder*

Description

Create permanent output folder

Usage

```
diroutput(policy_name, interv)
```

Arguments

`interv`

`expected_cost_QALY` *expected_cost_QALY*

Description

Calculate the average total cost using mean unit costs.

Usage

```
expected_cost_QALY(cohort, means)
```

Arguments

`means`	

`filter_cohort_by_policy` *Create policy cohort*

Description

Filter individuals by policy definition.

Usage

```
filter_cohort_by_policy(cohort_in, policy_name, interv)
```

Arguments

`cohort_in`	total sample
`interv`	list of conditions

Value

cohort

`handle_try_error` *handle_try_error*

Description

handle_try_error

Usage

```
handle_try_error(try_out)
```

Arguments

`try_out`	

`interv_constructor` *interv_constructor*

Description

Simple list making function with defaults.

Usage

```
interv_constructor(N.mc = 1, cluster = FALSE, use_discount = TRUE,
  no_students = FALSE, force_everyone_stays = FALSE,
  screen_with_delay = TRUE, MAX_SCREEN_DELAY = 5, FUP_MAX_YEAR = 100,
  screen_age_range = 18:35, year_cohort = "2009",
  incidence_grps_screen = c("(0,50]", "(50,150]", "(150,250]",
  "(250,350]", "(350,1e+05]"), min_screen_length_of_stay = 0,
  ENDPOINT_cost = "death", ENDPOINT_QALY = "death")
```

Arguments

`N.mc`	Global fixed constant; default 1
`use_discount`	Global fixed constant; TRUE/FALSE.
`screen_with_delay`	Rather than screen _everyone_ on entry screen at random 0-5 years from port of entry
`FUP_MAX_YEAR`	Time horizon for active TB progression; default 100
`year_cohort`	2012 is most recent complete year; largest cohort, corresponds with Pareek () LTBI risk
`incidence_grps_screen`	Modified in the deterministic sensitivity analysis but set default values
`min_screen_length_of_stay`	Modified in the deterministic sensitivity analysis but set default values
`ENDPOINT_cost`	Modified in the deterministic sensitivity analysis but set default values; exit uk or death
`ENDPOINT_QALY`	Modified in the deterministic sensitivity analysis but set default values; exit uk or death

`list_to_BCEA` *list_to_BCEA*

Description

transform to BCEA package input format

Usage

```
list_to_BCEA(scenario_list, discount = 1)

list_to_BCEA_incr(scenario_list, discount = 1)
```

Arguments

`discount`

`make_ce0` *make_ce0*

Description

make_ce0

Usage

```
make_ce0(popmod_res)
```

Arguments

`popmod_res`

`make_ce1` *make_ce1*

Description

make_ce1

Usage

```
make_ce1(popmod_res, t_dectree, sdiscount)
```

Arguments

`sdiscount`

`make_incremental_ce` *make_incremental_ce*

Description

For plotting purposed in particular, we want the cost and QALY output data to be in the same format as used by the `BCEA` package.

Usage

```
make_incremental_ce(popmod_res, t_dectree, sdiscount, folders = NA)
```

Arguments

`popmod_res`	`activetb_qaly_cost()` output
`t_dectree`	list of `mc_cost` and `mc_health` from `parallel_decision_tree()` output
`sdiscount`	average discounting due to delay to starting screening
`folders`	list of string locations

Value

list of incremental e and c

`mean_QALYs` *mean_QALYs*

Description

Contact tracing

Usage

```
mean_QALYs(cohort, p_contact_tracing)
```

Arguments

`p_contact_tracing`

Value

list status-quo and disease-free expected QALY

`my_ToDataFrameTable` *my_ToDataFrameTable*

Description

This is the same as the same named function in `data.tree` except it is not filtered by leaf.

Usage

```
my_ToDataFrameTable(x, ..., pruneFun = NULL)
```

Arguments

`pruneFun`

See Also

ToDataFrameTable

`my_ToDataFrameTypeCol` *my_ToDataFrameTypeCol*

Description

This is the same as the same named function in `data.tree` except it is not filtered by leaf.

Usage

```
my_ToDataFrameTypeCol(x, ..., type = "level", prefix = type,
  pruneFun = NULL)
```

Arguments

`pruneFun`

See Also

ToDataFrameTypeCol

`notif_cost` *Combined cost for each TB case*

Description

including secondary infections. with discounting

Usage

```
notif_cost(cost, probs, num_contacts, discounts)
```

Arguments

`cost`	vector
`probs`	vector
`num_contacts`	vector, per case
`discounts`	at time of notification; vector, per case

`parallel_decision_tree` *Parallel cost-effectiveness decision tree*

Description

Based on code from here: https://www.r-bloggers.com/how-to-go-parallel-in-r-basics-tips/

Usage

```
parallel_decision_tree(scenario_params, interv, folders,
  out_datatree = FALSE)
```

Arguments

`scenario_params`	list of dataframes of cost and probability values
`interv`	list of fixed policy parameters
`folders`	list os strings of locations for data and plots
`out_datatree`	include in return and save data.tree object; default: FALSE

Value

List of `decision_tree_cluster` outputs for each scenario

`rcontact_tracing_costs` *randomly sample constact tracing costs*

Description

randomly sample constact tracing costs

Usage

```
rcontact_tracing_costs(unit_cost)
```

Arguments

`unit_cost`	named list

Value

named vector matching `p_contact_tracing`: contact, aTB_Dx, aTB_Tx, LTBI_DxTx, index

`rows_first_n_ids` *rows_first_n_ids*

Description

Finds the rows corresponding to the first `n` individuals by ascending id numbers.

Usage

```
rows_first_n_ids(id_avoid, prop_avoid)
```

Arguments

`id_avoid`	IDs, may have gaps/missing numbers
`prop_avoid`	probability

Value

logical vector length ids

See Also

sample_avoid_lg, rsample_n_ids

`rsample_n_ids` *rsample_n_ids*

Description

rsample_n_ids

Usage

```
rsample_n_ids(id_avoid, prop_avoid)
```

Arguments

`prop_avoid`	probability

See Also

sample_avoid_lg, rows_first_n_ids

`run_final_message` *run_final_message*

Description

run_final_message

Usage

```
run_final_message(run)
```

Arguments

`run`	

`run_model` *Run model*

Description

Wrapper around `run_policy`.

Usage

```
run_model(cohort_data = NA, make_plots = FALSE, sink_out = FALSE)
```

Arguments

`cohort_data`	individual level data
`make_plots`	TRUE/FALSE
`sink_out`	output console to text file? Default: FALSE

See Also

`run_policy`

`run_policy` *run_policy*

Description

A single policy simulation (for multiple scenarios) of cost-effectiveness model.

Usage

```
run_policy(cohort = NA, make_plots = FALSE)
```

Arguments

`cohort`	individual level; default: NA
`make_plots`	TRUE/FALSE

Value

empty

See Also

`run_model`

`sample_avoid_lg` *sample_avoid_lg*

Description

sample_avoid_lg

Usage

```
sample_avoid_lg(id_avoided_tb, prop_avoided, ordered)
```

Arguments

`prop_avoided`	probability
`ordered`	random or ordered; TRUE/FALSE

See Also

rows_first_n_ids, rsample_n_ids

sample_subset_pop_dectree *sample_subset_pop_dectree*

Description

Iteratively randomly samples probabilities and then calculates subset sizes

Usage

```
sample_subset_pop_dectree(osNode, n = 1, sample_p = TRUE)
```

Arguments

`osNode`	data.tree object
`n`	Sample size
`sample_p`	Random sample TRUE/FALSE; default: TRUE

Value

matrix

save_session_info *Save session info*

Description

Text file includes package dependencies, versions, etc.

Usage

```
save_session_info(file)
```

Arguments

`file`	string location

`scenario_cost` *Calculate total active TB cost of a scenario*

Description

Calculate total active TB cost of a scenario

Usage

```
scenario_cost(endpoint, unit_cost, probs_contact, cohort, prop_avoided,
  ordered = TRUE)
```

Arguments

`endpoint`	`death` or `exit uk` i.e. time horizon
`unit_cost`	Diagnosis and treatment cost distributions; list
`probs_contact`	Proportions of individuals in subsets
`cohort`	individual level dataframe. nrow total number of TB cases in EWNI and after exit
`prop_avoided`	p_LTBI_to_cured; single numeric
`order`	default TRUE

Value

list total cost for `statusquo` and `screened`; numeric

`scenario_QALY` *Calculate total QALYs of a scenario*

Description

Calculate total QALYs of a scenario

Usage

```
scenario_QALY(prop_avoided, endpoint, cohort, ordered = TRUE)
```

Arguments

`prop_avoided`	probability cured of LTBI by screening
`endpoint`	'death' or 'exit uk' for time horizon
`cohort`	Individual level data
`ordered`	Should individuals have a fixed order when avoiding TB This is useful for reproducability and ensures that a higher proportion avoiding TB is always better; default: TRUE

Value

list of status-quo and screened total QALYs

`scenario_QALYloss` *scenario_QALYloss*

Description

Splits output also into due to morbidity and mortality.

Usage

```
scenario_QALYloss(prop_avoided, endpoint, cohort)
```

Arguments

`prop_avoided`	proportion LTBI cured via screening
`endpoint`	exit uk or death time horizon
`cohort`	individual level data

Value

list

- statusquo_mortality
- statusquo_morbidity
- screened_mortality
- screened_morbidity
- statusquo_mort_pp
- statusquo_morb_pp
- screened_mort_pp
- screened_morb_pp

`screened_cohort_cost` *Calculate Potentially Screened Cohort Costs*

Description

Substract the avoided cost of those successfully screened from status-quo cost.

Usage

```
screened_cohort_cost(n.diseasefree, cost.statusquo, unit_cost_case)
```

Arguments

`n.diseasefree`	Number of disease-free individuals
`cost.statusquo`	Cost under status-quo
`unit_cost_case`	Unit cost of detect and treat an active TB case

Value

Total cost for potentially screened cohort

See Also

`screened_cohort_QALYs`

`screen_discount` *screen_discount*

Description

Average discount cost and QALYs in decision tree due to delayed start of screening from port of entry.

Usage

```
screen_discount(cohort, discount_rate = 0.035)
```

Arguments

`cohort`	individual level data
`discount_rate`	default: 3.5% per annum

Value

single numeric proportion between 0 and 1

`setup_folders` *Setup folders*

Description

Setup folders

Usage

```
setup_folders(policy_name, interv)
```

Arguments

`policy_name`	String
`interv`	List of model run constants

Value

List of folder locations

set_branch_uniform_params *set_branch_uniform_params*

Description

set_branch_uniform_params

Usage

```
set_branch_uniform_params(vals, osNode)

## Default S3 method:
set_branch_uniform_params(vals, osNode)

## S3 method for class 'branch_unif_params'
set_branch_uniform_params(vals, osNode)

## S3 method for class 'test'
set_branch_uniform_params(vals, osNode)
```

Arguments

`osNode`

set_policy *set_policy*

Description

Set the intervention parameter values within an environment.

Usage

```
set_policy(policy_name, interv)
```

Arguments

`policy_name`	string
`interv`	list of fixed model parameter values

`subset_pop_dectree` *Subset populations of decision tree*

Description

Specific to the LTBI screening model, this gives the total probabilities of particular state on the pathway by summing across nodes, using pathprobs.

Usage

```
subset_pop_dectree(osNode)
```

Arguments

`osNode`	data.tree object

Value

data.frame of probabilities

`test` *TB test constructor*

Description

Define properties and assign `test` class.

Usage

```
test(sens, spec)
```

Arguments

`sens`	test sensitivity
`spec`	test specificity

Value

list

`total_contact_tracing_cost`
total_contact_tracing_cost

Description

For an index case.

Usage

```
total_contact_tracing_cost(num_contacts, cost, probs)
```

Arguments

`num_contacts`	vector, per case
`probs`	vector
`costs`	vector

Value

vector, per case

Index