Package ‘datafsm’

October 13, 2022

Title Estimating Finite State Machine Models from Data

Version 0.2.4

Date 2021-05-28

Description Automatic generation of finite state machine models of dynamic decision-making that both have strong predictive power and are interpretable in human terms. We use an efficient model representation and a genetic algorithm-based estimation process to generate simple deterministic approximations that explain most of the structure of complex stochastic processes. We have applied the software to empirical data, and demonstrated it's ability to recover known data-generating processes by simulating data with agent-based models and correctly deriving the underlying decision models for multiple agent models and degrees of stochasticity.

URL `https://jonathan-g.github.io/datafsm/`, `https://github.com/jonathan-g/datafsm`

BugReports `https://github.com/jonathan-g/datafsm/issues`

Depends R (>= 4.0), methods (>= 4.0), stats (>= 4.0)

License MIT + file LICENSE

LazyData true

LinkingTo Rcpp (>= 1.0)

Suggests doParallel (>= 1.0), foreach (>= 1.5), testthat (>= 3.0), diagram (>= 1.6), knitr (>= 1.33), rmarkdown (>= 2.8), pander (>= 0.6), dplyr (>= 1.0), tidyr (>= 1.0), purrr (>= 0.3)

Imports caret (>= 6.0), GA (>= 3.2), Rcpp (>= 1.0)

VignetteBuilder knitr

RoxygenNote 7.1.1

Encoding UTF-8

Language en-US

NeedsCompilation yes

Author John J. Nay [aut],
Jonathan M. Gilligan [cre, aut]
(<https://orcid.org/0000-0003-1375-6686>)

Maintainer Jonathan M. Gilligan <jonathan.gilligan@vanderbilt.edu>

Repository CRAN

Date/Publication 2021-05-29 23:20:05 UTC

R topics documented:

`action_vec` *Extracts slot of action_vec*

Description

Extracts slot of action_vec

Usage

```
action_vec(x)
```

Arguments

`x`	S4 ga_fsm object

`add_interact_num` *Add interaction numbers for panel data*

Description

`add_interact_num` takes in data and returns a vector of interactions

Usage

```
add_interact_num(d)
```

Arguments

`d`	data.frame of panel data

Value

Returns a vector specifying interactions

`best_performance` *Extracts performance*

Description

Extracts performance

Usage

```
best_performance(x)
```

Arguments

`x`	S4 ga_fsm object

`build_bitstring` *Builds Bitstring*

Description

`build_bitstring` creates a bitstring from an action vector, state matrix, and number of actions.

Usage

```
build_bitstring(action_vec, state_mat, actions)
```

Arguments

`action_vec`	Numeric vector indicating what action to take for each state.
`state_mat`	Numeric matrix with rows as states and columns as predictors.
`actions`	Numeric vector length one with the number of actions.

Value

Returns numeric vector bitstring.

`compare_fsm` *Compares FSMs*

Description

`compare_fsm` uses a specified distance measure to compare FSMs.

Usage

```
compare_fsm(users, gas, comparison = "manhattan")
```

Arguments

`users`	Numeric vector or numeric matrix with a predefined FSM
`gas`	Numeric vector or numeric matrix with an evolved FSM
`comparison`	Character string of length one with either "manhattan", "euclidean", or "binary".

Details

Compares a user-defined FSM to a decoded estimated FSM. If you have have FSMs that may have values in the matrices that are not all simple integers, you can use the distance metric that is most appropriate. Euclidean does sqrt(sum((x_i - y_i)^2)) - the L2 norm. Manhattan takes abs diff between them - the L1 norm. Binary treats non-zero elements as "on" and zero elements as "off" and distance is the proportion of bits in which only one is on amongst those in which at least one is on.

Value

Numeric vector of length one for the distance between the two supplied FSMs, calculated according to the comparison argument.

`datafsm` *datafsm: A package for estimating FSM models.*

Description

It relies on the **GA** package: Luca Scrucca (2013). GA: A Package for Genetic Algorithms in R. Journal of Statistical Software, 53 (4), 1-37. URL `https://www.jstatsoft.org/v53/i04/`.

datafsm functions

`datafsm`'s main function for estimating a fsm decision model:

1. `evolve_model`

`datafsm`'s helper functions:

1. `evolve_model_cv`
2. `var_imp`
3. `decode_state_mat`
4. `decode_action_vec`
5. `fitnessCPP`
6. `build_bitstring`
7. `compare_fsm`

Author(s)

Maintainer: Jonathan M. Gilligan `<jonathan.gilligan@vanderbilt.edu>` (ORCID)

Authors:

- John J. Nay `<john.j.nay@gmail.com>`

See Also

Useful links:

- `https://jonathan-g.github.io/datafsm/`
- `https://github.com/jonathan-g/datafsm`
- Report bugs at `https://github.com/jonathan-g/datafsm/issues`

`decode_action_vec` *Decodes Action Vector*

Description

`decode_action_vec` decodes action vector.

Usage

```
decode_action_vec(string, states, inputs, actions)
```

Arguments

`string`	Numeric (integer) vector of only 1's and 0's.
`states`	Numeric vector with the number of states, which is the number of rows.
`inputs`	Numeric vector length one, with the number of columns.
`actions`	Numeric vector with the number of actions. Actions (and states) determine how many binary elements we need to represent an element of the action (or state) matrix.

Details

This function takes a solution string of binary values in Gray representation, transforms it to a decimal representation, then puts it in matrix form with the correct sized matrices, given the specified numbers of states, inputs, and actions.

Value

Returns numeric (integer) vector.

`decode_state_mat` *Decodes State Matrix*

Description

`decode_state_mat` decodes state matrix.

Usage

```
decode_state_mat(string, states, inputs, actions)
```

Arguments

`string`	Numeric vector.
`states`	Numeric vector with the number of states, which is the number of rows.
`inputs`	Numeric vector length one, with the number of columns.
`actions`	Numeric vector with the number of actions. Actions (and states) determine how many binary elements we need to represent an element of the action (or state) matrix.

Details

This function takes a solution string of binary values in Gray representation, transforms it to a decimal representation, then puts it in matrix form with the correct sized matrices, given the specified numbers of states, inputs, and actions.

Value

Returns numeric (integer) matrix.

`degeneracy_check` *Determines if State Matrix is Degenerate for Given Data Set.*

Description

`degeneracy_check` finds indices for non-identifiable elements of state matrix and then flips values for those elements and checks changes in resulting fitness.

Usage

```
degeneracy_check(state_mat, action_vec, cols, data, outcome)
```

Arguments

`state_mat`	Numeric matrix with rows as states and columns as predictors.
`action_vec`	Numeric vector indicating what action to take for each state.
`cols`	Optional numeric vector same length as number of columns of the state matrix (`state_mat`) with the action that each column of the state matrix corresponds to the decision model taking in the previous period. This is only relevant when the predictor variables of the FSM are lagged outcomes that include the previous actions taken by that decision model.
`data`	Numeric matrix that has first col period and rest of cols are predictors.
`outcome`	Numeric vector same length as the number of rows as data.

Details

`degeneracy_check` finds indices for non-identifiable elements of state matrix and then flips values for those elements and checks changes in resulting fitness. Being in state/row k (e.g. 2) corresponds to taking action j (e.g. D). For row k, all entries in the matrix that corresponds to taking action j last period (e.g. columns 2 and 4 for D) are identifiable; however, columns that correspond to not taking action j last period (e.g. columns 1 and 3 for D) for the row k that corresponds to taking action j are not identifiable for a deterministic play of the strategy. For all elements of the matrix that are not identifiable, the value of the element can be any integer in the inclusive range of the number of rows of the matrix (e.g. 1 or 2). With empirical data, where the probability that a single deterministic model generated the data is effectively zero, it is useful to find every entry in the matrix that would be unidentifiable if the strategy were played deterministically and then for each element flip it to its opposite value and test for any change in fitness of the strategy on the data. This function implements this idea. If there is no change, a sparse matrix is returned where the the elements in that matrix with a 0 are unidentifiable because their value makes no difference to the fit of the strategy to the provided data. If, for each element in the matrix, switching its value led to a decrease in fitness the following message is displayed, “Your strategy is a deterministic approximation of a stochastic process and all of the elements of the state matrix can be identified.” If the model is fine, then `sparse_state_mat` and `corrected_state_mat` should be equal to `state_mat`.

Value

Returns a list of with sparse and corrected state matrix. If the model is fine, then `sparse_state_mat` and `corrected_state_mat` should be equal to `state_mat`.

`estimation_details` *Extracts slot relevant to estimating the fsm*

Description

Extracts slot relevant to estimating the fsm

Usage

```
estimation_details(x)
```

Arguments

`x`	S4 ga_fsm object

`evolve_model` *Use a Genetic Algorithm to Estimate a Finite-state Machine Model*

Description

`evolve_model` uses a genetic algorithm to estimate a finite-state machine model, primarily for understanding and predicting decision-making.

Usage

```
evolve_model(data, test_data = NULL, drop_nzv = FALSE,
        measure = c("accuracy", "sens", "spec", "ppv"),
        states = NULL, cv = FALSE, max_states = NULL, k = 2,
        actions = NULL, seed = NULL, popSize = 75,
        pcrossover = 0.8, pmutation = 0.1, maxiter = 50,
        run = 25, parallel = FALSE, priors = NULL,
        verbose = TRUE, return_best = TRUE, ntimes = 1)
```

Arguments

`data`	A `data.frame` that has columns named "period" and "outcome" (period is the time period that the outcome action was taken), and one to three additional columns, containing predictors. All of the 3-5 columns should be named. The period and outcome columns should be integer vectors and the columns with the predictor variable data should be logical vectors (`TRUE, FALSE`). If the predictor variable data is not logical, it will coerced to logical with `base::as.logical()`.
`test_data`	Optional `data.frame` that has "period" and "outcome" columns, with one to three additional columns containing predictors. All of the (3-5 columns) should be named. The outcome variable is the decision the decision-maker took for that period. This `data.frame` should be in the same format and have the same order of columns as the data.frame passed to the required `data` argument.
`drop_nzv`	Optional logical vector length one specifying whether predictors variables with variance in provided data near zero should be dropped before model building. Default is `FALSE`. See `caret::nearZeroVar()`, which calls: `caret::nzv()`.
`measure`	Optional length one character vector that is either: "accuracy", "sens", "spec", or "ppv". This specifies what measure of predictive performance to use for training and evaluating the model. The default measure is `"accuracy"`. However, accuracy can be a problematic measure when the classes are imbalanced in the samples, i.e. if a class the model is trying to predict is very rare. Alternatives to accuracy are available that illuminate different aspects of predictive power. Sensitivity answers the question, “ given that a result is truly an event, what is the probability that the model will predict an event?” Specificity answers the question, “given that a result is truly not an event, what is the probability that the model will predict a negative?” Positive predictive value answers, “what is the percent of predicted positives that are actually positive?”

`states`	Optional numeric vector with the number of states. If not provided, will be set to `max(data$outcome)`.
`cv`	Optional logical vector length one for whether cross-validation should be conducted on training data to select optimal number of states. This can drastically increase computation time because if `TRUE`, it will run `evolve_model` `k*max_states` times to estimate optimal value for states. Ties are broken by choosing the smaller number of states. Default is `FALSE`.
`max_states`	Optional numeric vector length one only relevant if `cv==TRUE`. It specifies how up to how many states that cross-validation should search through. If not provided, will be set to `states + 1`.
`k`	Optional numeric vector length one only relevant if cv==TRUE, specifying number of folds for cross-validation.
`actions`	Optional numeric vector with the number of actions. If not provided, then actions will be set as the number of unique values in the outcome vector.
`seed`	Optional numeric vector length one.
`popSize`	Optional numeric vector length one specifying the size of the GA population. A larger number will increase the probability of finding a very good solution but will also increase the computation time. This is passed to the GA::ga() function of the **GA** package.
`pcrossover`	Optional numeric vector length one specifying probability of crossover for GA. This is passed to the GA::ga() function of the **GA** package.
`pmutation`	Optional numeric vector length one specifying probability of mutation for GA. This is passed to the GA::ga() function of the **GA** package.
`maxiter`	Optional numeric vector length one specifying max number of iterations for stopping the GA evolution. A larger number will increase the probability of finding a very good solution but will also increase the computation time. This is passed to the GA::ga() function of the **GA** package. `maxiter` is scaled by how many parameters are in the model: `maxiter <- maxiter + ((maxiter*(nBits^2)) / maxiter)`.
`run`	Optional numeric vector length one specifying max number of consecutive iterations without improvement in best fitness score for stopping the GA evolution. A larger number will increase the probability of finding a very good solution but will also increase the computation time. This is passed to the GA::ga() function of the **GA** package.
`parallel`	Optional logical vector length one. For running the GA evolution in parallel. Depending on the number of cores registered and the memory on your machine, this can make the process much faster, but only works for Unix-based machines that can fork the processes.
`priors`	Optional numeric matrix of solutions strings to be included in the initialization. User needs to use a decoder function to translate prior decision models into bits and then provide them. If this is not specified, then random priors are automatically created.
`verbose`	Optional logical vector length one specifying whether helpful messages should be displayed on the user's console or not.

`return_best`	Optional logical vector length one specifying whether to return just the best model or all models. Only relevant if ntimes > 1. Default is TRUE.
`ntimes`	Optional integer vector length one specifying the number of times to estimate model. Default is 1 time.

Details

This is the main function of the **datafsm** package. It relies on the **GA** package for genetic algorithm optimization. `evolve_model` takes data on predictors and data on the outcome. It automatically creates a fitness function that takes the data, an action vector `evolve_model` generates, and a state matrix `evolve_model` generates as input and returns numeric vector of the same length as the `outcome`. `evolve_model` then computes a fitness score for that potential solution FSM by comparing it to the provided `outcome`. This is repeated for every FSM in the population and then the probability of selection for the next generation is proportional to the fitness scores. The default is also for the function to call itself recursively while varying the number of states inside a cross-validation loop in order to estimate the optimal number of states.

If parallel is set to TRUE, then these evaluations are distributed across the available processors of the computer using the **doParallel** package, otherwise, the evaluations of fitness are conducted sequentially. Because this fitness function that `evolve_model` creates must loop through all the data every time it is evaluated and we need to evaluate many possible solution FSMs, the fitness function is implemented in C++ so it is very fast.

`evolve_model` uses a stochastic meta-heuristic optimization routine to estimate the parameters that define a FSM model. Generalized simulated annealing, or tabu search could work, but they are more difficult to parallelize. The current version uses the **GA** package's genetic algorithm because GAs perform well in rugged search spaces to solve integer optimization problems, are a natural complement to our binary string representation of FSMs, and are easily parallelized.

This function evolves the models on training data and then, if a test set is provided, uses the best solution to make predictions on test data. Finally, the function returns the GA object and the decoded version of the best string in the population. See ga_fsm for the details of the slots (objects) that this type of object will have.

Value

Returns an S4 object of class ga_fsm. See ga_fsm for the details of the slots (objects) that this type of object will have and for information on the methods that can be used to summarize the calling and execution of `evolve_model()`, including `summary`, `print`, and `plot`. Timing measurement is in seconds.

References

Luca Scrucca (2013). GA: A Package for Genetic Algorithms in R. Journal of Statistical Software, 53(4), 1-37. URL `https://www.jstatsoft.org/v53/i04/`.

Examples

```
## Not run:
# Create data:
cdata <- data.frame(period = rep(1:10, 1000),
```

```
                   outcome = rep(1:2, 5000),
                   my.decision1 = sample(1:0, 10000, TRUE),
                   other.decision1 = sample(1:0, 10000, TRUE))
(res <- evolve_model(cdata, cv=FALSE))
summary(res)
plot(res, action_label = c("C", "D"))
library(GA)
plot(estimation_details(res))

## End(Not run)

# In scripts, it can makes sense to set parallel to
# 'as.logical(Sys.info()['sysname'] != 'Windows')'.
```

`evolve_model_cv` *Estimate Optimal Number of States of a Finite-state Machine Model*

Description

`evolve_model_cv` calls `evolve_model` with varied numbers of states and compares their performance with cross-validation.

Usage

```
evolve_model_cv(data, measure, k, actions, max_states, seed,
         popSize, pcrossover, pmutation, maxiter, run,  parallel,
         verbose, ntimes)
```

Arguments

`data`	A `data.frame` that has columns named "period" and "outcome" (period is the time period that the outcome action was taken), and one to three additional columns, containing predictors. All of the 3-5 columns should be named. The period and outcome columns should be integer vectors and the columns with the predictor variable data should be logical vectors (`TRUE, FALSE`). If the predictor variable data is not logical, it will coerced to logical with `base::as.logical()`.
`measure`	Optional length one character vector that is either: "accuracy", "sens", "spec", or "ppv". This specifies what measure of predictive performance to use for training and evaluating the model. The default measure is `"accuracy"`. However, accuracy can be a problematic measure when the classes are imbalanced in the samples, i.e. if a class the model is trying to predict is very rare. Alternatives to accuracy are available that illuminate different aspects of predictive power. Sensitivity answers the question, " given that a result is truly an event, what is the probability that the model will predict an event?" Specificity answers the question, "given that a result is truly not an event, what is the probability that the model will predict a negative?" Positive predictive value answers, "what is the percent of predicted positives that are actually positive?"

`k`	Optional numeric vector length one only relevant if cv==TRUE, specifying number of folds for cross-validation.
`actions`	Optional numeric vector with the number of actions. If not provided, then actions will be set as the number of unique values in the outcome vector.
`max_states`	Optional numeric vector length one only relevant if `cv==TRUE`. It specifies how up to how many states that cross-validation should search through. If not provided, will be set to `states + 1`.
`seed`	Optional numeric vector length one.
`popSize`	Optional numeric vector length one specifying the size of the GA population. A larger number will increase the probability of finding a very good solution but will also increase the computation time. This is passed to the GA::ga() function of the **GA** package.
`pcrossover`	Optional numeric vector length one specifying probability of crossover for GA. This is passed to the GA::ga() function of the **GA** package.
`pmutation`	Optional numeric vector length one specifying probability of mutation for GA. This is passed to the GA::ga() function of the **GA** package.
`maxiter`	Optional numeric vector length one specifying max number of iterations for stopping the GA evolution. A larger number will increase the probability of finding a very good solution but will also increase the computation time. This is passed to the GA::ga() function of the **GA** package. `maxiter` is scaled by how many parameters are in the model: `maxiter <- maxiter + ((maxiter*(nBits^2)) / maxiter)`.
`run`	Optional numeric vector length one specifying max number of consecutive iterations without improvement in best fitness score for stopping the GA evolution. A larger number will increase the probability of finding a very good solution but will also increase the computation time. This is passed to the GA::ga() function of the **GA** package.
`parallel`	Optional logical vector length one. For running the GA evolution in parallel. Depending on the number of cores registered and the memory on your machine, this can make the process much faster, but only works for Unix-based machines that can fork the processes.
`verbose`	Optional logical vector length one specifying whether helpful messages should be displayed on the user's console or not.
`ntimes`	Optional integer vector length one specifying the number of times to estimate model. Default is 1 time.

Value

Returns the number of states that maximizes the `measure`, e.g. accuracy.

References

Luca Scrucca (2013). GA: A Package for Genetic Algorithms in R. Journal of Statistical Software, 53(4), 1-37. URL `https://www.jstatsoft.org/v53/i04/`.

Hastie, T., R. Tibshirani, and J. Friedman. (2009). The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Second Edition. 2nd ed. New York, NY: Springer.

---

evolve_model_ntimes *Use a Genetic Algorithm to Estimate a Finite-state Machine Model n-times*

Description

`evolve_model` uses a genetic algorithm to estimate a finite-state machine model, primarily for understanding and predicting decision-making.

Usage

```
evolve_model_ntimes(data, test_data = NULL, drop_nzv = FALSE,
        measure = c("accuracy", "sens", "spec", "ppv"),
        states = NULL, cv = FALSE, max_states = NULL, k = 2,
        actions = NULL, seed = NULL, popSize = 75,
        pcrossover = 0.8, pmutation = 0.1, maxiter = 50,
        run = 25, parallel = FALSE, priors = NULL,
        verbose = TRUE, return_best = TRUE, ntimes = 10,
        cores = NULL)
```

Arguments

`data`	A `data.frame` that has columns named "period" and "outcome" (period is the time period that the outcome action was taken), and one to three additional columns, containing predictors. All of the 3-5 columns should be named. The period and outcome columns should be integer vectors and the columns with the predictor variable data should be logical vectors (`TRUE, FALSE`). If the predictor variable data is not logical, it will coerced to logical with `base::as.logical()`.
`test_data`	Optional `data.frame` that has "period" and "outcome" columns, with one to three additional columns containing predictors. All of the (3-5 columns) should be named. The outcome variable is the decision the decision-maker took for that period. This `data.frame` should be in the same format and have the same order of columns as the data.frame passed to the required `data` argument.
`drop_nzv`	Optional logical vector length one specifying whether predictors variables with variance in provided data near zero should be dropped before model building. Default is `FALSE`. See `caret::nearZeroVar()`, which calls: `caret::nzv()`.
`measure`	Optional length one character vector that is either: "accuracy", "sens", "spec", or "ppv". This specifies what measure of predictive performance to use for training and evaluating the model. The default measure is `"accuracy"`. However, accuracy can be a problematic measure when the classes are imbalanced in the samples, i.e. if a class the model is trying to predict is very rare. Alternatives to accuracy are available that illuminate different aspects of predictive power. Sensitivity answers the question, “ given that a result is truly an event, what is the probability that the model will predict an event?” Specificity answers the

	question, “given that a result is truly not an event, what is the probability that the model will predict a negative?” Positive predictive value answers, “what is the percent of predicted positives that are actually positive?”
`states`	Optional numeric vector with the number of states. If not provided, will be set to `max(data$outcome)`.
`cv`	Optional logical vector length one for whether cross-validation should be conducted on training data to select optimal number of states. This can drastically increase computation time because if `TRUE`, it will run `evolve_model` `k*max_states` times to estimate optimal value for states. Ties are broken by choosing the smaller number of states. Default is `FALSE`.
`max_states`	Optional numeric vector length one only relevant if `cv==TRUE`. It specifies how up to how many states that cross-validation should search through. If not provided, will be set to `states + 1`.
`k`	Optional numeric vector length one only relevant if cv==TRUE, specifying number of folds for cross-validation.
`actions`	Optional numeric vector with the number of actions. If not provided, then actions will be set as the number of unique values in the outcome vector.
`seed`	Optional numeric vector length one.
`popSize`	Optional numeric vector length one specifying the size of the GA population. A larger number will increase the probability of finding a very good solution but will also increase the computation time. This is passed to the GA::ga() function of the **GA** package.
`pcrossover`	Optional numeric vector length one specifying probability of crossover for GA. This is passed to the GA::ga() function of the **GA** package.
`pmutation`	Optional numeric vector length one specifying probability of mutation for GA. This is passed to the GA::ga() function of the **GA** package.
`maxiter`	Optional numeric vector length one specifying max number of iterations for stopping the GA evolution. A larger number will increase the probability of finding a very good solution but will also increase the computation time. This is passed to the GA::ga() function of the **GA** package. `maxiter` is scaled by how many parameters are in the model: `maxiter <- maxiter + ((maxiter*(nBits^2)) / maxiter)`.
`run`	Optional numeric vector length one specifying max number of consecutive iterations without improvement in best fitness score for stopping the GA evolution. A larger number will increase the probability of finding a very good solution but will also increase the computation time. This is passed to the GA::ga() function of the **GA** package.
`parallel`	Optional logical vector length one. For running the GA evolution in parallel. Depending on the number of cores registered and the memory on your machine, this can make the process much faster, but only works for Unix-based machines that can fork the processes.
`priors`	Optional numeric matrix of solutions strings to be included in the initialization. User needs to use a decoder function to translate prior decision models into bits and then provide them. If this is not specified, then random priors are automatically created.

`verbose`	Optional logical vector length one specifying whether helpful messages should be displayed on the user's console or not.
`return_best`	Optional logical vector length one specifying whether to return just the best model or all models. Only relevant if ntimes > 1. Default is TRUE.
`ntimes`	Optional integer vector length one specifying the number of times to estimate model. Default is 1 time.
`cores`	integer vector length one specifying number of cores to use if parallel is TRUE.

Details

This function of the **datafsm** package applies the `evolve_model` function multiple times and then returns a list with either all the models or the best one.

`evolve_model` uses a stochastic meta-heuristic optimization routine to estimate the parameters that define a FSM model. Because this is not guaranteed to return the best result, we run it many times.

Value

Returns a list where each element is an S4 object of class ga_fsm. See ga_fsm for the details of the slots (objects) that this type of object will have and for information on the methods that can be used to summarize the calling and execution of `evolve_model()`, including `summary`, `print`, and `plot`.

Examples

```
## Not run:
# Create data:
cdata <- data.frame(period = rep(1:10, 1000),
                    outcome = rep(1:2, 5000),
                    my.decision1 = sample(1:0, 10000, TRUE),
                    other.decision1 = sample(1:0, 10000, TRUE))
(res <- evolve_model_ntimes(cdata, ntimes=2))
(res <- evolve_model_ntimes(cdata, return_best = FALSE, ntimes=2))

## End(Not run)
```

`find_wildcards` *Find Indices for Non-identifiable Elements of State Matrix.*

Description

`find_wildcards` finds indices for non-identifiable elements of state matrix.

Usage

```
find_wildcards(state_mat, action_vec, cols)
```

Arguments

`state_mat`	Numeric matrix with rows as states and columns as predictors.
`action_vec`	Numeric vector indicating what action to take for each state.
`cols`	Numeric vector same length as number of columns of the state matrix (`state_mat`) with the action that each column of the state matrix corresponds to the decision model taking in the previous period. This is only relevant when the predictor variables of the FSM are lagged outcomes that include the previous actions taken by that decision model.

Details

This is a helper function for `degeneracy_check`.

Value

Returns a list of indices (tuples specifying row and column of a matrix).

Examples

```
tft_state <- matrix(c(1, 1, 1, 1, 2, 2, 2, 2), 2, 4)
tft_action <- matrix(c(1, 2))
find_wildcards(tft_state, tft_action,  c(1, 2, 1, 2))
```

`fitnessCPP` *Fitness Function in C++*

Description

A generated action vector and state matrix are input and this function returns a numeric vector of the same length as the `outcome`. `evolve_model` then computes a fitness score for that potential solution FSM by comparing it to the provided `outcome`. This is repeated for every FSM in the population and then the probability of selection for the next generation is set to be proportional to the fitness scores. This function is also used in the predict method for the resulting final model that is returned. The function aborts if the user aborts in R, checking every 1000 iterations.

Usage

```
fitnessCPP(action_vec, state_mat, covariates, period)
```

Arguments

`action_vec`	Integer Vector.
`state_mat`	Integer Matrix.
`covariates`	Integer Matrix.
`period`	Integer Vector.

`ga_fsm-class` *An S4 class to return the results of using a GA to estimate a FSM with* `evolve_model`.

Description

An S4 class to return the results of using a GA to estimate a FSM with `evolve_model`.

Turns ga_fsm S4 object into list of summaries for printing and then prints it.

Plots ga_fsm S4 object's state transition matrix

Plots ga_fsm S4 object's variable importances

Plots ga_fsm S4 object's variable importances

Extracts slot relevant to estimating the fsm

Extracts performance

Extracts slot of variable importances

Extracts slot of action_vec

Extracts number of states

Predicts new data with estimated model

Usage

```
## S4 method for signature 'ga_fsm'
print(x, ...)

## S4 method for signature 'ga_fsm'
show(object)

## S4 method for signature 'ga_fsm'
summary(object, digits = 3)

## S4 method for signature 'ga_fsm,ANY'
plot(x, y, maintitle = "Transition Diagram",
          action_label = NULL, transition_label = NULL,
          curvature = c(0.3, 0.6, 0.8))

## S4 method for signature 'ga_fsm'
barplot(height, ...)

## S4 method for signature 'ga_fsm'
dotchart(x, labels)

## S4 method for signature 'ga_fsm'
estimation_details(x)

## S4 method for signature 'ga_fsm'
```

```
best_performance(x)

## S4 method for signature 'ga_fsm'
varImp(x)

## S4 method for signature 'ga_fsm'
action_vec(x)

## S4 method for signature 'ga_fsm'
states(x)

## S4 method for signature 'ga_fsm'
predict(object, data, type = ”prob”, na.action = stats::na.omit, ...)
```

Arguments

`x`	S4 ga_fsm object. @export
`...`	arguments to be passed to/from other methods.
`object`	S4 ga_fsm object
`digits`	Optional numeric vector length one for how many significant digits to print, default is 3. @export
`y`	not used.
`maintitle`	optional character vector
`action_label`	optional character vector same length as action vector, where each ith element corresponds to what that ith element in the action vector represents. This will be used to fill in the states (circles) of the state transition matrix to be plotted.
`transition_label`	optional character vector same length as number of columns of state transition matrix.
`curvature`	optional numeric vector specifying the curvature of the lines for a diagram of 2 or more states.
`height`	ga_fsm S4 object
`labels`	vector of labels for each point. For vectors the default is to use names(x) and for matrices the row labels dimnames(x)[[1]].
`data`	A `data.frame` that has columns named "period" and "outcome" (period is the time period that the outcome action was taken), and one to three additional columns, containing predictors. All of the 3-5 columns should be named. The period and outcome columns should be integer vectors and the columns with the predictor variable data should be logical vectors (`TRUE, FALSE`). If the predictor variable data is not logical, it will coerced to logical with `base::as.logical()`.
`type`	Not currently used.
`na.action`	Optional function.

Methods (by generic)

- `print`: An S4 method for printing a ga_fsm S4 object
- `show`: An S4 method for showing a ga_fsm S4 object
- `summary`: An S4 method for summarizing a ga_fsm S4 object
- `plot`:
- `barplot`:
- `dotchart`: Plots ga_fsm S4 object's variable importances
- `estimation_details`: @export
- `best_performance`: @export
- `varImp`: @export
- `action_vec`: @export
- `states`: @export
- `predict`: Predicts new data with estimated model

Slots

`call` Language from the call of the function `evolve_model`.

`actions` Numeric vector with the number of actions.

`states` Numeric vector with the number of states.

`GA` S4 object created by ga() from the GA package.

`state_mat` Numeric matrix with rows as states and columns as predictors.

`action_vec` Numeric vector indicating what action to take for each state.

`predictive` Numeric vector of length one with test data accuracy if test data was supplied; otherwise, a character vector with a message that the user should provide test data for better estimate of performance.

`varImp` Numeric vector same length as number of columns of state matrix with relative importance scores for each predictor.

`varImp2` Numeric matrix same size as state matrix with relative importance scores for each transition.

`timing` Numeric vector length one with the total elapsed seconds it took `evolve_model` to execute.

`diagnostics` Character vector length one, to be printed with base::cat().

`NV_games` *Empirical prisoner's dilemma games from Nay and Vorobeychik*

Description

A dataset containing 168,386 total rounds of play in 30 different variations on the iterated prisoner's dilemma games. The data comes from J.J. Nay and Y. Vorobeychik, "Predicting Human Cooperation," PLOS ONE 11(5), e0155656 (2016).

Usage

```
NV_games
```

Format

A data frame with 168,386 rows and 51 variables:

period Which turn of the given game

my.decision The player's move in this turn

risk Boolean variable: 1 indicates stochastic payoffs, 0 deterministic payoffs

delta Probability the game ends after each round

r1 Normalized difference in payoff between both players cooperating and both defecting

r2 Normalized difference in payoff between both players cooperating and the payoff for being a sucker (cooperating when the opponent defects)

error Probability that the player's intended move is switched to the opposite move

data Which dataset did this game come from: AM = Andreoni & Miller; BR = Bereby-Meyer & Roth; DB = Dal Bo; DF = Dal Bo & Frechette; DO = Duffy & Ochs; FO = Friedman & Oprea; FR = Fudenberg, Rand, & Dreber; and KS = Kunreuther, Silvasi, Bradlow & Small

my.decision1 The player's move in the previous turn

my.decision2 The player's move two turns ago

my.decision3 The player's move three turns ago

my.decision4 The player's move four turns ago

my.decision5 The player's move five turns ago

my.decision6 The player's move six turns ago

my.decision7 The player's move seven turns ago

my.decision8 The player's move eight turns ago

my.decision9 The player's move nine turns ago

other.decision1 The opponent's move in the previous turn

other.decision2 The opponent's move two turns ago

other.decision3 The opponent's move three turns ago

other.decision4 The opponent's move four turns ago

other.decision5 The opponent's move five turns ago

other.decision6 The opponent's move six turns ago

other.decision7 The opponent's move seven turns ago

other.decision8 The opponent's move eight turns ago

other.decision9 The opponent's move nine turns ago

my.payoff1 The player's payoff in the previous turn

my.payoff2 The player's payoff two turns ago

my.payoff3 The player's payoff three turns ago

my.payoff4 The player's payoff four turns ago

my.payoff5 The player's payoff five turns ago

my.payoff6 The player's payoff six turns ago

my.payoff7 The player's payoff seven turns ago

my.payoff8 The player's payoff eight turns ago

my.payoff9 The player's payoff nine turns ago

other.payoff1 The opponent's payoff in the previous turn

other.payoff2 The opponent's payoff two turns ago

other.payoff3 The opponent's payoff three turns ago

other.payoff4 The opponent's payoff four turns ago

other.payoff5 The opponent's payoff five turns ago

other.payoff6 The opponent's payoff six turns ago

other.payoff7 The opponent's payoff seven turns ago

other.payoff8 The opponent's payoff eight turns ago

other.payoff9 The opponent's payoff nine turns ago

r Reward: payoff when both players cooperate

t Temptation: payoff when player defects and opponent cooperates

s Sucker: Payoff when player cooperates and opponent defects

p Punishment: payoff when both players defect

infin Boolean: 1 indicates infinite game with probability delta of ending at each round; 0 indicates pre-determined number of rounds

contin Boolean: 1 indicates the game is played in continuous time; 0 indicates discrete rounds

group Which group (version of the game) is being played?

Source

doi: 10.1371/journal.pone.0155656

`performance` *Measure Model Performance*

Description

`performance` measures difference between predictions and data

Usage

```
performance(results, outcome, measure)
```

Arguments

`results`	Numeric vector with predictions
`outcome`	Numeric vector same length as results with real data to compare to.
`measure`	Optional length one character vector that is either: "accuracy", "sens", "spec", or "ppv". This specifies what measure of predictive performance to use for training and evaluating the model. The default measure is `"accuracy"`. However, accuracy can be a problematic measure when the classes are imbalanced in the samples, i.e. if a class the model is trying to predict is very rare. Alternatives to accuracy are available that illuminate different aspects of predictive power. Sensitivity answers the question, “ given that a result is truly an event, what is the probability that the model will predict an event?” Specificity answers the question, “given that a result is truly not an event, what is the probability that the model will predict a negative?” Positive predictive value answers, “what is the percent of predicted positives that are actually positive?”

Details

This is the function of the **datafsm** package used to measure the fsm model performance. It uses the caret package.

Value

Returns a numeric vector length one.

`states` *Extracts number of states*

Description

Extracts number of states

Usage

```
states(x)
```

Arguments

`x`	S4 ga_fsm object

`varImp` *Extracts slot of variable importances*

Description

Extracts slot of variable importances

Usage

```
varImp(x)
```

Arguments

`x`	S4 ga_fsm object

`var_imp` *Variable Importance Measure for A FSM Model*

Description

`var_imp` calculates the importance of the covariates of the model.

Usage

```
var_imp(state_mat, action_vec, data, outcome, period, measure)
```

Arguments

`state_mat`	Numeric matrix with rows as states and columns as predictors.
`action_vec`	Numeric vector indicating what action to take for each state.
`data`	Data frame that has "period" and "outcome" columns and rest of cols are predictors, ranging from one to three predictors. All of the (3-5 columns) should be named.
`outcome`	Numeric vector same length as the number of rows as data.
`period`	Numeric vector same length as the number of rows as data.

`measure`	Optional length one character vector that is either: "accuracy", "sens", "spec", or "ppv". This specifies what measure of predictive performance to use for training and evaluating the model. The default measure is `"accuracy"`. However, accuracy can be a problematic measure when the classes are imbalanced in the samples, i.e. if a class the model is trying to predict is very rare. Alternatives to accuracy are available that illuminate different aspects of predictive power. Sensitivity answers the question, “ given that a result is truly an event, what is the probability that the model will predict an event?” Specificity answers the question, “given that a result is truly not an event, what is the probability that the model will predict a negative?” Positive predictive value answers, “what is the percent of predicted positives that are actually positive?”

Details

Takes the state matrix and action vector from an already evolved model and the fitness function and data used to evolve the model (or this could be test data), flips the values of each of the elements in the state matrix and measures the change in fitness (prediction of data) relative to the original model. Then these changes are summed across columns to provide the importance of each of the columns of the state matrix.

Value

Numeric vector the same length as the number of columns of the provided state matrix (the number of predictors in the model) with relative importance scores for each predictor.

Index