Package 'EasyMARK'

February 19, 2015

Type Package

Title Utility functions for working with mark-recapture data.

Version 1.0

Depends R (>= 3.0.2)

Date 2014-01-28

Author John Waller

Maintainer John Waller <john.waller@biol.lu.se>

Description Contains a few utility functions for working with capture-history data, a function for simulating capture-history data, and a function to fit this data, using a Gibbs sampler.

License GPL-2

Imports MASS, stringr, rjags, coda, foreach, doParallel,random, parallel

NeedsCompilation no

Repository CRAN

Date/Publication 2014-04-28 11:08:30

R topics documented:

`EasyMARK-package` *Make it easier to create, work with, and analyse capture histories, with bias towards natural selection studies.*

Description

Contains a few utility functions for working with capture-history data, a function for simulating capture-history data, and a function to fit this data, using a Gibbs sampler.

Details

Version:	1.0
Imports: MASS, stringr, rjags, coda, foreach, doParallel, random Package:	EasyMARK
Type:	Package
Date:	2014-01-29
License:	GPL-2

Author(s)

Maintainer: <john.waller@biol.lu.se>

`Collapse.CH` *Collapse a capture history matrix*

Description

Collapses a capture history matrix into a single-columned data.frame.

Usage

```
Collapse.CH(ch)
```

Arguments

`ch`	Input a `matrix` of 1's and 0's to be collapsed. Most likely this matrix will have one individual per row and one capture occassion per column.

Details

Some mark-recapture programs want data in collapsed form, as a character string. This function will collapse a matrix into a single columned character sring of ones and zeros.

Value

`data.frame of capture histories`
: Returns a single-columned data frame of the capture histories as a character string.

Author(s)

John Waller

See Also

`Split.CH`

Examples

```
#' set up variables to be used by Simulate.CH for ten individuals
N = 10
x1 = rnorm(N) #' each trait is normally distributed

#' Run Simulate.CH with a constant recapture probability
chObj = Simulate.CH(surv.form = 1 + 0.15*x1, p.constant = 1, N = N)

attributes(chObj) #' see what is inside our object
ch = chObj$ch_split #' grab our matrix from the list
ch #' lets look at it

Collapse.CH(ch) #' now let's turn it into a data.frame
```

`Lifespan` *minimum lifespan*

Description

Gives a `vector` of minimum lifespans for each individual.

Usage

```
Lifespan(ch)
```

Arguments

`ch`
: input a matrix or a single-columned data.frame of capture histories. It will return a vector of minimum lifespans for each row

Details

This function computes the mimimum lifespans given a capture history. This is computed as the first day seen subtracted by the last day seen. For example, an individual with the capture history of "1001" would have a minimum lifespan of 3; "1110" = 2; "1010" = 2, "01000001" = 6 etc. See examples.

Value

`vector` A numerical vector of lifespans

Author(s)

John Waller

Examples

```
#' set up variables to be used by Simulate.CH for ten individuals
N = 10
x1 = rnorm(N) #' each trait is normally distributed

#' Run Simulate.CH with a constant recapture probability
chObj = Simulate.CH(surv.form = 1 + 0.15*x1, p.constant = 1, N = N)

attributes(chObj) #' see what is inside our object
ch_matrix = chObj$ch_split #' grab our matrix from the list
ch_matrix #' lets look at it

ch_df = chObj$ch #' lets grab a data.frame also
ch_df

#' we can compute the minimum lifespan on either a matrix or a data.frame as input
Lifespan(ch_matrix) #' on a matrix
Lifespan(ch_df) #' on a data.frame
```

`Maker.CH` *Make a capture history from dates.*

Description

Creates a capture history from a vector of dates and individual ids.

Usage

```
Maker.CH(dates = dates, id = id, date.format)
```

Arguments

dates	a character vector or factor of dates in any format (see below).
id	a character vector or factor of unique ids the same length as the dates vector.
date.format	a character Rstring specifying how the date should be formatted.

Details

The vector of dates should, including date first marked. The date vector should line up with id vector, with ids repeated for each date the idvidual was re-captured or seen. Right now it assumes the capture occassions are per day. This might change in the future, allowing per month or per year time periods. Ids do not have to be sorted, but must be repeated for each date the individual is seen.

Value

ch	Returns a matrix of capture histories. Row names are the individual id names and column names are the dates that individual was seen

Author(s)

John Waller

Examples

```
#' our date vector, which should line up with the id vector
#' we would normally pull this from a data.frame from an external source
dates = c("2012_07_05","2012_07_01","2012_07_01", "2012_07_02", "2012_07_07",
"2012_07_01", "2012_07_03", "2012_07_07")

#' ids lined up with our dates. Each id is repeated for each day seen, including the day
#' it was marked.
id = c("B","B","A","A","A","C","C","C")

#' See that our vectors are the same length
length(id)
length(dates)

#' date.format can be in a variety of forms:
#' "%Y_%m_%d" = 2012_07_01
#' "%Y:%m:%d" = 2012:07:01
#' "%Y-%m-%d" = 2012-07-01
#' "%m-%Y-%d" = 07-2012-01
#' "%d-%m-%Y" = 01-07-2012
#' and so on...

#' run our function
ch = Maker.CH(dates = dates, id = id, date.format = "%Y_%m_%d")


#' out matrix with colnames as the dates seen and rownames as the individual ids
ch
```

MARK.MCMC *Mark Gibbs sampler*

Description

A Bayesian way of fitting a mark-recapture model to capture history data.

Usage

```
MARK.MCMC(ch, cov, n.iter = 100, burn.in = 50, number.of.models = 10, n.chains = 2,
add = TRUE, quad = TRUE, corr = TRUE)
```

Arguments

ch	A matrix or a collapsed, single-columned data.frame of capture histories, one row for each individual.
cov	A data.frame of covariates (traits) that should be considered
n.iter	Number iterations the sampler should take.
burn.in	The number of iterations to discard from each chain.
number.of.models	The number of models to calculate the posterior probability for.
n.chains	The number of chains. Each chain will be run on a separate core if possible.
add	Should all possilbe addative terms be considered.
quad	Should all possible quadratic terms be considered.
corr	Should all possible pairwise interaction terms be considered.

Details

This function implements a Gibbs sampler to estimate mark-recapture parameters. It is essentially a wrapper for a Jags or WinBugs model. Things it does not do right now: A. does not handle data with significant time or age dependent effects, B. cannot deal with re-capture heterogeniety (i.e. re-capture dependence on a trait), C. cannot fit a specific predefined model, D. cannot use predefined priors (usues diffuse priors instead, see reference). Other R libraries exist with this functionality, namely marked, RMark, mra. What it can do is do automatic model selection on all combinations of models supplied. See examples for usage.

Value

(mcmc = mcmc, mcmc.list = mcmc.list, pp = pp.results, estimates = estimates, p = p, gelman = gelman) Returns a list:

$mcmc	A single matrix with all the parameter estimates for each chain combined.
$mcmc.list	An object of class mcmc.list, one element for each chain.

$pp	A data.frame of posterior probabilities for each model.
$estimates	A data.frame of parameter estimates for survival probability.
$p	The estimated recapture probability.
$gelman	The the output from gelman.diag in the library coda, a convergence diagnostic.

Author(s)

John Waller

References

Gimenez et al. 2009 "Estimagint and visulizing fitness surfaces using mark-recapture data" Evolution

Examples

```
#' Example 1 perfect detection
## Not run:

#' generate some data to input into our simulator
N = 100
#' Two traits
x1 = rnorm(N,0,1)
x2 = rnorm(N,0,1)

#' Use our simulator function
#' with constant and perfect recapture probability, p.constant = 1
#' with positive linear selection on trait x1 and no selection on trait x2
chObj = Simulate.CH(surv.form = 1 + 0.15*x1 + 0*x2, p.constant = 1, N = N)
str(chObj) #' what is contained in our chObj

ch = chObj$ch #' Let's pull out our simulated capture histories
ch #' what it looks like

#' make a data.frame of covariate values
cov = data.frame(x1 = x1, x2 = x2)

#' cov should have the same number of rows as ch
nrow(ch)
nrow(cov)

#' Now let's estimate the parameters of our simulated data.
#' And test which model best fits the data.
#' We use a small number iteration here,
#' n.iter = 1000, so it runs quickly.
#' One should definitely use many more iterations in practice.
#' We we throw away half of our n.iter in the the burn in, burn.in = 500
MCMC = MARK.MCMC(ch = ch, cov = cov, n.iter = 1000, burn.in = 500, number.of.models = 5,
n.chains = 2, add = TRUE, quad = TRUE, corr = TRUE)
```

```
#' Let's look at what is inside our MCMC object
attributes(MCMC)

#' Let's look at the posterior probability, pp
#' Since we did not run very many iterations, the correct model (x1),
#' may not have the highest probability
MCMC$pp

#' Let's look at the recapture probability
#' Since we set it at 1, it should be close to 1
MCMC$p

#' Let's look at our estimates of our parameters.
#' Since we set the gradient on trait x1 to 0.15, x1's parameters should be close to 0.15
#' However, our estimates may not be very good, since we used so few iterations
MCMC$estimates

#' Let's look at our convergence diagnostic
#' These values should be close to 1 for all beta variables and p
#' w and sigmaeps can mostly be ignored
#' See gelman.diag in the coda library for more details.
MCMC$gelman


#' Example 2 imperfect detection
#' Same procedure as in Example 1
N = 100
x1 = rnorm(N,0,1)
x2 = rnorm(N,0,1)

#' Only this time we will lower our recapture probability, p.constant, from 1 to 0.5
chObj = Simulate.CH(surv.form = 1 + 0.15*x1 + 0*x2, p.constant = 0.5, N = N)
ch = chObj$ch

cov = data.frame(x1 = x1, x2 = x2)
MCMC = MARK.MCMC(ch = ch, cov = cov, n.iter = 1000, burn.in = 500, number.of.models = 5,
n.chains = 2, add = TRUE, quad = TRUE, corr = TRUE)

#' look at our output
MCMC$pp
#' p should be close to 0.5
MCMC$p
MCMC$estimates
MCMC$gelman


#' Example 3 Test Only Addative Models
#' Same as before...
N = 100
x1 = rnorm(N,0,1)
x2 = rnorm(N,0,1)

#' Only this time we will lower our recapture probability, p.constant, from 1 to 0.5
```

```
chObj = Simulate.CH(surv.form = 1 + 0.15*x1 + 0*x2, p.constant = 0.5, N = N)
ch = chObj$ch

cov = data.frame(x1 = x1, x2 = x2)
#' Now we set quad = FALSE, corr = FALSE
MCMC = MARK.MCMC(ch = ch, cov = cov, n.iter = 1000, burn.in = 500, number.of.models = 5,
n.chains = 2, add = TRUE, quad = FALSE, corr = FALSE)

#' Let's look at the posterior probability
#' It should only show the four possible addative models and blank slots for the rest
#' x1 should have the highest pp, since our data was simulated under those conditions
MCMC$pp


#' Example 3 Stabilizing selection
#' We will bump up the sample size to 500,
#' since stabilizing selection is  a little bit harder
#' to detect with small sample sizes
N = 500
x1 = rnorm(N,0,1)

#' For stabilizing selection, we will add a term to our simulator: -0.15*x1^2
#' We will keep our recapture probability at an high value
chObj = Simulate.CH(surv.form = 1 + 0*x1 + -0.3*x1^2, p.constant = 0.7, N = N)
ch = chObj$ch

cov = data.frame(x1 = x1)

#' We will set corr = FALSE, since we only have one trait, x1
#' May take a few minutes ~5 minutes to run...
MCMC = MARK.MCMC(ch = ch, cov = cov, n.iter = 1000, burn.in = 500, number.of.models = 5,
n.chains = 2, add = TRUE, quad = TRUE, corr = FALSE)

#' Let's look at the posterior probability
#' x1^2 should be the model with the higher posterior probability
MCMC$pp

#' x1^2 term should have an estimate close to -0.3
MCMC$estimates

## End(Not run)
```

`Simulate.CH` *Simulate capture histories for individuals*

Description

This function will simulate capture histories

Usage

```
Simulate.CH(surv.form, p.form, p.constant = NULL, surv.constant = NULL,
N = 100, max.occ = 100, noise = 0.2)
```

Arguments

surv.form	an expression dictating how survival probability should depend on a given trait
p.form	an expression dictating how re-capture probability should depend on a given trait
p.constant	a constant value for recapture probability. Should be between 0 and 1
surv.constant	a constant value for survival probability. Should be between 0 and 1
N	the number of individual recapture histories to make. Should be the same length as any "trait" variables
max.occ	the maximum number of re-capture occassions
noise	the level of random error to add. Probably should be a value between 0-1, but probably not much higher than 1

Details

This function will produce simulated capture histories. It is designed to simulate traits under some selection pressure. As of now it does not simulate time or age dependent effects. See the examples for how to implement various types of selection.

Value

Returns a list:

$ch	a single-columned data.frame of capture histories, one row for each indvidual
$ch_split	a matrix of capture histories, one row for each individual
$p	a vector of recapture probabilities, one for each individual
$phi	a vector of survival probablities, one for each individual

Author(s)

John Waller

Examples

```
#' a simple example to start with...
N = 10 #' number of individual capture histories

trait = rnorm(N,0,1) #' make a normal distributed trait value, with mean 0 and sd 1

#' here we will simulate a trait with positive linear selection
#' while keeping our recapture probability, p ,constant
chObj = Simulate.CH(surv.form = 1 + 0.15*trait, p.constant = 1, N = N)
```

```
#' negative moderate linear selection
chObj = Simulate.CH(surv.form = 1 + -0.15*trait, p.constant = 1, N = N)

#' weaker negative linear selection
chObj = Simulate.CH(surv.form = 1 + -0.05*trait, p.constant = 1, N = N)

#' very strong negative linear slection
chObj = Simulate.CH(surv.form = 1 + -0.5*trait, p.constant = 1, N = N)

#' now lets add a stabilizing term
chObj = Simulate.CH(surv.form = 1 + -0.03*trait + -0.15*trait^2, p.constant = 1, N = N)

#' we can make selection disruptive simply by changing the sign
chObj = Simulate.CH(surv.form = 1 + -0.03*trait + 0.15*trait^2, p.constant = 1, N = N)

#' we can use multiple traits
trait1 = rnorm(N,0,1)
trait2 = rnorm(N,0,1)

#' negative linear selection on trait1 and positive selection on trait2
chObj = Simulate.CH(surv.form = 1 + -0.5*trait1 + 0.5*trait2, p.constant = 1, N = N)

#' stabilizing selection on trait1 and positive selection on trait2
chObj = Simulate.CH(surv.form = 1 + 0.03*trait1 + -0.5*trait^2 +
0.5*trait2, p.constant = 1, N = N)


#' We can also vary our intercept term
N = 10
trait1 = rnorm(N,0,1)

chObj = Simulate.CH(surv.form = 0 + 0.13*trait1, p.constant = 1, N = N)

phi = chObj$phi #' lets grab the survival probability
mean(phi) #' lets get the mean survival probability

#' Now lets raise the intercept value
chObj = Simulate.CH(surv.form = 2 + 0.13*trait1, p.constant = 1, N = N)

phi = chObj$phi
#' We see that the intercept term controls the mean survival probability
mean(phi) #' this value should be higher now


#'We can now play with recapture probability, p.constant
N = 10
trait = rnorm(N,0,1)

#' p.constant at 0.5, we see individuals in our population 50% of the time
chObj = Simulate.CH(surv.form = 1 + 0.15*trait, p.constant = 0.5, N = N)

#' p.constant at 0.1, we see individuals in our population 10% of the time
```

```
chObj = Simulate.CH(surv.form = 1 + 0.15*trait, p.constant = 0.1, N = N)


#' We can make recapture probability dependent on some trait in the same way
N = 10
trait = rnorm(N,0,1)

#' Here is a situation with weak linear selection but strong bias in detection probability
#' for a given trait
chObj = Simulate.CH(surv.form = 1 + 0.03*trait, p.form = 1 + 0.5*trait)


#' Finally we can make survival probability constant and recapture probability
#' dependent on some trait
chObj = Simulate.CH(p.form = 1 + 0.5*trait, surv.constant = 0.8)

#' One more thing, let's look at the structure chObj list
str(chObj)
chObj$ch #' is a data.frame of capture histories
chObj$ch_split #' is a matrix of capture histories
chObj$p #' is a vector of recapture probabilities
```

`Split.CH` *Split a capture history data.frame*

Description

Splits a data.frame into a matrix

Usage

```
Split.CH(ch)
```

Arguments

ch	a single-column data.frame of class character to split

Details

Mark-recapture software often requires data in split form, with each value in its own matrix cell. Here is a function to split a data.frame into a matrix with each column representing a capture occassion and each row representing an individual's capture history

Value

A matrix of 1's and 0's, a row for each individual.

Author(s)

John Waller

See Also

Collapse.CH

Examples

```
#' set up variables to be used by Simulate.CH for ten individuals
N = 10
x1 = rnorm(N) #' each trait is normally distributed

#' Run Simulate.CH with a constant recapture probability
chObj = Simulate.CH(surv.form = 1 + 0.15*x1, p.constant = 1, N = N)
str(chObj)
ch = chObj$ch #' grab our data.frame
ch #' it isn't split

Split.CH(ch) #' returns a matrix
```

Index