# Package: mpmi (via r-universe)

October 28, 2026

**Type** Package

**Title** Mixed-Pair Mutual Information Estimators

**Version** 0.43.2.1

**Date** 2019-04-22

**Author** Chris Pardy

**Maintainer** Chris Pardy <cooliomcdude@gmail.com>

**Description** Uses a kernel smoothing approach to calculate Mutual Information for comparisons between all types of variables including continuous vs continuous, continuous vs discrete and discrete vs discrete. Uses a nonparametric bias correction giving Bias Corrected Mutual Information (BCMI). Implemented efficiently in Fortran 95 with OpenMP and suited to large genomic datasets.

**License** GPL-3

**URL** https://r-forge.r-project.org/projects/mpmi/

**Depends** R(>= 3.6.2), KernSmooth

**LazyData** true

**NeedsCompilation** yes

**Repository** https://cran.r-universe.dev

**Date/Publication** 2023-07-05 08:37:38 UTC

**RemoteUrl** https://github.com/cran/mpmi

**RemoteRef** HEAD

**RemoteSha** 9c6049b502a0fd7cb0b3c6a18b4ec28acc19e4b0

## Contents

cmi . . . . . 2
cts . . . . . 4
disc . . . . . 4
dmi . . . . . 5
mmi . . . . . 6
mp . . . . . 8

**Index** 9

---

`cmi` *Calculate BCMI between a set of continuous variables*

---

## Description

This function calculates MI and BCMI between a set of continuous variables held as columns in a matrix. It also performs jackknife bias correction and provides a z-score for the hypothesis of no association. Also included are the *.pw functions that calculate MI between two vectors only. The *njk functions do not perform the jackknife and are therefore faster.

## Usage

```
cmi(cts, level = 3L, na.rm = FALSE, h, ...)
cminjk(cts, level = 3L, na.rm = FALSE, h, ...)
cmi.pw(v1, v2, h, ...)
cminjk.pw(v1, v2, h, ...)
```

## Arguments

| | |
|---|---|
| `cts` | The data matrix. Each row is an observation and each column is a variable of interest. Should be numerical data. |
| `level` | The number of levels used for plug-in bandwidth estimation (see the documentation for the KernSmooth package.) |
| `na.rm` | Remove missing values if TRUE. This is required for the bandwidth calculation. |
| `h` | A (double) vector of smoothing bandwidths, one for each variable. If missing this will be calculated using the dpik() function from the KernSmooth package. |
| `...` | Additional options passed to dpik() if necessary. |
| `v1` | A vector for the pairwise version |
| `v2` | A vector for the pairwise version |

## Details

The results of cmi() are in many ways similar to a correlation matrix, with each row and column index corresponding to a given variable. cminjk() and cminjk.pw() just returns the MI values without performing the jackknife. cmi.pw() and cminjk.pw() each only require two bandwidths, one for each variable. The number of processor cores used can be changed by setting the environment variable "OMP_NUM_THREADS" *before* starting R.

## Value

Returns a list of 3 matrices each of size ncol(cts) by ncol(cts)

| | |
|---|---|
| `mi` | The raw MI estimates. |
| `bcmi` | Jackknife bias corrected MI estimates (BCMI). These are each MI value minus the corresponding jackknife estimate of bias. |

zvalues Z-scores for each hypothesis that the corresponding BCMI value is zero. These have poor statistical properties but can be useful as a rough measure of the strength of association.

## Examples

```
################################################
# The USArrests dataset

# Matrix version
c1 <- cmi(USArrests)
lapply(c1, round, 2)

# Pairwise version
cmi.pw(USArrests[,1], USArrests[,2])

# Without jackknife
c2 <- cminjk(USArrests)
round(c2, 2)
cminjk.pw(USArrests[,1], USArrests[,2])

################################################
# A look at Anscombe's famous dataset.
par(mfrow = c(2,2))
plot(anscombe$x1, anscombe$y1)
plot(anscombe$x2, anscombe$y2)
plot(anscombe$x3, anscombe$y3)
plot(anscombe$x4, anscombe$y4)

cor(anscombe$x1, anscombe$y1)
cor(anscombe$x2, anscombe$y2)
cor(anscombe$x3, anscombe$y3)
cor(anscombe$x4, anscombe$y4)

cmi.pw(anscombe$x1, anscombe$y1)
cmi.pw(anscombe$x2, anscombe$y2)
cmi.pw(anscombe$x3, anscombe$y3)
# dpik() has some trouble with zero scale estimates on this one:
cmi.pw(anscombe$x4, anscombe$y4, scalest = "stdev")
################################################

################################################
# The highly collinear Longley dataset

pairs(longley, main = "longley data")
l1 <- cmi(longley)
lapply(l1, round, 2)

# Here we demonstrate the scale-invariance of MI.
# Note: Scaling can help stabilise estimates when there are
# difficulties with the bandwidth estimation, but is unnecessary
# here.
long2 <- scale(longley)
```

```
l2 <- cmi(long2)
lapply(l2, round, 2)

################################################
# See the vignette for large-scale examples.
```

---

`cts` *A group of simulated continuous variables*

---

## Description

50 observations on each of 100 variables. The data are simulated such that variables with similar indices are associated with the degree of association decaying as variables are further apart (i.e., a correlation or information matrix with have larger values near the diagonal). Details are given in the vignette.

## Usage

```
cts
```

## Format

A matrix with 50 rows and 100 columns

---

`disc` *A group of simulated categorical (discrete) variables*

---

## Description

50 observations on each of 75 categorical variables. These variables are designed to be similar to categorical single nucleotide polymorphism (SNP) data which have 3 categories (A, H and B where H represents a heterozygous mutation). There are no associations between any of the variables. The variables are stored as characters. See the vignette for details

## Usage

```
disc
```

## Format

A matrix of characters with 50 rows and 75 columns

dmi *Calculate BCMI for categorical (discrete) data*

## Description

This function calculates MI and BCMI between a set of discrete variables held as columns in a matrix. It also performs jackknife bias correction and provides a z-score for the hypothesis of no association. Also included are the *.pw functions that calculate MI between two vectors only. The *njk functions do not perform the jackknife and are therefore faster.

## Usage

```
dmi(dmat)
dminjk(dmat)
dmi.pw(disc1, disc2)
dminjk.pw(disc1, disc2)
```

## Arguments

| | |
|---|---|
| dmat | The data matrix. Each row is an observation and each column is a variable of interest. Should contain categorical data, all types of data will be coerced via factors to integers. |
| disc1 | A vector for the pairwise version |
| disc2 | A vector for the pairwise version |

## Details

The results of dmi() are in many ways similar to a correlation matrix, with each row and column index corresponding to a given variable. dminjk() and dminjk.pw() just returns the MI values without performing the jackknife. The number of processor cores used can be changed by setting the environment variable "OMP_NUM_THREADS" *before* starting R.

## Value

Returns a list of 3 matrices each of size `ncol(dmat)` by `ncol(dmat)`

| | |
|---|---|
| mi | The raw MI estimates. |
| bcmi | Jackknife bias corrected MI estimates (BCMI). These are each MI value minus the corresponding jackknife estimate of bias. |
| zvalues | Z-scores for each hypothesis that the corresponding bcmi value is zero. These have poor statistical properties but can be useful as a rough measure of the strength of association. |

## Examples

```
data(cars)

# Discretise the data first
d <- cut(cars$dist, breaks = 10)
s <- cut(cars$speed, breaks = 10)

# Discrete MI values
dmi.pw(s, d)

# For comparison, analysed as continuous data:
cmi.pw(cars$dist, cars$speed)

# Exploring a group of categorical variables
dat <- mtcars[, c("cyl","vs","am","gear","carb")]
discresults <- dmi(dat)
discresults

# Plot the relative magnitude of the BCMI values
diag(discresults$bcmi) <- NA
mp(discresults$bcmi)
```

---

# mmi *Calculate mixed-pair BCMI between a set of continuous variables and a set of discrete variables.*

---

## Description

This function calculates MI and BCMI between a set of continuous variables and a set of discrete variables (variables in columns). It also performs jackknife bias correction and provides a z-score for the hypothesis of no association. Also included are the *.pw functions that calculate MI between two vectors only. The *njk functions do not perform the jackknife and are therefore faster.

## Usage

```
mmi(cts, disc, level = 3L, na.rm = FALSE, h, ...)
mminjk(cts, disc, level = 3L, na.rm = FALSE, h, ...)
mmi.pw(cts, disc, h, ...)
mminjk.pw(cts, disc, h, ...)
```

## Arguments

| | |
|---|---|
| cts | The data matrix. Each row is an observation and each column is a variable of interest. Should be numerical data. (For the pairwise functions this should be a vector.) |
| disc | Matrix of discrete data, each row is an observation and each column is a variable. Will be coerced to integers. (For the pairwise functions this should be a vector.) |

| | |
|---|---|
| level | The number of levels used for plug-in bandwidth estimation (see the documentation for the KernSmooth package.) |
| na.rm | Remove missing values if TRUE. This is required for the bandwidth calculation. |
| h | A (double) vector of smoothing bandwidths, one for each variable. If missing this will be calculated using the dpik() function from the KernSmooth package. |
| ... | Additional options passed to dpik() if necessary. |

## Details

mminjk() and mminjk.pw() return just the MI values without performing the jackknife. mmi.pw() and mminjk.pw() only require one bandwidth for the continuous variable. The number of processor cores used can be changed by setting the environment variable "OMP_NUM_THREADS" *before* starting R.

## Value

Returns a list of 3 matrices each of size ncol(cts) by ncol(disc). Each row index represents a continuous variable and each column index a discrete variable.

| | |
|---|---|
| mi | The raw MI estimates. |
| bcmi | Jackknife bias corrected MI estimates (BCMI). These are each MI value minus the corresponding jackknife estimate of bias. |
| zvalues | z-scores for each hypothesis that the corresponding bcmi value is zero. These have poor statistical properties but can be useful as a rough measure of the strength of association. |

## Examples

```
##################################################
# A dataset with discrete and continuous variables

cts <- state.x77
disc <- data.frame(state.division,state.region)
summary(cts)
table(disc)
m1 <- mmi(cts, disc)
lapply(m1, round, 2)
# Division gives more information about the continuous variables than region.

# Here is one where both division and region show a strong association:
boxplot(cts[,6] ~ disc[,1])
boxplot(cts[,6] ~ disc[,2])

# In this case the states need to be divided into regions before a clear
# association can be seen:
boxplot(cts[,1] ~ disc[,1])
boxplot(cts[,1] ~ disc[,2])

# Look at associations within the continuous variables:
pairs(cts, col = state.region)
```

```
c1 <- cmi(cts)
lapply(c1, round, 2)

################################################
# A pairwise comparison

# Note that the ANOVA homoskedasticity assumption is not satisfied here.
boxplot(InsectSprays[,1] ~ InsectSprays[,2])
mmi.pw(InsectSprays[,1], InsectSprays[,2])

################################################
# Another pairwise comparison

boxplot(morley[,3] ~ morley[,1])
m2 <- mmi.pw(morley[,3], morley[,1])
m2

################################################
# See the vignette for large-scale examples.
```

---

mp *Matrix Plot*

---

## Description

Plot a matrix of values in the same order that it is stored (the usual mathematical way).

## Usage

```
mp(mat, ...)
```

## Arguments

| | |
|---|---|
| mat | A numeric matrix to be plotted |
| ... | Additional arguments to pass to image() |

## Details

The mp() function is a simple wrapper to image() with a few minor changes. The plot is flipped so that points correspond to their position in the matrix. Also, the colours are scaled so that red is the maximum value in the matrix and white is the minimum value.

## Examples

```
# From the vignette:
data(mpmidata)
ctsresult <- cmi(cts)
mp(ctsresult$bcmi)
```

# Index

cmi, 2
cminjk *(*cmi*)*, 2
cts, 4

disc, 4
dmi, 5
dminjk *(*dmi*)*, 5

mmi, 6
mminjk *(*mmi*)*, 6
mp, 8