Package ‘acss’

February 19, 2015

Type Package
Title Algorithmic Complexity for Short Strings
Version 0.2-5
Date 2014-11-23
Imports zoo
Depends R (>= 2.15.0), acss.data
Suggests effects, lattice
Description Main functionality is to provide the algorithmic complexity for short strings, an approximation of the Kolmogorov Complexity of a short string using the coding theorem method (see ?acss). The database containing the complexity is provided in the data only package acss.data, this package provides functions accessing the data such as prob_random returning the posterior probability that a given string was produced by a random process. In addition, two traditional (but problematic) measures of complexity are also provided: entropy and change complexity.

URL http://complexitycalculator.com/methodology.html
License GPL (>= 2)
Author Nicolas Gauvrit [aut],
Henrik Singmann [aut, cre],
Fernando Soler Toscano [ctb],
Hector Zenil [ctb]
Maintainer Henrik Singmann <singmann+acss@gmail.com>
NeedsCompilation no
Repository CRAN
Date/Publication 2014-11-25 08:45:18

R topics documented:

acss .................................................. 2
entropy ................................................. 5
exp1 .................................................. 6
Description

Functions to obtain the algorithmic complexity for short strings, an approximation of the Kolmogorov Complexity of a short string using the coding theorem method.

Usage

acss(string, alphabet = 9)

local_complexity(string, alphabet = 9, span = 5)

likelihood_d(string, alphabet = 9)

likelihood_ratio(string, alphabet = 9)

prob_random(string, alphabet = 9, prior= 0.5)

Arguments

string character vector containing the to be analyzed strings (can contain multiple strings).

alphabet numeric, the number of possible symbols (not necessarily actually appearing in str). Must be one of c(2, 4, 5, 6, 9) (can also be NULL or contain multiple values for acss()). Default is 9.

prior numeric, the prior probability that the underlying process is random.

span size of substrings to be created from string.

Details

The algorithmic complexity is computed using the coding theorem method: For a given alphabet size (number of different symbols in a string), all possible or a large number of random samples of Turing machines (TM) with a given number of states (e.g., 5) and number of symbols corresponding to the alphabet size were simulated until they reached a halting state or failed to end. The outputs of the TMs at the halting states produces a distribution of strings known as the algorithmic probability of the strings. The algorithmic coding theorem (Levin, 1974) establishes the connection between the complexity of a string $K(s)$ and its algorithmic probability $D(s)$ as:

$$K(s) \approx -\log_2(D(s))$$
This package accesses a database containing data on 4.5 million strings from length 1 to 12 simulated on TMs with 2, 4, 5, 6, and 9 symbols.

For a more detailed discussion see Gauvrit, Singmann, Soler-Toscano, and Zenil (2014), http://complexitycalculator.com/methodology.html, or references below.

Value

"acss" A matrix in which the rows correspond to the strings entered and the columns to the algorithmic complexity K and the algorithmic probability D of the string (see http://complexitycalculator.com/methodology.html).

"local_complexity" A list with elements corresponding to the strings. Each list contains a named vector of algorithmic complexities (K) of all substrings in each string with length span.

"likelihood_d" A named vector with the likelihoods for string given a deterministic process.

"likelihood_ratio" A named vector with the likelihood ratios (or Bayes factors) for string given a random rather than deterministic process.

"prob_random" A named vector with the posterior probabilities that for a random process given the strings and the provided prior for being produced by a random process (default is 0.5, which correspond to a prior of 1 - 0.5 = 0.5 for a deterministic process).

Note

The first time per session one of the functions described here is used, a relatively large dataset is loaded into memory which can take a considerable amount of time (> 10 seconds).

References


Examples

# WARNING: The first call to one of the functions
# discussed on this page loads a large data set
# and usually takes > 10 seconds. Stay patient.
acss(c("HEHHEE", "GHHGGGHH", "HSHSHSHSS"))
## K.9 D.9
## HEHHEE 23.38852 9.106564e-08
## GHHGGGHH 33.50168 8.222005e-11
## HSHSHSHSS 35.15241 2.616813e-11
acss(c("HEHHEE", "GHHGGGHH", "HSHSHSHSS"))["K.9"]
## [1] 23.38852 33.50168 35.15241
acss(c("HEHHEE", "GHHGGGHH", "HSHSHSHSS"), alphabet = 2)
## K.2 D.2
## HEHHEE 14.96921 3.117581e-05
## GHHGGGHH 25.60208 1.963387e-08
## HSHSHSHSS 26.90906 7.933321e-09
acss(c("HEHHEE", "GHHGGGHH", "HSHSHSHSS"), NULL)
## K.2 K.4 K.5 K.6 K.9
## HEHHEE 14.96921 18.55227 19.70361 20.75762 23.38852
## GHHGGGHHUE NA 31.75832 33.00795 34.27457 37.78935
## HSHSHSHSS 26.90906 29.37852 30.52566 31.76229 35.15241
## D.2 D.4 D.5 D.6
## HEHHEE 3.117581e-05 2.601421e-06 1.71176e-06 5.640722e-07
## GHHGGGHHUE NA 2.752909e-10 1.577555e-10 4.812021e-11
## HSHSHSHSS 7.933321e-09 1.432793e-09 6.469341e-10 2.745360e-10
## D.9
## HEHHEE 9.106564e-08
## GHHGGGHHUE 4.209915e-12
## HSHSHSHSS 2.618613e-11

## Not run:
likelihood_d(c("HTHTHTHT", "HTHTHTHT"), alphabet = 2)
## HTHTHTHT HHTHTHTT
## 0.010366951 0.003102718
likelihood_ratio(c("HTHTHTHT", "HTHTHTHT"), alphabet = 2)
## HTHTHTHT HHTHTHTT
## 0.3767983 1.2589769
prob_random(c("HTHTHTHT", "HTHTHTHT"), alphabet = 2)
## HTHTHTHT HHTHTHTT
## 0.2736772 0.5573217

## End(Not run)
local_complexity(c("01011010111", "GHHGGGHHUE"), alphabet = 5, span=5)
## $'01011010111'
## 01011 10110 01101 11010 10101 01011 10111
## $GHHGGGHHUE
## GHHG HHHGH HGGGH GHHG GHGHU HHHUE
entropy

local_complexity(c("01011010111", "GHHGGHUE"), span=7)
## $'01011010111'
## 0101101 1011010 0110101 1101011 1010111
##
## $GHHGGHUE
## GHHG GUH HGGHGUH GHHGUE
## 27.04623 26.86992 27.30871 27.84322

| entropy | Standard measures of complexity for strings |

Description

Functions to compute different measures of complexity for strings: Entropy, Second-Order Entropy, and Change Complexity

Usage

entropy(string)
entropy2(string)
change_complexity(string)

Arguments

string character vector containing the to be analyzed strings (can contain multiple strings for the entropy measures).

Details

For users who need advanced functions, a comprehensive package computing various versions of entropy estimators is available entropy. For users who just need first and second-order entropy and which to apply them to short string, the acss package provides two functions: entropy (first-order entropy) and entropy2 second-order entropy.

Change complexity (change_complexity) assesses cognitive complexity or the subjective perception of complexity of a binary string. It has been comprehensively defined by Aksentijevic and Gibson (2012). Although the algorithm will work with any number of symbols up to 10, the rationale of Change Complexity only applies to binary strings.

Value

numeric, the complexity of the string. For entropy and entropy2 of the same length as string. change_complexity currently only works with inputs of length 1.
References

Examples

```r
strings1 <- c("010011010001", "001020328837", "0000000000")
strings2 <- c("001011", "01213", "0101010101")

entropy(strings1)
entropy("XYXXYYXXXXY") # "same" string as strings1[1]
entropy(c("HUXHEGGTE", "EGGHHU"))

entropy2(strings1)
entropy2("XYXXYYXXXXY")

entropy2(strings2)

change_complexity(strings1)
change_complexity("XYXXYYXXXXY")
```

---

**exp1**

*Data from Experiment 1 in Gauvrit, Singmann, Soler-Toscano & Zenil*

**Description**

34 participants were asked to produce at their own pace a series of 10 symbols among "A", "B", "C", and "D" that would "look as random as possible, so that if someone else sees the sequence, she will believe it is a truly random one".

**Usage**

exp1

**Format**

A data.frame with 34 rows and 2 variables.

**Source**

Examples

```r
# load data
data(exp1)

# summary statistics
nrow(exp1)
summary(exp1$age)
mean(exp1$age)
sd(exp1$age)

## Not run:
# this uses code from likelihood_d() to calculate the mean complexity K
# for all strings of length 10 with alphabet = 4:
tmp <- acss_data[nchar(rownames(acss_data)) == 10, "K.4", drop = FALSE]
tmp <- tmp[!is.na(tmp[,"K.4"]),,drop = FALSE]
tmp$count <- count_class(rownames(tmp), alphabet = 4)
(mean_k <- with(tmp, sum(K.4*count)/sum(count))

t.test(acss(exp1$string, 4)[,"K.4", mu = mean_k)

## End(Not run)
```

---

**exp2**  
*Data from Experiment 2 in Gauvrit, Singmann, Soler-Toscano & Zenil*

Description

Responses of one participant (42 years old) to 200 randomly generated strings of length 10 from an alphabet of 6 symbols. For each string, the participant was asked to indicate whether or not the string appears random or not.

Usage

exp2

Format

A data.frame with 200 rows and 2 variables.

Source

Examples

```r
# load data
data(exp2)

exp2$K <- acss(exp2$string, 6)[,"K.6"]
m_log <- glm(response ~ K, exp2, family = binomial)
summary(m_log)

# odds ratio of K:
exp(coef(m_log)[2])

# calculate threshold of 0.5
(threshold <- -coef(m_log)[1]/coef(m_log)[2])
require/effects
require/lattice
plot(Effect("K", m_log), rescale.axis = FALSE, ylim = c(0, 1))
trellis.focus("panel", 1, 1)
panel.lines(rep(threshold, 2), c(0, 0.5), col = "black", lwd = 2.5, lty = 3)
panel.lines(c(33,threshold), c(0.5, 0.5), col = "black", lwd = 2.5, lty = 3)
trellis.unfocus()
```

(matthews2013)

Data from Experiment 1 in Matthews (2013)

Description

Mean responses on a 6-point scale ("definitely random" to "definitely not random") of participants to 216 strings of length 21.

Usage

matthews2013

Format

A data.frame with 216 rows and 3 variables.

Source

Examples

```r
## Not run:
data(matthews2013)

spans <- 3:11
# note, the next loop takes more than 5 minutes.
for (i in spans) {
  matthews2013[,paste0("K2_span", i)] <-
  sapply(local_complexity(matthews2013$string, alphabet=2, span = i), mean)
}

lm_list <- vector("list", 8)
for (i in seq_along(spans)) {
  lm_list[[i]] <- lm(as.formula(paste0("mean ~ K2_span", spans[i])), matthews2013)
}

plot(spans, sapply(lm_list, function(x) summary(x)$r.squared), type = "o")

# do more predictors increase fit?
require(MASS)
m_initial <- lm(mean ~ 1, matthews2013)
m_step <- stepAIC(m_initial, 
  scope = as.formula(paste("~", paste(paste0("K2_span", spans), 
  collapse = "+"))))
  summary(m_step)

m_initial2 <- lm(as.formula(paste("mean ~ ", paste(paste0("K2_span", spans), 
  collapse = "+")))), matthews2013)
m_step2 <- stepAIC(m_initial2)
  summary(m_step2)

## End(Not run)
```

---

normalize_string  

*Helper functions for calculating cognitive complexity.*

**Description**

normalize_string takes a character vector and normalizes its input using the symbols 0, 1, 2...9. count_class takes a character vector and an integer alphabet (with the restriction that the number of different symbols in the character vector doesn't exceed alphabet) and returns the total number of strings that are equivalent to the input when normalized and considering alphabet. alternations returns the number of alternations of symbols in a string.

**Usage**

```r
normalize_string(string)
```
normalize_string

count_class(string, alphabet)

alternations(string, proportion = FALSE)

Arguments

string    character vector containing the to be analyzed strings (can contain multiple strings).
alphabet  numeric, the number of possible symbols (not necessarily actually appearing in string).
proportion boolean, indicating if the result from alternation should be given as a proportion (between 0 and 1) or the raw number of alternations (default is FALSE corresponding to raw values).

Details

nothing yet.

Value

normalize_string A normalized vector of strings of the same length as string.
count_class  A vector of the same length as string with the number of possible equivalent strings when string is normalized and considering alphabet.
alternations A vector with the number (or proportion) of alternations of the same length as string

Examples

#normalize_string:
normalize_string(c("HUHHEGGTE", "EGGHHU"))

normalize_string("293948837163536")

# count_class
count_class("010011",2)

count_class("332120",4)

count_class(c("HUHHEGGTE", "EGGHHU"), 5)
count_class(c("HUHHEGGTE", "EGGHHU"), 6)

# alternations:
alternations("0010233")
alternations("0010233", proportion = TRUE)

alternations(c("HUHHEGGTE", "EGGHHU"))
alternations(c("HUHHEGGTE", "EGGHHU"), proportion = TRUE)
Index

*Topic **dataset**
  - exp1, 6
  - exp2, 7
  - matthews2013, 8

acss, 2
alternations (normalize_string), 9

change_complexity (entropy), 5
count_class (normalize_string), 9

entropy, 5
entropy2 (entropy), 5
exp1, 6
exp2, 7

likelihood_d (acss), 2
likelihood_ratio (acss), 2
local_complexity (acss), 2

matthews2013, 8

normalize_string, 9

prob_random (acss), 2