Package 'preseqR'

June 26, 2014

Type Package

Title Predicting Library Complexity

Version 1.0

Date 2014-06-26

Author Andrew D. Smith, Timothy Daley and Chao Deng

Maintainer Chao Deng <chaodeng@usc.edu>

Description This is a R package to make the functionality of Preseq available in the R statistical computing environment.

License GPL-3

Keyword Library, Complexity, Rational Function

R topics documented:

`bootstrap.complex.curve`
Complexity curve

Description

The function estimates the complexity curve of a library when its histogram is provided. Bootstrap is add to help the approximated rational function estimate the curve and make a confidence interval.

Usage

```
bootstrap.complex.curve(hist, times = 100, di = 0, mt = 100, ss = 1e+06,
mv = 1e+10, max.extrapolation = 1e+10)
```

Arguments

hist	A histogram. It can be either a file name or a count vector of the histogram. For histogram file, it contains two columns. The first column is frequencies of molecules. Values of frequency should be at least one. For each given frequency, the second column is the number of molecules with that frequency.
times	The number of resampling times as a bootstrap process.
di	Diagonal value for a constructed continued fraction.
mt	Maximum number of parameters in a continued fraction.
ss	Step size of sampling points along a library.
mv	The maximum value to train a continued fraction.
max.extrapolation	The maximum possible value to extrapolate

Value

yield.estimates	Yields of distinct molecules given a sample size list as the first column
LOWER_0.95CI	Lower bound for a 95% confident interval
UPPER_0.95CI	Upper bound for a 96% confident interval

Author(s)

Chao Deng

References

http://smithlabresearch.org/software/preseq/

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (hist, times = 100, di = 0, mt = 100, ss = 1e+06,
    mv = 1e+10, max.extrapolation = 1e+10)
{
    if (mode(hist) == "character") {
hist.count = preseqR.read.hist(hist);
}
else {
hist.count = hist;
}
    hist.count = preseqR.read.hist(hist.file)
    total.sample = 0
    for (i in 1:length(hist.count)) total.sample <- total.sample +
        i * hist.count[i]
```

```
    if (times == 1) {
        out <- preseqR.continued.fraction.estimate(hist.count,
            di, mt, ss, mv, max.extrapolation)
        if (!is.null(out)) {
            return(out$yield.estimates)
        }
        else {
            return()
        }
    }
    else if (times > 1) {
WER_0.95CI
        N = 0
        count = 0
        step.size = 0
        estimates = matrix(data = NA, nrow = max.extrapolation/ss,
            ncol = times, byrow = FALSE)
        for (i in 1:as.integer(times)) {
            sample = preseqR.hist.sample(hist.count, as.integer(total.sample),
                replace = TRUE)
            hist = preseqR.sample2hist.count(sample, replace = TRUE)
            out <- preseqR.continued.fraction.estimate(hist,
                di, mt, ss, mv, max.extrapolation)
            if (!is.null(out)) {
                count <- count + 1
                N = length(out$yield.estimates$yields)
                step.size = out$step.size
                estimates[, i][1:N] = out$yield.estimates$yields;
            }
        }
        if (N == 0) {
            write("can not make prediction based on the given histogram",
                stderr())
            return()
        }
        if (count < BOOTSTRAP.factor * times) {
            write("fail to bootstrap since the histogram is poor",
                stderr())
            return()
        }
        index = step.size * (1:N)
        mean = apply(estimates[1:N, ], 1, mean, na.rm = TRUE)
        variance = apply(estimates[1:N, ], 1, var, na.rm = TRUE)
        n = as.vector(apply(estimates, 1, function(x) length(which(!is.na(x)))))
        n = n[1:N]
        left.interval = mean - qnorm(0.975) * sqrt(variance / n);
        right.interval = mean + qnorm(0.975) * sqrt(variance / n);
        yield.estimates = list(sample.size = index, yields = yield.estimates)
        result = list(yield.estimates, left.interval, right.interval);
        names(result) = c("yield.estimates", "LOWER_0.95CI", "UPPER_0.95CI");
return(result);
   }
    else {
```

```
        write("the paramter times should be at least one", stderr())
        return()
    }
  }
```

`preseqR.continued.fraction.estimate`
A function to predict library complexity

Description

preseqR.continued.fraction.estimate creates an continued fraction to estimate the number of distinct molecules, reads, or species given its histogram. It also provides a complexity curve to describe the complexity of the capture-recapture data.

Usage

```
preseqR.continued.fraction.estimate(hist, di = 0, mt = 100, ss = 1e+06,
mv = 1e+10, max.extrapolation = 1e+10)
```

Arguments

`hist`	A histogram. It can be either a file name or a count vector of the histogram. For histogram file, it contains two columns. The first column is frequencies of molecules. Values of frequency should be at least one. For each given frequency, the second column is the number of molecules with that frequency.
`di`	Diagonal value for a constructed continued fraction.
`mt`	Maximum number of parameters in a continued fraction.
`ss`	Step size of sampling points along a library.
`mv`	The maximum value to train a continued fraction.
`max.extrapolation`	The maximum possible value to extrapolate

Value

`CF`	All components of a continued fraction. ps.coeffs is the coeffiencies of a power series, which the continued fraction estimates; co.coeffs is the coeffiencies of the continued fraction; offset.coeffs is offset coeffiencies of the contined fraction; di is the diagonal value of the continued fraction; de is the degree of the contined fracton
`yield.estimates`	Yields of distinct molecules given a sample size list as the first column

Author(s)

Chao Deng

References

http://smithlabresearch.org/software/preseq/

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (hist, di = 0, mt = 100, ss = 1e+06, mv = 1e+10, max.extrapolation = 1e+10)
{
    if (mode(hist) == "character") {
        hist.count = preseqR.read.hist(hist)
    }
    else {
        hist.count = hist
    }
    MIN_REQUIRED_TERMS = 4
    total.sample = 0
    for (i in 1:length(hist.count)) total.sample <- total.sample +
        i * hist.count[i]
    step.size = ss
    if (step.size > total.sample) {
        yield.estimates = vector(mode = "numeric", length = 0)
        starting.size = step.size
    }
    else {
        if (step.size < (total.sample/20)) {
            step.size = max(step.size, step.size * round(total.sample/(20 *
                step.size)))
            m = paste("adjust step size to", toString(step.size),
                "\n", sep = " ")
            write(m, stderr())
        }
        out = preseqR.interpolate.distinct(hist.count, step.size)
        yield.estimates = out$yield.estimates
        starting.size = out$sample.size
    }
    counts.before.first.zero = 1
    while (as.integer(counts.before.first.zero) <= length(hist.count) &&
        hist.count[counts.before.first.zero] != 0)
counts.before.first.zero <- counts.before.first.zero + 1
    mt = min(mt, counts.before.first.zero - 1)
    mt = mt - (mt%%2)
    if (mt < MIN_REQUIRED_TERMS) {
        m = paste("max count before zero is les than min required count (4), ",
            "sample not sufficiently deep or duplicates removed",
            sep = "")
        write(m, stderr())
        return()
    }
```

```
    if (goodtoulmin.2x.extrap(hist.count) < 0) {
        m = paste("Library expected to saturate in doubling of size, ",
            "unable to extrapolate", sep = "")
        write(m, stderr())
        return()
    }
    hist.count = c(0, hist.count)
    out <- .C("c_continued_fraction_estimate", as.double(hist.count),
        as.integer(length(hist.count)), as.integer(di), as.integer(mt),
        step.size = as.double(step.size), as.double(mv),
ps.coeffs = as.double(vector(mode = "numeric", length = MAXLENGTH)),
ps.coeffs.l = as.integer(0),
        cf.coeffs = as.double(vector(mode = "numeric", length = MAXLENGTH)),
        cf.coeffs.l = as.integer(0),
offset.coeffs = as.double(vector(mode = "numeric", length = MAXLENGTH)),
    diagonal.idx = as.integer(0),
        degree = as.integer(0), is.valid = as.integer(0))
    if (!out$is.valid) {
        write("Fail to construct and need to bootstrap to obtain estimates",
            stderr())
        return()
    }
    length(out$ps.coeffs) = out$ps.coeffs.l
    length(out$cf.coeffs) = out$cf.coeffs.l
    length(out$offset.coeffs) = as.integer(abs(out$diagonal.idx))
    CF = list(out$ps.coeffs, out$cf.coeffs, out$offset.coeffs,
        out$diagonal.idx, out$degree)
    names(CF) = c("ps.coeffs", "cf.coeffs", "offset.coeffs",
        "diagonal.idx", "degree")
    if (starting.size > max.extrapolation) {
index = as.integer(out$step.size) * (1: length(yield.estimates));
yield.estimates = list(sample.size = index, yields = yield.estimates);
        result = list(CF, yield.estimates, out$step.size)
        names(result) = c("continued.fraction", "yield.estimates",
            "step.size")
        return(result)
    }
    est <- preseqR.extrapolate.distinct(hist.count, CF, (starting.size -
        total.sample)/total.sample, out$step.size/total.sample,
        (max.extrapolation - total.sample)/total.sample)
    est = est[-1]
    yield.estimates = c(yield.estimates, est)
index = as.integer(out$step.size) * (1: length(yield.estimates));
yield.estimates = list(sample.size = index, yields = yield.estimates);
    result = list(CF, yield.estimates, out$step.size)
    names(result) = c("continued.fraction", "yield.estimates",
        "step.size")
    return(result)
  }
```