

# Package ‘VLMC’

August 19, 2024

**Version** 1.4-4

**VersionNote** Released 1.4-3-1 on 2019-04-29

**Date** 2024-08-14

**Title** Variable Length Markov Chains ('VLMC') Models

**Description** Functions, Classes & Methods for estimation, prediction, and simulation (bootstrap) of Variable Length Markov Chain ('VLMC') Models.

**Imports** stats, MASS

**Suggests** astda

**SuggestsNote** {astda} mentioned in docu only.

**BuildResaveData** no

**License** GPL (>= 2)

**NeedsCompilation** yes

**Author** Martin Maechler [aut, cre] (<<https://orcid.org/0000-0002-8685-9910>>)

**Maintainer** Martin Maechler <maechler@stat.math.ethz.ch>

**Repository** CRAN

**Date/Publication** 2024-08-19 09:50:10 UTC

## Contents

alpha2int . . . . .	2
alphabet . . . . .	3
as.dendrogram.vlmc . . . . .	3
bnrf1 . . . . .	4
deviance.vlmc . . . . .	5
draw.vlmc . . . . .	6
id2ctxt . . . . .	8
int2char . . . . .	8
logLik . . . . .	9
OZrain . . . . .	11
predict.vlmc . . . . .	12
prt.vvec . . . . .	14

RCplot . . . . .	15
residuals.vlmc . . . . .	17
simulate.vlmc . . . . .	18
summary.vlmc . . . . .	19
vlmc . . . . .	20
vlmc.version . . . . .	23
vlmctree . . . . .	23

<b>Index</b>	<b>25</b>
--------------	-----------

---

alpha2int	<i>'Single Character' &lt;-&gt; Integer Conversion for Discrete Data</i>
-----------	--

---

## Description

Simple conversion functions for discrete data (e.g., time series), between  $0:k$  integers and *single* letter characters.

## Usage

```
alpha2int(x, alpha)
int2alpha(i, alpha)
```

## Arguments

x	character vector of single letters.
alpha	the alphabet, as one character string.
i	integer vector of numbers in $0:k$ .

## Value

alpha2int(x,\*) returns an **integer** vector of the same length as x, consisting of values from  $0:k$  where  $k + 1$  is the length of the alphabet, `nchar(alpha)`.

int2alpha(i,\*) returns a vector of *single letter character* of the same length as i.

## See Also

[vlmc](#), and [int2char\(\)](#) and its inverse, [char2int\(\)](#), both working with multi-character strings instead of vectors of single characters; further, [alphabet](#).

## Examples

```
alphabet <- "abcdefghijkl"
(ch <- sample(letters[1:10], 30, replace = TRUE))
(ic <- alpha2int(ch, alphabet))
stopifnot(int2alpha(ic, alphabet) == ch)
```

---

alphabet

*The Alphabet in Use*

---

### Description

Return the alphabet in use, as a vector of “characters”.

### Usage

```
alphabet(x, ...)  
## S3 method for class 'vlmc'  
alphabet(x, ...)
```

### Arguments

x                    any R object, currently only available for [vlmc](#) ones.  
...                   potential further arguments passed to and from methods.

### Value

a [character](#) vector, say *r*, with length equal to the alphabet size. Currently, typically all *r*[*i*] are strings of just one character.

### See Also

[alpha2int](#) for conversion to and from integer codings.

### Examples

```
data(bnrf1)  
vb <- vlmc(bnrf1EB, cutoff = 5)  
alphabet(vb) # |--> "a" "c" "g" "t"
```

---

as.dendrogram.vlmc

*Dendrogram Construction from VLMCs*

---

### Description

This is a method for the `as.dendrogram` generic function

### Usage

```
## S3 method for class 'vlmc'  
as.dendrogram(object, ...)
```

**Arguments**

object            a `v1mc` object.  
 ...              further arguments passed to and from methods.

**Value**

An object of class `dendrogram`, i.e. a nested list described on that page.

**See Also**

[as.dendrogram](#), [plot.dendrogram](#).

**Examples**

```
data(presidents)
dpr <- factor(cut(presidents, c(0,45,70,100)), exclude=NULL)# NA = 4th level
(v1mc.pres <- v1mc(dpr))
draw(v1mc.pres)
(dv.dpr <- as.dendrogram(v1mc.pres))
str(dv.dpr)
str(unclass(dv.dpr))

plot(dv.dpr, type = "tr", nodePar = list(pch=c(1,16), cex = 1.5))

## Artificial example
f1 <- c(1,0,0,0) ; f2 <- rep(1:0, 2)
(dt1 <- c(f1,f1,f2,f1,f2,f1))
(v1mc.dt1c01 <- v1mc(dts = dt1, cutoff.prune = 0.1))
(dv1mc <- as.dendrogram(v1mc.dt1c01))
str(dv1mc)
## not so useful:
plot(dv1mc, nodePar= list(pch=c(1,16)))
## complete disaster:
plot(dv1mc, type = "tr", nodePar= list(pch=c(1,16)))

## but this is not (yet) so much better (want the same angles to left
## and right!!
plot(dv1mc, type = "tr", nodePar = list(pch=c(1,16)), center=TRUE,
      main = format(v1mc.dt1c01$call))
mtext(paste("dt1 =", gsub(" ", "", deparse(dt1,width=100))))
```

---

 bnrf1

*BNRF1 Gene DNA sequences: Epstein-Barr and Herpes*


---

**Description**

Two gene DNA data “discrete time series”,

bnrf1EB the BNRF1 gene from the Epstein-Barr virus,

bnrf1HV the BNRF1 gene from the herpes virus.

**Usage**

```
data(bnrf1)
```

**Format**

The EB sequence is of **length** 3954, whereas the HV has 3741 nucleotides. Both are **R factors** with the four levels `c("a", "c", "g", "t")`.

**Author(s)**

Martin Maechler (original packaging for R).

**Source**

See the references; data used to be at <https://anson.ucdavis.edu/~shumway/tsa.html>, and are now available in CRAN package **astsa**, e.g., [bnrf1ebv](#).

**References**

Shumway, R. and Stoffer, D. (2000) *Time Series Analysis and its Applications*. Springer Texts in Statistics.

**Examples**

```
data(bnrf1)
bnrf1EB[1:500]
table(bnrf1EB)
table(bnrf1HV)
n <- length(bnrf1HV)
table(t = bnrf1HV[-1], "t-1" = bnrf1HV[-n])

plot(as.integer(bnrf1EB[1:500]), type = "b")

## Simplistic gene matching:
percent.eq <- sapply(0:200,
  function(i) 100 * sum(bnrf1EB[(1+i):(n+i)] == bnrf1HV)/n)
plot.ts(percent.eq)
```

---

deviance.vlmc

---

*Compute the Deviance of a Fitted VLMC Object*


---

**Description**

Compute the Deviance, i.e.,  $-2 \log[\text{likelihood}^*]$  of a fitted VLMC object. The log-likelihood is also known as “entropy”.

**Usage**

```
## S3 method for class 'vlmc'
deviance(object, ...)
```

**Arguments**

```
object      typically the result of vlmc(.).
...         possibly further arguments (none at the moment).
```

**Value**

A number, the deviance, i.e.,  $-2\log.likelihood(*)$ . where the `log.likelihood` is really what we currently have as `entropy()`.

**Author(s)**

Martin Maechler

**See Also**

[entropy](#), [vlmc](#), [residuals.vlmc](#)

**Examples**

```
example(vlmc)
deviance(vlmc.pres)

devianceR <- function(object)
{
  dn <- dimnames(pr <- predict(object))
  -2 * sum(log(pr[cbind(2:nrow(pr), match(dn[[1]][-1], dn[[2]]))]))
}
all.equal(deviance(vlmc.pres), devianceR(vlmc.pres), tol = 1e-14)
```

---

draw.vlmc

*Draw a "VLMC" Object (in ASCII) as Tree*

---

**Description**

Draws a `vlmc` object, typically the result of `vlmc(.)`, to the R console, using one line per node.

**Usage**

```
draw(x, ...)
## S3 method for class 'vlmc'
draw(x, kind = 3, flag = TRUE, show.hidden = 0,
     cumulative = TRUE, delta = cumulative, debug = FALSE, ...)
```

**Arguments**

x	typically the result of <code>vlmc(. .)</code> .
kind	integer code for the “kind of drawing”, in {0,1,2,3}.
flag	logical; ..
show.hidden	integer code; if not 0, give some indications about hidden (final) nodes .....
cumulative	logical indicating if the cumulative counts should be shown for nonterminal nodes; the ‘delta’s can only be computed from the cumulative counts, i.e., <code>cumulative = FALSE</code> should be used only by the knowing one.
delta	logical indicating if delta, i.e. $\delta(n, p(n))$ should be computed and printed for each (non-root) node $n$ with parent $p(n)$ . Note that this does not really make sense when <code>cumulative = FALSE</code> .
debug	logical; if TRUE, some extraneous progress information is printed to the R console.
...	(potentially more arguments)

**Details**

.....  
 .....

Note that the counts internally are stored “non-cumulatively”, i.e., as *difference* counts which is useful for likelihood (ratio) computations. In the internal C code, the *difference* counts are originally computed by the `comp_difference()` function after tree generation. `draw(*, cumulative = TRUE)` internally calls the C function `cumulate()` for the cumulative sums.

**Value**

nothing is returned.

**Author(s)**

Martin Maechler

**See Also**

[vlmc](#).

**Examples**

```
example(vlmc)
draw(vlmc.dt1c01)
draw(vlmc.dt1c01, flag = FALSE)
draw(vlmc.dt1c01, kind = 1)
draw(vlmc.dt1)
draw(vlmc.dt1, show = 3)
draw(vlmc.dt1, cumulative = FALSE)
```

---

**id2ctxt** *VLMC Context ID Conversion*


---

**Description**

Utility for converting a `vlmc` state ID to the corresponding context. Of rare interest to the average user.

**Usage**

```
id2ctxt(id, m=nchar(alpha), alpha=NULL)
```

**Arguments**

<code>id</code>	integer, a context ID such as optionally returned by <code>predict.vlmc</code> .
<code>m</code>	integer, the alphabet length. Defaults to <code>nchar(alpha)</code> , the alphabet size if that is given.
<code>alpha</code>	alphabet string

**Value**

a list (if `alpha` is not specified) or character vector of the same length as `id`, giving the context (as integer vector or single string) of the corresponding `id`

**See Also**

`predict.vlmc(*, type = "ID")`.

**Examples**

```
id2ctxt(c(2,3,5,9), alpha = "Ab")
str(id2ctxt(c(2,3,5,9), 2))
```

---

**int2char** *Character - Integer Conversion*


---

**Description**

Simple conversion utilities for character to integer conversion and vice versa.

**Usage**

```
int2char(i, alpha)
char2int(x, alpha)
```



**Arguments**

i	integer vectors, typically in $0:m$ when alpha has $m + 1$ letters.
alpha	character string with several letters, representing the alphabet.
x	character string, typically with letters from alpha.

**Value**

int2char() gives a string (length 1 character) with as many characters as length(i), by 0-indexing into the alphabet alpha.

char2int() gives an integer vector of length nchar(x) of integer codes according to alpha (starting at 0!).

**See Also**

[int2alpha\(\)](#) (which is used by int2char) and its inverse, [alpha2int\(\)](#), both working with vectors of *single* characters instead of multi-character strings.

**Examples**

```
char2int("vlmc", paste(letters, collapse=""))
int2char(c(0:3, 3:1), "abcd")
int2char(c(1:0,3,3), "abc") # to eat ;-)
```

---

logLik

*Log Likelihood of and between VLMC objects*


---

**Description**

Compute the log-likelihood or “entropy” of a fitted `vlmc` object. This is a method for the generic [logLik](#).

**Usage**

```
entropy(object)
## S3 method for class 'vlmc'
logLik(object, ...)
entropy2(ivlmc1, ivlmc2, alpha.len = ivlmc1[1])
```

**Arguments**

object	typically the result of <code>vlmc(...)</code> .
ivlmc1, ivlmc2	two <code>vlmc</code> (sub) trees, see <code>vlmc</code> .
alpha.len	positive integer specifying the alphabet length.
...	(potentially more arguments; required by generic)

**Details**

The `logLik.vlmc()` method computes the log likelihood for a fitted `vlmc` object. `entropy` is an alias for `logLik` for reasons of back compatibility.

`entropy2` is less clear ... .. `[[[ FIXME ]]]` ... ..

**Value**

a negative number, in some contexts typically further divided by  $\log(x\$alpha.len)$ .

Note that the `logLik` method is used by the default method of the `AIC` generic function (from R version 1.4.x), and hence provides `AIC(object)` for `vlmc` objects. Also, since `vlmc` version 1.3-13, `BIC()` works as well.

**Author(s)**

Martin Maechler

**See Also**

[deviance.vlmc](#), [vlmc](#), [draw.vlmc](#).

**Examples**

```
dd <- cumsum(rpois(999, 1.5)) %% 10
(vd <- vlmc(dd))
entropy(vd)# the bare number
logLik(vd)
logLik(vdL <- vlmc(dd, cutoff = 3))
entropy2(vd $vlmc.vec,
         vdL$vlmc.vec)

## AIC model selection:
f1 <- c(1,0,0,0) # as in example(vlmc)
f2 <- rep(1:0,2)
(dt1 <- c(f1,f1,f2,f1,f2,f2,f1))
AIC(print(vlmc(dt1)))
AIC(print(vlmc(dt1, cutoff = 2.6)))
AIC(print(vlmc(dt1, cutoff = 0.4)))# these two differ ``not really''
AIC(print(vlmc(dt1, cutoff = 0.1)))

## Show how to compute it from the fitted conditional probabilities :
logLikR <- function(x) {
  dn <- dimnames(pr <- predict(x))
  sum(log(pr[cbind(2:nrow(pr), match(dn[[1]][-1], dn[[2]]))]))
}

all.equal( logLikR(vd),
          c(logLik(vd)), tol=1e-10) # TRUE, they do the same

## Compare different ones: [cheap example]:
example(draw)
for(n in ls())
```

```

if(is.vlmc(get(n))) {
  vv <- get(n)
  cat(n, ":", formatC(logLik(vv) / log(vv$alpha.len),
                    format= "f", wid=10), "\n")
}

```

---

OZrain

*Daily Rainfall in Melbourne, Australia, 1981-1990*


---

### Description

Amount of daily rainfall in Melbourne, Australia, 1981-1990, measured in millimeters. The amounts are integers with many zeros and three days of more than 500mm rain.

### Usage

```
data(OZrain)
```

### Format

A time-series of length 3653 with the amount of daily rainfall in mm. Because of the two leap years 1984 and '88, we have constructed it with `ts(*, start=1981, frequency=365.25, end = 1981 + (3653 - 1)/365.25)`.

### Note

There must be one extra observation since for the ten years with two leap years, there are only 3652 days. In 61 out of 100 days, there's no rain.

### Source

'rainfall.dat' in Rob J. Hyndman's *Time Series Data Library*, currently available at <https://pkg.yangzhuoranyang.com/tsdl/> originally, Australian Bureau of Meteorology, <https://www.abs.gov.au>.

### Examples

```

data(OZrain)
(n <- length(OZrain)) ## should be 1 more than
ISOdate(1990,12,31) - ISOdate(1981, 1,1)## but it's 2 ..

has.rain <- OZrain > 0

summary(OZrain[has.rain])# Median = 18, Q3 = 50
table(rain01 <- as.integer(has.rain))
table(rain4c <- cut(OZrain, c(-.1, 0.5, 18.5, 50.1, 1000)))

```

```

AIC(v1 <- vlmc(rain01))# cutoff = 1.92
AIC(v00 <- vlmc(rain01, cut = 1.4))
AIC(v0 <- vlmc(rain01, cut = 1.5))

hist(OZrain)
hist(OZrain, breaks = c(0,1,5,10,50,1000), xlim = c(0,100))

plot(OZrain, main = "Rainfall 1981-1990 in Melbourne")
plot(OZrain, log="y", main = "Non-0 Rainfall [LOG scale]")

l0Z <- lowess(log10(OZrain[has.rain]), f= .05)
lines(time(OZrain)[has.rain], 10^l0Z$y, col = 2, lwd = 2)

```

---

predict.vlmc

*Prediction of VLMC for (new) Series*


---

### Description

Compute predictions on a fitted VLMC object for each (but the first) element of another discrete time series. Computes by default a matrix of prediction probabilities. The argument `type` allows other predictions such as the most probable "class" or "response", the context length (tree "depth"), or an "ID" of the corresponding context.

### Usage

```

## S3 method for class 'vlmc'
predict(object, newdata,
        type = c("probs", "class", "response", "id.node", "depth", "ALL"),
        se.fit = FALSE,
        allow.subset = TRUE, check.alphabet=TRUE,
        ...)
## S3 method for class 'vlmc'
fitted(object, ...)

```

### Arguments

<code>object</code>	typically the result of <code>vlmc(...)</code> .
<code>newdata</code>	a discrete "time series", a numeric, character or factor, as the <code>dts</code> argument of <code>vlmc(...)</code> .
<code>type</code>	character indicating the type of prediction required, options given in the <i>Usage</i> section above, see also the <i>Value</i> section below. The default "probs" returns a matrix of prediction probabilities, whereas "class" or "response" give the corresponding most probable class. The value of this argument can be abbreviated.
<code>se.fit</code>	a switch indicating if standard errors are required. — NOT YET supported — .

allow.subset logical; if TRUE, newdata may not have all different “alphabet letters” used in x.  
 check.alphabet logical; if TRUE, consistency of newdata’s alphabet with those of x is checked.  
 ... (potentially further arguments) required by generic.

## Value

Depending on the type argument,

“probs” an  $n \times m$  matrix pm of (prediction) probabilities, i.e., all the rows of pm sum to 1.  
 pm[i, k] is  $\hat{P}[Y_i = k | Y_{i-1}, \dots]$  (and is therefore NA for i=1). The dimnames of pm are the values of newdata[] and the alphabet letters k.

“class”, “response” the corresponding most probable value of Y[]; as factor for “class” and as integer in 0: (m-1) for type = “response”. If there is more than one most probable value, the first one is chosen.

“id.node” an (integer) “ID” of the current context (= node of the tree represented VLMC).

“depth” the context length, i.e., the depth of the Markov chain, at the current observation (of newdata).

“ALL” an object of class “predict.vlmc”, a list with the following components,  
**ID** integer vector as for type = “id.node”,  
**probs** prediction probability matrix, as above,  
**flags** integer vector, non-zero for particular states only, rather for debugging.  
**ctxt** character, ctxt[i] a string giving the context (backwards) for newdata[i], using alphabet letters.  
**fitted** character with fitted values, i.e., the alphabet letter with the highest probability, using max.col where ties are broken at random.  
**alpha, alpha.len** the alphabet (single string) and its length.  
 which has its own print method (print.predict.vlmc).

## Note

The predict method and its possible arguments may still be developed, and we are considering to return the marginal probabilities instead of NA for the first value(s).

The print method print.predict.vlmc uses fractions from package MASS to display the probabilities  $Pr[X = j]$ , for  $j \in \{0, 1, \dots\}$ , as these are rational numbers, shown as fractions of integers.

## See Also

vlmc and residuals.vlmc. For simulation, simulate.vlmc.

**Examples**

```

f1 <- c(1,0,0,0)
f2 <- rep(1:0,2)
(dt2 <- rep(c(f1,f1,f2,f1,f2,f2,f1),2))

(vlmc.dt2c15 <- vlmc(dt2, cutoff = 1.5))
draw(vlmc.dt2c15)

## Fitted Values:
all.equal(predict(vlmc.dt2c15, dt2), predict(vlmc.dt2c15))
(pa2c15 <- predict(vlmc.dt2c15, type = "ALL"))

## Depth = context length ([1] : NA) :
stopifnot(nchar(pa2c15 $ ctxt)[-1] ==
           predict(vlmc.dt2c15, type = "depth")[-1])

same <- (ff1 <- pa2c15 $ fitted) ==
         (ff2 <- int2alpha(predict(vlmc.dt2c15, type="response"), alpha="01"))
which(!same) #-> some are different, since max.col() breaks ties at random!

ndt2 <- c(rep(0,6),f1,f1,f2)
predict(vlmc.dt2c15, ndt2, "ALL")

(newdt2 <- sample(dt2, 17))
pm <- predict(vlmc.dt2c15, newdt2, allow.subset = TRUE)
summary(apply(pm, 1, sum))# all 1

predict(vlmc.dt2c15, newdt2, type = "ALL")

data(bnrf1)
(vbnrf <- vlmc(bnrf1EB))
(pA <- predict(vbnrf, bnrf1EB[1:24], type = "ALL"))
pc <- predict(vbnrf, bnrf1EB[1:24], type = "class")
pr <- predict(vbnrf, bnrf1EB[1:24], type = "resp")
stopifnot(as.integer (pc[-1]) == 1 + pr[-1],
           as.character(pc[-1]) == strsplit(vbnrf$alpha,NULL)[[1]][1 + pr[-1]])

##-- Example of a "perfect" fit -- just for illustration:
## the default, thresh = 2 doesn't fit perfectly(i=38)
(vlmc.dt2c0th1 <- vlmc(dt2, cutoff = 0, thresh = 1))

## "Fitted" = "Data" (but the first which can't be predicted):
stopifnot(dt2[-1] == predict(vlmc.dt2c0th1,type = "response")[-1])

```

---

prt.vvec

*Recursively Print the VLMC Result Vector*


---

**Description**

This is an auxiliary function which recursively displays (prints) the integer result vector of a `vlmc` fit.

**Usage**

```
prt.vvec(v, nalph, pad=" ")
```

**Arguments**

`v` typically `x $ vlmc.vec[-1]` where `x` is the result of `vlmc(*)`.  
`nalph` alphabet size; typically `x $ vlmc.vec[1]`.  
`pad` character, to be used for padding `paste(*, collapse=pad)`.

**See Also**

[summary.vlmc](#) which uses `prt.vvec`.

**Examples**

```
example(vlmc)
str(vv <- vlmc.dt1$vlmc)
prt.vvec(vv[-1], n = 2)
prt.vvec(vv[-1], n = 2, pad = " | ")
```

---

RCplot

*Residuals vs Context plot*


---

**Description**

Plots the residuals of a fitted VLMC model against the contexts, i.e., produces a boxplot of residuals for all contexts used in the model fit.

This has proven to be useful function, and the many optional arguments allow quite a bit of customization. However, the current implementation is somewhat experimental and the defaults have been chosen from only a few examples.

**Usage**

```
RCplot(x, r2 = residuals(x, "deviance")^2,
       alphabet = x$alpha, lab.horiz = k <= 20,
       do.call = TRUE,
       cex.axis = if (k <= 20) 1 else if (k <= 40) 0.8 else 0.6,
       y.fact = if (.Device == "postscript") 1.2 else 0.75,
       col = "gray70", xlab = "Context", main = NULL,
       med.pars = list(col = "red", pch = 12, cex = 1.25 * cex.axis),
       ylim = range(0, r2, finite = TRUE),
       ...)
```

**Arguments**

<code>x</code>	an R object of class <code>v1mc</code> .
<code>r2</code>	numeric vector, by default of squared deviance residuals of <code>x</code> , but conceptually any (typically non-negative) vector of the appropriate length.
<code>alphabet</code>	the alphabet to use for labeling the contexts, via <code>id2ctxt</code> .
<code>lab.horiz</code>	logical indicating if the context labels should be written <b>horizontally</b> or vertically.
<code>do.call</code>	logical indicating if the <code>v1mc</code> call should be put as subtitle.
<code>cex.axis</code>	the character expansion for axis labeling, see also <code>par</code> . The default is only approximately good.
<code>y.fact</code>	numeric factor for expanding the space to use for the context labels (when <code>lab.horiz</code> is false).
<code>col</code>	color used for filling the boxes.
<code>xlab</code>	x axis label (with default).
<code>main</code>	main title to be used, NULL entailing a sensible default.
<code>med.pars</code>	graphical parameters to be used for coding of medians that are almost 0.
<code>ylim</code>	y range limits for plotting.
<code>...</code>	further arguments to be passed to <code>plot()</code> .

**Value**

*Invisibly*, a list with components

<code>k</code>	the number of contexts (and hence box plots) used.
<code>fID</code>	a factor (as used in the internal call to <code>plot.factor</code> ).
<code>rp</code>	a list as resulting from the above call to <code>plot.factor()</code> .

**Author(s)**

Martin Maechler

**References**

Mächler M. and Bühlmann P. (2004) Variable Length Markov Chains: Methodology, Computing, and Software. *J. Computational and Graphical Statistics* **2**, 435–455.

**See Also**

[summary.v1mc](#) for other properties of a VLMC model.

**Examples**

```
example(v1mc)
RCplot(v1mc.pres)
RCplot(v1mc.dt1c01)## << almost perfect fit (0 resid.)
```



---

residuals.vlmc	<i>Compute Residuals of a Fitted VLMC Object</i>
----------------	--

---

### Description

Compute residuals of a fitted `vlmc` object.

This is yet a matter of research and may change in the future.

### Usage

```
## S3 method for class 'vlmc'
residuals(object,
  type = c("classwise",
           "deviance", "pearson", "working", "response", "partial"),
  y = object$y, ...)
```

### Arguments

object	typically the result of <code>vlmc(...)</code> .
type	The type of residuals to compute, defaults to "classwise" which returns an $n \times m$ matrix, see below. The other types only make sense when the discrete values of <code>y</code> are ordered which always includes the binary case ( $m = 2$ ). The "deviance" residuals $r$ are defined similarly as for logistic regression, see below. "pearson", "working" and "response" are currently identical and give the difference of the underlying integer code (of the discrete data). <b>Note</b> that "partial" residuals are not yet defined!
y	discrete time series with respect to which the residuals are to be computed.
...	possibly further arguments (none at the moment).

### Value

If `type = "classwise"` (the default), a numeric matrix of dimension  $n \times m$  of values  $I_{i,j} - p_{i,j}$  where the indicator  $I_{i,j}$  is 1 iff `y[i] == a[j]` and `a` is the alphabet (or levels) of `y`, and  $p_{i,j}$  are the elements of the estimated (1-step ahead) predicted probabilities, `p <- predict(object)`. Hence, for each  $i$ , the only positive residual stands for the observed class.

For all other types, the result is a numeric vector of the length of the original time-series (with first element NA).

For `type = "deviance"`,  $r_i = \pm \sqrt{-2 \log(P_i)}$  where  $P_i$  is the predicted probability for the  $i$ -th observation which is the same as  $p_{i,y_i}$  above (now assuming  $y_i \in \{1, 2, \dots, m\}$ ). The sum of the squared deviance residuals is the deviance of the fitted model.

### Author(s)

Martin Maechler

**See Also**

[vlmc](#), [deviance.vlmc](#), and [RCplot](#) for a novel residual plot.

**Examples**

```
example(vlmc)
rp <- residuals(vlmc.pres)
stopifnot(all(abs(apply(rp[-1,],1,sum)) < 1e-15))
matplot(seq(presidents), rp, ylab = "residuals", type="l")
## ``Tukey-Anscombe'' (the following is first stab at plot method):
matplot(fitted(vlmc.pres), rp, ylab = "residuals", xaxt = "n",
        type="b", pch=vlmc.pres$alpha)
axis(1, at = 0:(vlmc.pres$alpha.len-1),
     labels = strsplit(vlmc.pres$alpha,"")[[1]])

summary(rd <- residuals(vlmc.pres, type = "dev"))
rd[1:7]
## sum of squared dev.residuals === deviance :
all.equal(sum(rd[-1] ^ 2),
          deviance(vlmc.pres))
```

---

simulate.vlmc

*Simulate a Discrete Time Series from fitted VLMC model*

---

**Description**

Simulate from fitted VLMC model – basis of the VLMC bootstrap

**Usage**

```
## S3 method for class 'vlmc'
simulate(object, nsim = 1, seed = NULL, n,
        n.start = 64 * object$size[["context"]],
        integer.return = FALSE, keep.RSeed = TRUE, ...)
```

**Arguments**

object	typically the result of <code>vlmc(...)</code> .
nsim, n	non-negative integer, giving the length of the result. Note that n is deprecated and just there for back compatibility.
seed	random seed initializer; see <a href="#">simulate</a> .
n.start	the number of initial values to be discarded (because of initial effects).
integer.return	logical; if TRUE, the result will be an <a href="#">integer</a> vector with values in $0:(k-1)$ ; otherwise the resulting vector consists of letters from the alphabet $x$alpha$ .
keep.RSeed	logical indicating if the seed should be stored with the result (as ‘required’ by the generic <a href="#">simulate</a> ). Only set this FALSE with good reasons (back compatibility).
...	(potentially further arguments for other simulate methods.

**Details**

The `.Random.seed` is used and updated as with other random number generation routines such as `rbinom`.

Note that if you want to simulate from a given start sequence `x0`, you'd use `predict.vlmc(x, x0, type= "response")` — actually not quite yet.

**Value**

A "simulate.vlmc" object, basically a vector of length `nsim`. Either `integer` or `character`, depending on the `integer.return` argument, see above. Further, if `keep.RSeed` was true (as by default), a "seed" attribute with the random seed at the start of the simulation, for reproducibility.

**Author(s)**

Martin Maechler

**See Also**

`vlmc` and `predict.vlmc`.

**Examples**

```
example(vlmc)

simulate(vlmc.dt1, 100)
simulate(vlmc.dt1c01, 100, int = TRUE)
# n.start = 0: 1st few observations will resemble the data
simulate(vlmc.dt1c01, 20, n.start=0, int = TRUE)
```

---

summary.vlmc

---

*Summary of Fitted Variable Length Markov Chain (VLMC)*


---

**Description**

Compute (and print) a summary of a `vlmc` object which is typically the result of `vlmc(. .)`.

**Usage**

```
## S3 method for class 'vlmc'
summary(object, ...)
## S3 method for class 'summary.vlmc'
print(x, digits = getOption("digits"),
      vvec.printing = FALSE, ...)
```

**Arguments**

<code>object</code>	an R object of class <code>vlmc</code> .
<code>x</code>	an R object of class <code>summary.vlmc</code> .
<code>digits</code>	integer giving the number of significant digits for printing numbers.
<code>vvec.printing</code>	logical indicating if the <code>vvec</code> component should be printed recursively via <code>pvt.vvec()</code> .
<code>...</code>	potentially further arguments [Generic].

**Value**

`summary.vlmc()` returns an object of class "summary.vlmc" for which there's a print method. It is basically a list containing all of `object`, plus additionally

<code>confusion.table</code>	the symmetric contingency table of data vs fitted.
<code>depth.stats</code>	statistics of Markov chain depth along the data; currently just <code>summary(predict(object, type="depth"))</code> .
<code>R2</code>	the $R^2$ statistic, i.e. the percentage (in [0,1]) of correctly predicted data.

**See Also**

[vlmc](#), [draw.vlmc](#).

**Examples**

```
data(bnrf1)
vb <- vlmc(bnrf1EB)
svb <- summary(vb)
svb
```

---

 vlmc

---

*Fit a Variable Length Markov Chain (VLMC)*


---

**Description**

Fit a Variable Length Markov Chain (VLMC) to a discrete time series, in basically two steps: First a large Markov Chain is generated containing (all if `threshold.gen = 1`) the context states of the time series. In the second step, many states of the MC are collapsed by *pruning* the corresponding context tree.

Currently, the "alphabet" may contain can at most 26 different "character"s.

**Usage**

```
vlmc(dts,
     cutoff.prune = qchisq(alpha.c, df=max(.1,alpha.len-1),lower.tail=FALSE)/2,
     alpha.c = 0.05,
     threshold.gen = 2,
     code1char = TRUE, y = TRUE, debug = FALSE, quiet = FALSE,
     dump = 0, ctl.dump = c(width.ct = 1+log10(n), nmax.set = -1) )

is.vlmc(x)
## S3 method for class 'vlmc'
print(x, digits = max(3, getOption("digits") - 3), ...)
```

**Arguments**

dts	a discrete “time series”; can be a numeric, character or factor.
cutoff.prune	non-negative number; the cutoff used for pruning; defaults to half the $\alpha$ -quantile of a chisq distribution, where $\alpha = \text{alpha.c}$ , the following argument:
alpha.c	number in (0,1) used to specify cutoff.prune in the more intuitive $\chi^2$ quantile scale; defaulting to 5%.
threshold.gen	integer $\geq 1$ (usually left at 2). When <i>generating</i> the initial large tree, only generate nodes with count $\geq$ threshold.gen.
code1char	logical; if true (default), the data dts will be .....FIXME.....
y	logical; if true (default), the data dts will be returned. This allows to ensure that residuals ( <a href="#">residuals.vlmc</a> ) and “k-step ahead” predictions can be computed from the result.
debug	logical; should debugging info be printed to stderr.
quiet	logical; if true, don’t print some warnings.
dump	integer in 0:2. If positive, the pruned tree is dumped to stderr; if 2, the initial <b>un</b> pruned tree is dumped as well.
ctl.dump	integer of length 2, say <code>ctl[1:2]</code> controlling the above dump when <code>dump &gt; 0</code> . <code>ctl[1]</code> is the width (number of characters) for the “counts”, <code>ctl[2]</code> the maximal number of set elements that are printed per node; when the latter is not positive (by default), currently <code>max(6, 15 - log10(n))</code> is used.
x	a fitted “vlmc” object.
digits	integer giving the number of significant digits for printing numbers.
...	potentially further arguments [Generic].

**Value**

A “vlmc” object, basically a list with components

nobs	length of data series when fit. (was named “n” in earlier versions.)
threshold.gen, cutoff.prune	the arguments (or their defaults).
alpha.len	the alphabet size.

alpha	the alphabet used, as one string.
size	a named integer vector of length ( $\geq$ ) 4, giving characteristic sizes of the fitted VLMC. Its named components are <b>"ord.MC"</b> the (maximal) order of the Markov chain, <b>"context"</b> the “context tree size”, i.e., the number of leaves plus number of “hidden nodes”, <b>"nr.leaves"</b> is the number of leaves, and <b>"total"</b> the number of integers needed to encode the VLMC tree, i.e., <code>length(vlmc.vec)</code> (see below).
vlmc.vec	integer vector, containing (an encoding of) the fitted VLMC tree.
y	if <code>y = TRUE</code> , the data <code>dts</code> , as <a href="#">character</a> , using the letters from alpha.
call	the <code>call vlmc(..)</code> used.

**Note**

Set `cutoff = 0`, `thresh = 1` for getting a “perfect fit”, i.e. a VLMC which perfectly re-predicts the data (apart from the first observation). Note that even with `cutoff = 0` some pruning may happen, for all (terminal) nodes with  $\delta=0$ .

**Author(s)**

Martin Maechler

**References**

- Buhlmann P. and Wyner A. (1998) Variable Length Markov Chains. *Annals of Statistics* **27**, 480–513.
- Mächler M. and Bühlmann P. (2004) Variable Length Markov Chains: Methodology, Computing, and Software. *J. Computational and Graphical Statistics* **2**, 435–455.
- Mächler M. (2004) VLMC — Implementation and R interface; working paper.

**See Also**

[draw.vlmc](#), [entropy](#), [simulate.vlmc](#) for “VLMC bootstrapping”.

**Examples**

```
f1 <- c(1,0,0,0)
f2 <- rep(1:0,2)
(dt1 <- c(f1,f1,f2,f1,f2,f2,f1))

(vlmc.dt1 <- vlmc(dt1))
vlmc(dt1, dump = 1,
      ctl.dump = c(wid = 3, nmax = 20), debug = TRUE)
(vlmc.dt1c01 <- vlmc(dts = dt1, cutoff.prune = .1, dump=1))

data(presidents)
dpres <- cut(presidents, c(0,45,70, 100)) # three values + NA
```

```

table(dpres <- factor(dpres, exclude = NULL)) # NA as 4th level
levels(dpres)#-> make the alphabet -> warning
vlmc.pres <- vlmc(dpres, debug = TRUE)
vlmc.pres

## alphabet & and its length:
vlmc.pres$alpha
stopifnot(
  length(print(strsplit(vlmc.pres$alpha,NULL)[[1]])) == vlmc.pres$ alpha.len
)

## You now can use larger alphabets (up to 95) letters:
set.seed(7); it <- sample(40, 20000, replace=TRUE)
v40 <- vlmc(it)
v40
## even larger alphabets now give an error:
il <- sample(100, 10000, replace=TRUE)
ee <- tryCatch(vlmc(il), error= function(e)e)
stopifnot(is(ee, "error"))

```

---

vlmc.version	<i>Version of VLMC Package</i>
--------------	--------------------------------

---

### Description

Character string, giving the version number (and date) of the VLMC package.

### Examples

```

vlmc.version
## Not run:
[1] "VLMC 1.3-14; after $Date: 2014/06/03 08:05:21 $ UTC"

## End(Not run)

```

---

vlmctree	<i>Compute the tree structure of a "vlmc" object</i>
----------	--

---

### Description

Compute the tree representation of a "[vlmc](#)" object as R [list](#).

### Usage

```

vlmctree(x)

## S3 method for class 'vtree'
str(object, ...)
.vvec2tree(vv, k, chk.lev)

```

**Arguments**

x, object	typically the result of <code>vlmc(. .)</code> .
vv	integer vector encoding the fitted vlmc, typically <code>x\$vlmc.vec[-1]</code> .
k	integer, the alphabet size.
chk.lev	integer internally used for consistency checking.
...	further arguments passed to or from methods.

**Details**

`.vvec2tree` is the internal (recursive) function building up the tree.

`str.vtree` is a method for the generic `str` function and typically for the output of `vlmctree()`. For each node, it gives the “parenting level” in braces and the counts.

**Value**

A list of class “vtree” representing the tree structure recursively.

Each “node” of the tree is itself a list with components

level	length-2 integer giving the level in $\{0, 1, \dots\}$ , counted from the root (which is 0) and the parenting level, i.e the longest branch.
count	integer vector of length k where k is the number of “letters” in the alphabet.
total	equals to <code>sum(* \$ count)</code> .
child	a list (of length k) of child nodes or <code>NULL</code> (i.e. not there).

**Author(s)**

Martin Maechler

**See Also**

[vlmc](#).

**Examples**

```
data(presidents)
dpres <- cut(presidents, c(0,45,70, 100)) # three values + NA
table(dpres <- factor(dpres, exclude = NULL)) # NA as 4th level

(vlmc.prc1 <- vlmc(dpres, cut = 1, debug = TRUE))
str(vv.prc1 <- vlmctree(vlmc.prc1))
```



# Index

- \* **character**
  - alpha2int, 2
  - alphabet, 3
  - int2char, 8
- \* **datasets**
  - bnrf1, 4
  - OZrain, 11
- \* **data**
  - vlmc.version, 23
- \* **graphs**
  - as.dendrogram.vlmc, 3
- \* **hplot**
  - RCplot, 15
- \* **iplot**
  - as.dendrogram.vlmc, 3
- \* **models**
  - deviance.vlmc, 5
  - draw.vlmc, 6
  - logLik, 9
  - predict.vlmc, 12
  - residuals.vlmc, 17
  - simulate.vlmc, 18
  - summary.vlmc, 19
  - vlmc, 20
  - vlmctree, 23
- \* **ts**
  - deviance.vlmc, 5
  - draw.vlmc, 6
  - logLik, 9
  - predict.vlmc, 12
  - residuals.vlmc, 17
  - simulate.vlmc, 18
  - summary.vlmc, 19
  - vlmc, 20
  - vlmctree, 23
- \* **utilities**
  - alpha2int, 2
  - alphabet, 3
  - id2ctxt, 8
  - int2char, 8
  - pvt.vvec, 14
  - RCplot, 15
  - .Random.seed, 19
  - .vvec2tree (vlmctree), 23
- AIC, 10
- alpha2int, 2, 3
- alphabet, 2, 3
- as.dendrogram, 4
- as.dendrogram.vlmc, 3
- BIC, 10
- bnrf1, 4
- bnrf1EB (bnrf1), 4
- bnrf1ebv, 5
- bnrf1HV (bnrf1), 4
- call, 22
- char2int, 2
- char2int (int2char), 8
- character, 2, 3, 19, 22
- class, 24
- dendrogram, 4
- deviance.vlmc, 5, 10, 18
- dimnames, 13
- draw (draw.vlmc), 6
- draw.vlmc, 6, 10, 20, 22
- entropy, 6, 22
- entropy (logLik), 9
- entropy2 (logLik), 9
- factor, 5
- fitted.vlmc (predict.vlmc), 12
- fractions, 13
- id2ctxt, 8, 16
- int2alpha, 9
- int2alpha (alpha2int), 2

int2char, [2](#), [8](#)  
integer, [2](#), [18](#), [19](#)  
is.vlmc (vlmc), [20](#)

length, [5](#)  
list, [23](#), [24](#)  
logLik, [9](#), [9](#)  
logLik.vlmc (logLik), [9](#)

max.col, [13](#)

NA, [13](#)  
nchar, [2](#), [8](#)  
NULL, [24](#)

OZrain, [11](#)

par, [16](#)  
plot.dendrogram, [4](#)  
plot.factor, [16](#)  
predict.vlmc, [8](#), [12](#), [19](#)  
print, [13](#)  
print.predict.vlmc (predict.vlmc), [12](#)  
print.summary.vlmc (summary.vlmc), [19](#)  
print.vlmc (vlmc), [20](#)  
prt.vvec, [14](#), [20](#)

rbinom, [19](#)  
RCplot, [15](#), [18](#)  
residuals.vlmc, [6](#), [13](#), [17](#), [21](#)

simulate, [18](#)  
simulate.vlmc, [13](#), [18](#), [22](#)  
str, [24](#)  
str.vtree (vlmctree), [23](#)  
summary.vlmc, [15](#), [16](#), [19](#)

ts, [11](#)

vlmc, [2-4](#), [6-10](#), [12-20](#), [20](#), [23](#), [24](#)  
vlmc.version, [23](#)  
vlmctree, [23](#)