Package ‘NNLM’

June 18, 2018

Type Package

Title Fast and Versatile Non-Negative Matrix Factorization

Description This is a package for Non-Negative Linear Models (NNLM). It implements fast sequential coordinate descent algorithms for non-negative linear regression and non-negative matrix factorization (NMF). It supports mean square error and Kullback-Leibler divergence loss. Many other features are also implemented, including missing value imputation, domain knowledge integration, designable W and H matrices and multiple forms of regularizations.

Version 0.4.2

Date 2018-05-17

Depends R (>= 3.0.1)

Imports Rcpp (>= 0.11.0), stats, utils

LinkingTo Rcpp, RcppArmadillo, RcppProgress

LazyData yes

LazyLoad yes

NeedsCompilation yes

Suggests testthat, knitr, rmarkdown, mice, missForest, ISOpureR

VignetteBuilder knitr

RoxygenNote 5.0.0

License BSD_2_clause + file LICENSE

BugReports https://github.com/linxihui/NNLM/issues

URL https://github.com/linxihui/NNLM

Author Xihui Lin [aut, cre],
Paul C Boutros [aut]

Maintainer Xihui Lin <ericxihuilin@gmail.com>

Repository CRAN

Date/Publication 2018-06-18 18:24:15 UTC
\textbf{R topics documented:}

\begin{itemize}
  \item mse.mkl \hspace{1cm} 2
  \item nlm \hspace{1cm} 2
  \item nnmf \hspace{1cm} 4
  \item nsclc \hspace{1cm} 8
  \item predict.nnmf \hspace{1cm} 8
\end{itemize}

\section*{mse.mkl}

\textit{Compute mean square error (MSE) and mean kL divergence (MKL)}

\subsection*{Description}

Compute mean square error (MSE) and mean kL divergence (MKL)

\subsection*{Usage}

\texttt{mse.mkl(\texttt{obs}, \texttt{pred}, \texttt{na.rm = TRUE}, \texttt{show.warning = TRUE})}

\subsection*{Arguments}

- \texttt{obs} observed value
- \texttt{pred} prediction/estimate
- \texttt{na.rm} if to remove NAs
- \texttt{show.warning} if to show warning if any

\subsection*{Value}

A vector of c(MSE, MKL)

\section*{nnlm}

\textit{Non-negative linear model/regression (NNLM)}

\subsection*{Description}

Solving non-negative linear regression problem as

\[
\arg\min_{\beta \geq 0} L(y - x^T \beta) + \alpha_1 ||\beta||_2^2 + \alpha_2 \sum_{i < j} \beta_i^T \beta_j^T + \alpha_3 ||\beta||_1
\]

where \( L \) is a loss function of either square error or Kullback-Leibler divergence.
Usage

```r
nnlm(x, y, alpha = rep(0, 3), method = c("scd", "lee"), loss = c("mse", "mkl"),
    init = NULL, mask = NULL, check.x = TRUE, max.iter = 10000L,
    rel.tol = 1e-12, n.threads = 1L, show.warning = TRUE)
```

Arguments

- **x**: Design matrix
- **y**: Vector or matrix of response
- **alpha**: A vector of non-negative value length equal to or less than 3, meaning \([L2, \text{angle}, L1]\) regularization on \(\beta\) (non-masked entries)
- **method**: Iteration algorithm, either 'scd' for sequential coordinate-wise descent or 'lee' for Lee's multiplicative algorithm
- **loss**: Loss function to use, either 'mse' for mean square error or 'mkl' for mean KL-divergence. Note that if \(x, y\) contains negative values, one should always use 'mse'
- **init**: Initial value of \(\beta\) for iteration. Either NULL (default) or a non-negative matrix of
- **mask**: Either NULL (default) or a logical matrix of the same shape as \(\beta\), indicating if an entry should be fixed to its initial (if init specified) or 0 (if init not specified).
- **check.x**: If to check the condition number of \(x\) to ensure unique solution. Default to TRUE but could be slow
- **max.iter**: Maximum number of iterations
- **rel.tol**: Stop criterion, relative change on \(x\) between two successive iteration. It is equal to \(2 \times |e_2 - e_1|/(e_2 + e_1)\). One could specify a negative number to force an exact max.iter iteration, i.e., not early stop
- **n.threads**: An integer number of threads/CPUs to use. Default to 1 (no parallel). Use 0 or a negative value for all cores
- **show.warning**: If to shown warnings if exists. Default to TRUE

Details

The linear model is solve in column-by-column manner, which is paralleled. When \(y_j\) (j-th column) contains missing values, only the complete entries are used to solve \(\beta_j\). Therefore, the minimum complete entries of each column should be not smaller than number of columns of \(x\) when penalty is not used.

**method = 'scd'** is recommended, especially when the solution is probably sparse. Though both "mse" and "mkl" loss are supported for non-negative \(x\) and \(y\), only "mse" is proper when either \(y\) or \(x\) contains negative value. Note that loss "mkl" is much slower then loss "mse", which might be your concern when \(x\) and \(y\) is extremely large.

**mask** is can be used for hard regularization, i.e., forcing entries to their initial values (if init specified) or 0 (if init not specified). Internally, mask is achieved by skipping the masked entries during the element-wise iteration.
Value

An object of class ‘nnlm’, which is a list with components

- coefficients : a matrix or vector (depend on y) of the NNLM solution, i.e., \( \beta \)
- n.iteration : total number of iteration (sum over all column of beta)
- error : a vector of errors/loss as c(MSE, MKL, target.error) of the solution
- options : list of information of input arguments
- call : function call

Author(s)

Eric Xihui Lin, <xihui1.silence@gmail.com>

References


Examples

```r
# without negative value
x <- matrix(runif(50*20), 50, 20);
beta <- matrix(rexp(20*2), 20, 2);
y <- x %*% beta + 0.1*matrix(runif(50*2), 50, 2);
beta.hat <- nnlm(x, y, loss = 'mkl');

# with negative values
x2 <- 10*matrix(rnorm(50*20), 50, 20);
y2 <- x2 %*% beta + 0.2*matrix(rnorm(50*2), 50, 2);
beta.hat2 <- nnlm(x, y);
```

### Description

Non-negative matrix factorization (NMF or NNMF) using sequential coordinate-wise descent or multiplicative updates.
Usage

nnmf(A, k = 1L, alpha = rep(0, 3), beta = rep(0, 3), method = c("scd", "lee"), loss = c("mse", "mkl"), init = NULL, mask = NULL, 
W.norm = -1L, check.k = TRUE, max.iter = 500L, rel.tol = 1e-04, 
n.threads = 1L, trace = 100/inner.max.iter, verbose = 1L, 
show.warning = TRUE, inner.max.iter = ifelse("mse" == loss, 50L, 1L), 
inner.rel.tol = 1e-09)

Arguments

A
An integer of decomposition rank

k
A matrix to be decomposed

alpha
[L2, angle, L1] regularization on W (non-masked entries)

beta
[L2, angle, L1] regularization on H (non-masked entries)

method
Decomposition algorithms, either 'scd' for sequential coordinate-wise descent(default) or 'lee' for Lee's multiplicative algorithm

loss
Loss function to use, either 'mse' for mean square error (default) or 'mkl' for mean KL-divergence

init
A list of initial matrices for W and H. One can also supply known matrices W0, H0, and initialize their correspondent matrices H1 and W1. See details

mask
A list of mask matrices for W, H, H1 (if init$w supplied), W1 (if init$h supplied), which should have the same shapes as W, H, H1 and W1 if specified. If initial matrices not specified, masked entries are fixed to 0. See details

W.norm
A numeric value 'p' indicating the $L_p$-norm (can be infinity) used to normalized the outcome W matrix. No normalization will be performed if 0 or negative. This argument has no effect on outcome correspondent to known profiles W0, H0. Default to 1, i.e. sum of W should sum up to 1, which can be interpreted as "distribution" or "proportion"

check.k
If to check whether $k \leq nm/(n+m)$, where (n,m)=dim(A), or k is smaller than the column-wise and row-wise minimum numbers of complete observation

max.iter
Maximum iteration of alternating NNLS solutions to H and W

rel.tol
Stop criterion, which is relative tolerance between two successive iterations, = le2-e1/avg(e1, e2)

n.threads
An integer number of threads/CPUs to use. Default to 1(no parallel). Specify 0 for all cores

trace
An integer indicating how frequent the error should be checked. MSE and MKL error will computed every trace iterations. If 0 or negative, trace is set to a very large number and only errors of the first and last iterations are checked.

verbose
Either 0/FALSE, 1/TRUE or 2, with 0/FALSE/else = no any tracking, 1 == progression bar, 2 == print iteration info.

show.warning
If to show warnings when targeted rel.tol is not reached

inner.max.iter
Maximum number of iterations passed to each inner W or H matrix updating loop

inner.rel.tol
Stop criterion for the inner loop, which is relative tolerance passed to inner W or H matrix updating i.e., le2-e1/avg(e1, e2)
Details

The problem of non-negative matrix factorization is to find \( W, H, W_1, H_1 \), such that

\[
A = WH + W_0H_1 + W_1H_0 + \varepsilon = [WW_0W_1][H'H_1H_0]' + \varepsilon
\]

where \( W_0, H_0 \) are known matrices, which are NULLs in most application case and \( \varepsilon \) is noise. In tumour content deconvolution, \( W_0 \) can be thought as known healthy profile, and \( W \) is desired pure cancer profile. One also set \( H_0 \) to a row matrix of 1, and thus \( W_1 \) can be treated as common profile among samples. Use init to specify \( W_0 \) and \( H_0 \).

Argument init, if used, must be a list with entries named as 'W', 'H', 'W0', 'W1', 'H1', 'H0'. One could specify only a few of them. Only use 'W0' (and its correspondent 'H1') or 'H0' (and its correspondent 'W1') for known matrices/profiles.

Similarly, argument mask, if used, must be a list entries named as 'W', 'H', 'W0', 'W1', 'H1', 'H0', and they should be either NULL (no specified) or a logical matrix. If a masked for matrix is specified, then masked entries will be fixed to their initial values if initialized (skipped during iteration), or 0 if not initialized.

To simplify the notations, we denote right hand side of the above equation as \( WH \). The problem to solved using square error is

\[
\arg\min_{W \geq 0, H \geq 0} L(A, WH) + J(W, \alpha) + J(H', \beta)
\]

where \( L(x, y) \) is a loss function either a square loss

\[
\frac{1}{2}||x - y||_2^2
\]

or a Kullback-Leibler divergence

\[
x \log(x/y) - x - y,
\]

and

\[
J(X, \alpha) = \alpha_1J_1(X) + \alpha_2J_2(X) + \alpha_3J_3(X),
\]

\[
J_1(X) = \frac{1}{2}||X||_F^2 = \frac{1}{2}tr(XXT),
\]

\[
J_2(X) = \sum_{i<j}(X_{i,j})^2X_{i,j} = \frac{1}{2}tr(X(E-I)XT),
\]

\[
J_3(X) = \sum_{i,j} |x_{i,j}| = tr(XE).
\]

The formal one is usually better for symmetric distribution, while the later one is more suitable for skewed distribution, especially for count data as it can be derived from Poisson distributed observation. The penalty function \( J \) is a composition of three types of penalties, which aim to minimizing L2 norm, maximizing angles between hidden features (columns of \( W \) and rows of \( H \)) and L1 norm (sparsity). The parameters \( \alpha, \beta \) of length 3 indicates the amount of penalties.

When method == 'scd', a sequential coordinate-wise descent algorithm is used when solving \( W \) and \( H \) alternatively, which are non-negative regression problem. The inner.max.iter and
inner.rel.tol is used to control the number of iteration for these non-negative regressions. This is also applicable to method == 'lee' (the original algorithm only iteration through all entries once for each iteration), which is usually faster than the original algorithm when loss == 'mse'. When loss == 'mkl', a quadratic approximation to the KL-divergence is used when method == 'scd'. Generally, for run time, 'scd' is faster than 'lee' and 'mse' is faster than 'mkl'.

Value

A list with components

- W : left matrix, including known W0 and W1 if available, i.e., column stacked as [W, W0, W1]
- H : right matrix, including H1 and known H0 if available, i.e. row stacked as [H', H1', H0']
- mse : a vector of mean squared errors through iterations
- mkl : a vector of mean KL-divergence through iterations
- target.loss : target for minimization, which is mean KL-divergence (if loss == 'mkl') or half of mean squared error if loss == 'mse' plus penalties
- average.epochs : a vector of average epochs (one complete swap over W and H)
- n.iteration : total number of iteration (sum over all column of beta)
- run.time : running time
- options : list of information of input arguments
- call : function call

Author(s)

Eric Xihui Lin, <xihuil.silence@gmail.com>

References


See Also

nnlm, predict.nnmf
Examples

# Pattern extraction, meta-gene
set.seed(123);

data(nsclc, package = 'NNLM')
str(nsclc)

decomp <- nnmf(nsclc[, 1:80], 3, rel.tol = 1e-5);

heatmap(decomp$W, Colv = NA, xlab = 'Gene', margins = c(2,2),
labRow = '', labCol = '', scale = 'column', col = cm.colors(100));
heatmap(decomp$H, Rowv = NA, ylab = 'Gene', xlab = 'Patient', margins = c(2,2),
labRow = '', labCol = '', scale = 'row', col = cm.colors(100));

# missing value imputation
set.seed(123);
nsclc2 <- nsclc;
index <- sample(length(nsclc2), length(nsclc2)*0.3);
nsclc2[index] <- NA;

# impute using NMF
system.time(nsclc2.nmf <- nnmf(nsclc2, 2));
nsclc2.hat.nmf <- with(nsclc2.nmf, W %*% H);
mse.mkl(nsclc[index], nsclc2.hat.nmf[index])

---

nsclc  
**Micro-array data of NSCLC patients**

---

Description

This dataset is a random subset (matrix) of micro-array data from a group of Non-Small Cell Lung Cancer (NSCLC) patients. It contains 200 probes / genes (row) for 100 patients / samples (column).

---

predict.nnmf  
**Methods for nnmf object returned by nnmf**

---

Description

Methods for nnmf object returned by nnmf
Usage

```r
## S3 method for class 'nnmf'
predict(object, newdata, which = c("A", "W", "H"),
         method = object$options$method, loss = object$options$loss, ...)

## S3 method for class 'nnmf'
print(x, ...)
```

Arguments

- `object`: An NNMF object returned by `nnmf`
- `newdata`: A new matrix of x. No required when `which == 'A'`
- `which`: Either 'A' (default), 'W' or 'H'
- `method`: Either 'scd' or 'lee'. Default to `object$options$method`
- `loss`: Either 'mse' or 'mkl'. Default to `object$options$loss`
- `...`: Further arguments passed to 'nnlm' or 'print'
- `x`: An NNMF object returned by `nnmf`

Value

'`A'` or a class of 'nnlm' for 'predict.nnmf' and no return for 'print'.

See Also

- `nnmf`, `nnlm`

Examples

```r
x <- matrix(runif(50*20), 50, 20)
r <- nnmf(x, 2)
r
newx <- matrix(runif(50*30), 50, 30)
pred <- predict(r, newx, 'H')
```
Index

*Topic data
  nsclc, 8

mse.mkl, 2

nnlm, 2, 7, 9
nnmf, 4, 9
nsclc, 8

predict.nnmf, 7, 8
print.nnmf (predict.nnmf), 8