Package ‘rsparse’

April 14, 2019

Type Package

Title Statistical Learning on Sparse Matrices

Version 0.3.3.1

Maintainer Dmitriy Selivanov <selivanov.dmitriy@gmail.com>

Description Implements many algorithms for statistical learning on sparse matrices - matrix factorizations, matrix completion, elastic net regressions, factorization machines.
Also 'rsparse' enhances 'Matrix' package by providing methods for multithreaded <sparse, dense> matrix products and native slicing of the sparse matrices in Compressed Sparse Row (CSR) format.
List of the algorithms for regression problems:
1) Elastic Net regression via Follow The Proximally-Regularized Leader (FTRL) Stochastic Gradient Descent (SGD), as per McMahan et al(, <doi:10.1145/2487575.2488200>)
2) Factorization Machines via SGD, as per Rendle (2010, <doi:10.1109/ICDM.2010.127>)
List of algorithms for matrix factorization and matrix completion:
1) Weighted Regularized Matrix Factorization (WRMF) via Alternating Least Squares (ALS) - paper by Hu, Koren, Volinsky (2008, <doi:10.1109/ICDM.2008.22>)
2) Maximum-Margin Matrix Factorization via ALS, paper by Rennie, Srebro (2005, <doi:10.1145/1102351.1102441>)
3) Fast Truncated Singular Value Decomposition (SVD), Soft-Thresholded SVD, Soft-Impute matrix completion via ALS - paper by Hastie, Mazumder et al. (2014, <arXiv:1410.2596>)
4) Linear-Flow matrix factorization, from 'Practical linear models for large-scale one-class collaborative filtering' by Sedhain, Bui, Kawale et al (2016, ISBN:978-1-57735-770-4)
5) GlobalVectors (GloVe) matrix factorization via SGD, paper by Pennington, Socher, Manning (2014, <https://www.aclweb.org/anthology/D14-1162>)
Package is reasonably fast and memory efficient - it allows to work with large datasets - millions of rows and millions of columns. This is particularly useful for practitioners working on recommender systems.

License GPL (>= 2)

Encoding UTF-8

LazyData true

ByteCompile true

Depends R (>= 3.1.0), methods

Imports Matrix (>= 1.2), Rcpp (>= 0.11), mlapi (>= 0.1.0), data.table (>= 1.10.0), float (>= 0.2-2), RhpcBLASctl, lgr (>= 0.2)

LinkingTo Rcpp, RcppArmadillo (>= 0.9.100.5.0)

Suggests testthat, covr

URL https://github.com/dselivanov/rsparse

BugReports https://github.com/dselivanov/rsparse/issues

RoxygenNote 6.1.0

NeedsCompilation yes

Author Dmitriy Selivanov [aut, cre, cph] (<https://orcid.org/0000-0001-5413-1506>), Drew Schmidt [ctb] (configure script for BLAS, LAPACK detection), Wei-Chen Chen [ctb] (configure script and work on linking to float package)

Repository CRAN

Date/Publication 2019-04-14 20:13:09 UTC

R topics documented:

`detect_number_omp_threads`

Detects number of OpenMP threads in the system

Description

Detects number of OpenMP threads in the system respecting environment variables such as `OMP_NUM_THREADS` and `OMP_THREAD_LIMIT`

Usage

```
detect_number_omp_threads()
```

FactorizationMachine *Creates FactorizationMachine model.*

Description

Creates second order Factorization Machines model

Usage

```
FactorizationMachine
```

Format

`R6Class` object.

Usage

For usage details see **Methods, Arguments and Examples** sections.

```
fm = FM$new(learning_rate_w = 0.2, rank = 4, lambda_w = 0, lambda_v = 0, family = c("binomial", "gaussi
  intercept = TRUE, learning_rate_v = learning_rate_w)
fm$partial_fit(x, y, ...)
fm$predict(x, ...)
```

Methods

`FM$new(learning_rate_w = 0.2, rank = 4, lambda_w = 0, lambda_v = 0, family = c("binomial", "gaussia`
Constructor for FactorizationMachines model. For description of arguments see **Arguments** section.

`$partial_fit(x, y, ...)` fits/updates model given input matrix `x` and target vector `y`. `x` shape = (n_samples, n_features)

`$predict(x, ...)` predicts output `x`

Arguments

fm `FM` object

x Input sparse matrix - native format is `Matrix::RsparseMatrix`. If `x` is in different format, model will try to convert it to `RsparseMatrix` with `as(x, "RsparseMatrix")` call

learning_rate_w learning rate for linear weights in AdaGrad SGD

learning_rate_v learning rate for interactions in AdaGrad SGD

rank rank of the latent dimension in factorization

lambda_w regularization parameter for linear terms

lambda_v regularization parameter for interactions terms

intercept logical flag which specify whether to allow model to have non-zero intercept/bias

family a description of the error distribution and link function to be used in the model. Can be "gaussian" (for regression) or "binomial" (for classification)

Examples

```
# Factorization Machines can fit XOR function!
x = rbind(
  c(0, 0),
  c(0, 1),
  c(1, 0),
  c(1, 1)
)
y = c(0, 1, 1, 0)

x = as(x, "RsparseMatrix")
fm = FactorizationMachine$new(learning_rate_w = 10, rank = 2, lambda_w = 0,
  lambda_v = 0, family = 'binomial', intercept = TRUE)
res = fm$fit(x, y, n_iter = 100)
preds = fm$predict(x)
all(preds[c(1, 4)] < 0.01)
all(preds[c(2, 3)] > 0.99)
```

FTRL *Creates FTRL proximal logistic regression model.*

Description

Creates 'Follow the Regularized Leader' model. Only logistic regression implemented at the moment.

Usage

```
FTRL
```

Format

`R6Class` object.

Fields

`verbose logical = TRUE` whether to display training inforamtion

Usage

For usage details see **Methods, Arguments and Examples** sections.

```
ftrl = FTRL$new(learning_rate = 0.1, learning_rate_decay = 0.5,
lambda = 0, l1_ratio = 1, dropout = 0, family = "binomial")
ftrl$partial_fit(x, y, ...)
ftrl$predict(x, ...)
ftrl$coef()
```

Methods

`FTRL$new(learning_rate = 0.1, learning_rate_decay = 0.5, lambda = 0, l1_ratio = 1, dropout = 0, fam`
: Constructor for FTRL model. For description of arguments see **Arguments** section.

`$partial_fit(x, y, ...)` fits/updates model given input matrix `x` and target vector `y`. `x` shape = (n_samples, n_features)

`$predict(x, ...)` predicts output `x`

`$coef()` return coefficients of the regression model

`$dump()` create dump of the model (actually `list` with current model parameters)

`$load(x)` load/initialize model from dump)

Arguments

ftrl `FTRL` object

x Input sparse matrix - native format is `Matrix::RsparseMatrix`. If `x` is in different format, model will try to convert it to `RsparseMatrix` with `as(x, "RsparseMatrix")` call

learning_rate learning rate

learning_rate_decay learning rate which controls decay. Please refer to FTRL paper for details. Usually convergense does not heavily depend on this parameter, so default value 0.5 is safe.

lambda regularization parameter

l1_ratio controls L1 vs L2 penalty mixing. 1 = Lasso regression, 0 = Ridge regression. Elastic net is in between.

dropout dropout - percentage of random features to exclude from each sample. Acts as regularization.

family a description of the error distribution and link function to be used in the model. Only `binomial` (or logistic regression) supported at the moment.

Examples

```
library(rsparse)
library(Matrix)
i = sample(1000, 1000 * 100, TRUE)
j = sample(1000, 1000 * 100, TRUE)
y = sample(c(0, 1), 1000, TRUE)
x = sample(c(-1, 1), 1000 * 100, TRUE)
odd = seq(1, 99, 2)
x[i %in% which(y == 1) & j %in% odd] = 1
```

```
m = sparseMatrix(i = i, j = j, x = x, dims = c(1000, 1000), giveCsparse = FALSE)
x = as(m, "RsparseMatrix")

ftrl = FTRL$new(learning_rate = 0.01, learning_rate_decay = 0.1,
lambda = 10, l1_ratio = 1, dropout = 0)
ftrl$partial_fit(x, y)

w = ftrl$coef()
head(w)
sum(w != 0)
p = ftrl$predict(m)
```

GloVe *Creates Global Vectors matrix factorization model.*

Description

GloVe matrix factorization model. Model can be trained via fully can asynchronous and parallel AdaGrad with `$fit_transform()` method.

Usage

```
GloVe
```

Format

`R6Class` object.

Fields

`components` represents context embeddings

`shuffle logical = FALSE` by default. Defines shuffling before each SGD iteration. Generally shuffling is a good idea for stochastic-gradient descent, but from my experience in this particular case it does not improve convergence.

Usage

For usage details see **Methods, Arguments and Examples** sections.

```
glove = GloVe$new(rank, x_max, learning_rate = 0.15,
                          alpha = 0.75, lambda = 0.0, shuffle = FALSE)
glove$fit_transform(x, n_iter = 10L, convergence_tol = -1,
              n_threads = getOption("rsparse_omp_threads", 1L), ...)
glove$components
```

Methods

`$new(rank, x_max, learning_rate = 0.15, alpha = 0.75, lambda = 0, shuffle = FALSE)`
: Constructor for Global vectors model. For description of arguments see **Arguments** section.

`$fit_transform(x, n_iter = 10L, convergence_tol = -1, n_threads = getOption("rsparse_omp_threads",`
: fit Glove model given input matrix x

Arguments

glove A `GloVe` object

x An input term co-occurence matrix. Preferably in `dgTMatrix` format

n_iter `integer` number of SGD iterations

rank desired dimension for the latent vectors

x_max `integer` maximum number of co-occurrences to use in the weighting function. see the GloVe paper for details: http://nlp.stanford.edu/pubs/glove.pdf

learning_rate `numeric` learning rate for SGD. I do not recommend that you modify this parameter, since AdaGrad will quickly adjust it to optimal

convergence_tol `numeric = -1` defines early stopping strategy. We stop fitting when one of two following conditions will be satisfied: (a) we have used all iterations, or (b) `cost_previous_iter / cost_current_it` `convergence_tol`. By default perform all iterations.

alpha `numeric = 0.75` the alpha in weighting function formula : $f(x) = 1 if x > x_max ax; else (x/x_m ax)^a lpha$

lambda `numeric = 0.0` regularization parameter

init `list(w_i = NULL, b_i = NULL, w_j = NULL, b_j = NULL)` initialization for embeddings (w_i, w_j) and biases (b_i, b_j). `w_i, w_j` - numeric matrices, should number of #rows = rank, #columns - expected number of rows/columns in input matrix. `b_i, b_j` = numeric vectors, should have length of # expected number of rows/columns in input matrix

See Also

http://nlp.stanford.edu/projects/glove/

Examples

```
temp = tempfile()
download.file('http://mattmahoney.net/dc/text8.zip', temp)
text8 = readLines(unz(temp, "text8"))
it = itoken(text8)
vocabulary = create_vocabulary(it)
vocabulary = prune_vocabulary(vocabulary, term_count_min = 5)
v_vect = vocab_vectorizer(vocabulary)
tcm = create_tcm(it, v_vect, skip_grams_window = 5L)
glove_model = GloVe$new(rank = 50, x_max = 10, learning_rate = .25)
# fit model and get word vectors
word_vectors_main = glove_model$fit_transform(tcm, n_iter = 10)
word_vectors_context = glove_model$components
word_vectors = word_vectors_main + t(word_vectors_context)
```

LinearFlow *Creates Linear-FLow model for one-class collaborative filtering*

Description

Creates **Linear-FLow** model described in Practical Linear Models for Large-Scale One-Class Collaborative Filtering. The goal is to find item-item (or user-user) similarity matrix which is **low-rank and has small Frobenius norm**. Such double regularization allows to better control the generalization error of the model. Idea of the method is somewhat similar to **Sparse Linear Methods(SLIM)** but scales to large datasets much better.

Usage

```
LinearFlow
```

Format

`R6Class` object.

Usage

For usage details see **Methods, Arguments and Examples** sections.

```
model = LinearFlow$new( rank = 8L,
                        lambda = 0,
                        init = NULL,
                        preprocess = identity,
                       solve_right_singular_vectors = c("soft_impute", "svd")
                        ...)
model$fit_transform(x, ...)
model$transform(x, ...)
model$predict(x, k, not_recommend = x, ...)
model$components
model$v
model$cross_validate_lambda(x, x_train, x_test, lambda = "auto@10",
                     metric = "map@10", not_recommend = x_train, ...)
```

Methods

`$new(rank = 8L, lambda = 0, init = NULL, preprocess = identity, solve_right_singular_vectors = c("s`
: creates Linear-FLow model with `rank` latent factors. If `init` (right singular vectors of the user-item interactions matrix) is provided then model initialized with its values.

`$fit_transform(x, ...)`
: fits model to an input user-item interaction matrix. **Returns user embeddings matrix of the size** `n_users * rank`

`$transform(x, ...)`
: transforms user-item interaction matrix into user-embeddings matrix.

`$predict(x, k, not_recommend = x, ...)` predicts **top k** item ids for users `x`. Users features should be defined the same way as they were defined in training data - as **sparse matrix**. Column names (=item ids) should be in the same order as in the `fit_transform()`.

preprocess `function = identity()` by default. User spectified function which will be applied to user-item interaction matrix before running matrix factorization (also applied in inference time before making predictions).

`$cross_validate_lambda(x, x_train, x_test, lambda = "auto@10", metric = "map@10", not_recommend = x,` perfroms search of the best regularization parameter `lambda`:

1. Model is trained on `x` data
2. Then model makes predictions based on `x_train` data
3. And finally these predications are validated using specified `metric` against `x_test` data

Note that this is implemented smartly with **"warm starts"**. So it is very cheap - **cost is almost the same as for single fit** of the model. The only considerable additional cost is time to predict *top k* items. In most cases automatic lambda like `lambda = "auto@20"` is able to find good value of the parameter

`$components` item factors matrix of size `rank * n_items`. In the paper this matrix is called **Y**

`$v` right singular vector of the user-item matrix. Size is `n_items * rank`. In the paper this matrix is called **v**

Arguments

model A `LinearFlow` model.

x An input sparse user-item matrix (inherits from `sparseMatrix`)

rank `integer` - number of latent factors

lambda `numeric` - regularization parameter or sequence of regularization values for `cross_validate_lambda` method.

not_recommend `sparse matrix` or `NULL` - points which items should be excluided from recommendations for a user. By default it excludes previously seen/consumed items.

metric metric to use in evaluation of top-k recommendations. Currently only `map@k` and `ndcg@k` are supported (`k` can be any integer).

... other arguments (not used at the moment)

See Also

- http://www.bkveton.com/docs/ijcai2016.pdf
- http://www-users.cs.umn.edu/~xning/slides/ICDM2011_slides.pdf

Examples

```
data('movielens100k')
train = movielens100k[1:900, ]
cv = movielens100k[901:nrow(movielens100k), ]
model = LinearFlow$new(rank = 10, lambda = 0, init = NULL,
                       solve_right_singular_vectors = "svd")
user_emb = model$fit_transform(train)
preds = model$predict(cv, k = 10)
```

matmult *Multithreaded Sparse-Dense Matrix Multiplication*

Description

Multithreaded `%*%`, `crossprod`, `tcrossprod` for sparse-dense matrix multiplication

Usage

```
## S4 method for signature 'dgRMatrix,matrix'
x %*% y

## S4 method for signature 'dgRMatrix,matrix'
tcrossprod(x, y)

## S4 method for signature 'matrix,dgCMatrix'
x %*% y

## S4 method for signature 'matrix,dgCMatrix'
crossprod(x, y)
```

Arguments

`x, y`	dense `matrix` and sparse `Matrix::RsparseMatrix` / `Matrix::CsparseMatrix` matrices.

Details

Accelerates sparse-dense matrix multiplications using openmp. Applicable to the following pairs: (`dgRMatrix`, `matrix`), (`matrix`, `dgRMatrix`), (`dgCMatrix`, `matrix`), (`matrix`, `dgCMatrix`) combinations

Value

A dense `matrix`

Examples

```
library(Matrix)
data("movielens100k")
k = 10
nc = ncol(movielens100k)
nr = nrow(movielens100k)
x_nc = matrix(rep(1:k, nc), nrow = nc)
x_nr = t(matrix(rep(1:k, nr), nrow = nr))
csc = movielens100k
csr = as(movielens100k, "RsparseMatrix")
dense = as.matrix(movielens100k)
```

```
identical(csr %*% x_nc, dense %*% x_nc)
identical(x_nr %*% csc, x_nr %*% dense)
```

---

metrics *Ranking Metrics for Top-K Items*

Description

ap_k calculates **Average Precision at K** (ap@k). Please refer to https://en.wikipedia.org/wiki/Information_retrieval#Average_

ndcg_k() calculates **Normalized Discounted Cumulative Gain at K** (ndcg@k). Please refer to https://en.wikipedia.org/wiki/Discounted_cumulative_gain#Normalized_DCG

Usage

```
ap_k(predictions, actual, ...)

ndcg_k(predictions, actual, ...)
```

Arguments

predictions	matrix of predictions. Predctions can be defined 2 ways: 1. predictions = integer matrix with item indices (correspond to column numbers in actual) 2. predictions = character matrix with item identifiers (characters which correspond to colnames(actual)) which has attribute "indices" (integer matrix with item indices which correspond to column numbers in actual).
actual	sparse Matrix of relevant items. Each non-zero entry considered as relevant item. Value of the each non-zero entry considered as relevance for calculation of ndcg@k. It should inherit from Matrix::sparseMatrix. Internally Matrix::RsparseMatrix is used.
...	other arguments (not used at the moment)

Examples

```
predictions = matrix(
  c(5L, 7L, 9L, 2L),
  nrow = 1
)
actual = matrix(
  c(0, 0, 0, 0, 1, 0, 1, 0, 1, 0),
  nrow = 1
)
actual = as(actual, "RsparseMatrix")
identical(rsparse::ap_k(predictions, actual), 1)
```

`movielens100k` *MovieLens 100K Dataset*

Description

This data set consists of:

1. 100,000 ratings (1-5) from 943 users on 1682 movies.
2. Each user has rated at least 20 movies.

MovieLens data sets were collected by the GroupLens Research Project at the University of Minnesota.

Usage

```
data("movielens100k")
```

Format

A sparse column-compressed matrix (`Matrix::dgCMatrix`) with 943 rows and 1682 columns.

1. rows are users
2. columns are movies
3. values are ratings

Source

https://en.wikipedia.org/wiki/MovieLens#Datasets

`PureSVD` *Soft-SVD decompomposition*

Description

Creates matrix factorization model based on Soft-SVD. Soft SVD is very similar to truncated SVD with ability do add regularization based on nuclear norm.

Usage

```
PureSVD
```

Format

`R6Class` object.

Usage

For usage details see **Methods, Arguments and Examples** sections.

```
model = PureSVD$new(rank = 10L,
                    lambda = 0,
                    init = NULL,
                    preprocess = identity,
                    ...)
model$fit_transform(x, n_iter = 5L, ...)
model$transform(x, ...)
model$predict(x, k, not_recommend = x, ...)
model$components
```

Methods

`$new(rank = 10L, lambda = 0, init = NULL, preprocess = identity, ... )` creates matrix factorization model model with at most `rank` latent factors. If `init` is not null then initializes with provided SVD solution

`$fit_transform(x, n_iter = 5L, ...)` fits model to an input user-item matrix. **Returns factor matrix for users of size** `n_users * rank`

`$transform(x, ...)` Calculates user embeddings from given `x` user-item matrix. Result is `n_users * rank` matrix

`$predict(x, k, not_recommend = x, ...)` predict `top k` item ids for users `x` (= column names from the matrix passed to `fit_transform()` method). Users features should be defined the same way as they were defined in training data - as **sparse matrix** of confidence values (implicit feedback) or ratings (explicit feedback). Column names (=item ids) should be in the same order as in the `fit_transform()`.

`$components` item factors matrix of size `rank * n_items`

Arguments

model A `PureSVD` model.

x An input sparse user-item matrix(of class `dgCMatrix`).

rank `integer` - maximum number of latent factors

lambda `numeric` - regularization parameter for nuclear norm

preprocess `function` = `identity()` by default. User spectified function which will be applied to user-item interaction matrix before running matrix factorization (also applied in inference time before making predictions). For example we may want to normalize each row of user-item matrix to have 1 norm. Or apply `log1p()` to discount large counts.

not_recommend `sparse matrix` or `NULL` - points which items should be excluided from recommendations for a user. By default it excludes previously seen/consumed items.

convergence_tol `numeric = -Inf` defines early stopping strategy. We stop fitting when one of two following conditions will be satisfied: (a) we have used all iterations, or (b) relative change of frobenious norm of the two consequent solution is less then provided `convergence_tol`

... other arguments. Not used at the moment

Examples

```
data('movielens100k')
i_train = sample(nrow(movielens100k), 900)
i_test = setdiff(seq_len(nrow(movielens100k)), i_train)
train = movielens100k[i_train, ]
test = movielens100k[i_test, ]
rank = 32
lambda = 0
model = PureSVD$new(rank = rank,  lambda = lambda)
user_emb = model$fit_transform(sign(test), n_iter = 100, convergence_tol = 0.00001)
item_emb = model$components
preds = model$predict(sign(test), k = 1500, not_recommend = NULL)
mean(ap_k(preds, actual = test))
```

---

slice *CSR Matrices Slicing*

Description

natively slice CSR matrices without converting them to triplet/CSC

Usage

```
## S4 method for signature 'RsparseMatrix,index,index,logical'
x[i, j, drop = TRUE]

## S4 method for signature 'RsparseMatrix,missing,index,logical'
x[i, j, drop = TRUE]

## S4 method for signature 'RsparseMatrix,index,missing,logical'
x[i, j, drop = TRUE]

## S4 method for signature 'RsparseMatrix,missing,missing,logical'
x[i, j, drop = TRUE]
```

Arguments

x	input RsparseMatrix
i	row indices to subset
j	column indices to subset
drop	whether to simplify 1d matrix to a vector

Value

A RsparseMatrix

Examples

```
library(Matrix)
library(rsparse)
# dgCMatrix - CSC
m = rsparsematrix(20, 20, 0.1)
# make CSR
m = as(m, "RsparseMatrix")
inherits(m[1:2, ], "RsparseMatrix")
inherits(m[1:2, 3:4], "RsparseMatrix")
```

`soft_impute` *SoftImpute/SoftSVD matrix factorization*

Description

Fit SoftImpute/SoftSVD via fast alternating least squares. Based on the paper by Trevor Hastie, Rahul Mazumder, Jason D. Lee, Reza Zadeh by "Matrix Completion and Low-Rank SVD via Fast Alternating Least Squares" - https://arxiv.org/pdf/1410.2596.pdf

Usage

```
soft_impute(x, rank = 10L, lambda = 0, n_iter = 100L,
  convergence_tol = 0.001, init = NULL, final_svd = TRUE)

soft_svd(x, rank = 10L, lambda = 0, n_iter = 100L,
  convergence_tol = 0.001, init = NULL, final_svd = TRUE)
```

Arguments

- `x` sparse matrix. Both CSR `dgRMatrix` and CSC `dgCMatrix` are supported. CSR matrix is preffered because in this case algorithm will benefit from multithreaded CSR * dense matrix products (if OpenMP is supported on your platform). On many-cores machines this reduces fitting time significantly.
- `rank` maximum rank of the low-rank solution.
- `lambda` regularization parameter for the nuclear norm
- `n_iter` maximum number of iterations of the algorithms
- `convergence_tol` convergence tolerance. Internally functions keeps track of the relative change of the Frobenious norm of the two consequent iterations. If the change is less than `convergence_tol` then the process is considered as converged and function returns result.
- `init` svd like object with `u, v, d` components to initialize algorithm. Algorithm benefit from warm starts. `init` could be rank up `rank` of the maximum allowed rank. If `init` has rank less than max rank it will be padded automatically.
- `final_svd` `logical` whether need to make final preprocessing with SVD. This is not necessary but cleans up rank nicely - hithly recommnded to leave it `TRUE`.

Value

svd-like object - `list(u, v, d)`. `u, v, d` components represent left, right singular vectors and singular values.

Examples

```
set.seed(42)
data('movielens100k')
k = 10
seq_k = seq_len(k)
m = movielens100k[1:100, 1:200]
svd_ground_true = svd(m)
svd_soft_svd = soft_svd(m, rank = k, n_iter = 100, convergence_tol = 1e-6)
m_restored_svd = svd_ground_true$u[, seq_k]  %*%
   diag(x = svd_ground_true$d[seq_k]) %*%
   t(svd_ground_true$v[, seq_k])
m_restored_soft_svd = svd_soft_svd$u %*%
  diag(x = svd_soft_svd$d) %*%
  t(svd_soft_svd$v)
all.equal(m_restored_svd, m_restored_soft_svd, tolerance = 1e-1)
```

`WRMF` *Weighted Regularized Matrix Facrtorization for collaborative filtering*

Description

Creates matrix factorization model which could be solved with Alternating Least Squares (Weighted ALS for implicit feedback). For implicit feedback see (Hu, Koren, Volinsky)'2008 paper http://yifanhu.net/PUB/cf.pdf. For explicit feedback model is classic model for rating matrix decomposition with MSE error (without biases at the moment). These two algorithms are proven to work well in recommender systems.

Usage

```
WRMF
```

Format

`R6Class` object.

Usage

For usage details see **Methods, Arguments and Examples** sections.

```
model = WRMF$new(rank = 10L, lambda = 0,
                 feedback = c("implicit", "explicit"),
                 non_negative = FALSE,
                 solver = c("conjugate_gradient", "cholesky"),
                 cg_steps = 3L,
                 init = NULL)
model$fit_transform(x, n_iter = 5L, ...)
model$transform(x)
model$predict(x, k, not_recommend = x, items_exclude = NULL, ...)
model$components
model$remove_scorer(name)
```

Methods

`$new(rank = 10L, lambda = 0, feedback = c("implicit", "explicit"), non_negative = FALSE, solver = c` creates matrix factorization model model with `rank` latent factors. If `init` is provided then initialize item embeddings with its values.

`$fit_transform(x, n_iter = 5L, ...)` fits model to an input user-item matrix. (preferably in "dgCMatrix" format). For implicit feedback `x` should be a confidence matrix which corresponds to `1 + alpha * r_ui` in original paper. Usually `r_ui` corresponds to the number of interactions of user `u` and item `i`. For explicit feedback values in `x` represents ratings. **Returns factor matrix for users of size** `n_users * rank`

`$transform(x, ...)` Calculates user embeddings from given `x` user-item matrix. Result is `n_users * rank` matrix

`$predict(x, k, not_recommend = x, ...)` predicts `top k` item indices for users `x`. Additionally contains `scores` attribute - "score" values for each prediction. If model contains item ids (input matrix to `fit_transform()` had column-names then result additionally will have `ids` attribute - item ids which correspond to item indices. Users features `x` should be defined the same way as they were defined in training data - as **sparse matrix** of confidence values (implicit feedback) or ratings (explicit feedback). Column names (=item ids) should be in the same order as in the `fit_transform()`.

`$components` items embeddings matrix of size `rank * n_items`

Arguments

model A `WRMF` model.

x An input sparse user-item matrix(of class `dgCMatrix`). For explicit feedback should consists of ratings. For implicit feedback all positive interactions should be filled with **confidence** values. Missed interactions should me zeros/empty. So for simple case case when `confidence = 1 + alpha * x`

rank `integer` - number of latent factors

lambda `numeric` - regularization parameter

feedback `character` - feedback type - one of `c("implicit", "explicit")`

solver `character` - solver for "implicit feedback" problem. One of `c("conjugate_gradient", "cholesky")`. Usually approximate `"conjugate_gradient"` is significantly faster and solution is on par with exact `"cholesky"`

cg_steps `integer > 0` - max number of internal steps in conjugate gradient (if "conjugate_gradient" solver used). `cg_steps = 3` by default. Controls precision of linear equation solution at the each ALS step. Usually no need to tune this parameter.

preprocess `function` = `identity()` by default. User spectified function which will be applied to user-item interaction matrix before running matrix factorization (also applied in inference time before making predictions). For example we may want to normalize each row of user-item matrix to have 1 norm. Or apply `log1p()` to discount large counts. This essentially corresponds to the "confidence" function from (Hu, Koren, Volinsky)'2008 paper http://yifanhu.net/PUB/cf.pdf

precision one of `c("double", "float")`. Should embeeding matrices be usual numeric or float (from `float` package). The latter is usually 2x faster and consumes less RAM. BUT `float` matrices are not "base" objects. Use carefully.

not_recommend `sparse matrix` or `NULL` - points which items should be excluided from recommendations for a user. By default it excludes previously seen/consumed items.

items_exclude `character` = item ids or `integer` = item indices or `NULL` - items to exclude from recommendations for **all** users.

convergence_tol `numeric = -Inf` defines early stopping strategy. Model stops fitting when one of two following conditions is satisfied: (a) exceed number of iterations, or (b) `loss_previous_iter / loss_current_it`

... other arguments. Not used at the moment

See Also

- https://math.stackexchange.com/questions/1072451/analytic-solution-for-matrix-factorization-us 1073170#1073170
- http://activisiongamescience.github.io/2016/01/11/Implicit-Recommender-Systems-Biased-Matrix-F
- http://datamusing.info/blog/2015/01/07/implicit-feedback-and-collaborative-filtering/
- https://jessesw.com/Rec-System/
- http://danielnee.com/2016/09/collaborative-filtering-using-alternating-least-squares/
- http://www.benfrederickson.com/matrix-factorization/
- http://www.benfrederickson.com/fast-implicit-matrix-factorization/

Examples

```
data('movielens100k')
train = movielens100k[1:900, ]
cv = movielens100k[901:nrow(movielens100k), ]
model = WRMF$new(rank = 5,  lambda = 0, feedback = 'implicit')
user_emb = model$fit_transform(train, n_iter = 5, convergence_tol = -1)
item_emb = model$components
preds = model$predict(cv, k = 10, not_recommend = cv)
```

Index