SPAMS Documentation

Release 1

jp

Jan 15, 2019

CONTENTS

Contents:

sort(X[, mode])	sort the elements of X using quicksort
calcAAt(A)	Compute efficiently AAt = A*A’, when A is sparse and has a lot more columns than rows.
calcXAt(X, A)	Compute efficiently XAt = X*A’, when A is sparse and has a lot more columns than rows.
calcXY(X, Y)	Compute Z=XY using the BLAS library used by SPAMS.
calcXYt(X, Y)	Compute Z=XY’ using the BLAS library used by SPAMS.
calcXtY(X, Y)	Compute Z=X’Y using the BLAS library used by SPAMS.
bayer(X, offset)	bayer applies a Bayer pattern to an image X.
conjGrad(A, b[, x0, tol, itermax])	Conjugate gradient algorithm, sometimes faster than the equivalent python function solve.
invSym(A)	returns the inverse of a symmetric matrix A
normalize(A)	rescale the columns of X so that they have unit l2-norm.
sparseProject(U[, thrs, mode, lambda1, . . .])	sparseProject solves various optimization problems, including projections on a few convex sets.
lasso(X[, D, Q, q, return_reg_path, L, . . .])	lasso is an efficient implementation of the homotopy-LARS algorithm for solving the Lasso.
lassoMask(X, D, B[, L, lambda1, lambda2, . . .])	lasso is a variant of lasso that handles binary masks.
lassoWeighted(X, D, W[, L, lambda1, mode, . . .])	lassoWeighted is an efficient implementation of the LARS algorithm for solving the weighted Lasso.
omp(X, D[, L, eps, lambda1, . . .])	omp is an efficient implementation of the Orthogonal Matching Pursuit algorithm.
ompMask(X, D, B[, L, eps, lambda1, . . .])	ompMask is a variant of omp that allow using a binary mask B

Continued on next page

Table 1 – continued from previous page

cd(X, D, A0[, lambda1, mode, itermax, tol, . . .])	cd addresses l1-decomposition problem with a coordinate descent type of approach.
somp(X, D, list_groups[, L, eps, numThreads])	somp is an efficient implementation of a Simultaneous Orthogonal Matching Pursuit algorithm.
l1L2BCD(X, D, alpha0, list_groups[, . . .])	l1L2BCD is a solver for a Simultaneous signal decomposition formulation based on block coordinate descent.
fistaFlat(Y, X, W0[, return_optim_info, . . .])	fistaFlat solves sparse regularized problems.
fistaTree(Y, X, W0, tree[, . . .])	fistaTree solves sparse regularized problems.
fistaGraph(Y, X, W0, graph[, . . .])	fistaGraph solves sparse regularized problems.
proximalFlat(U[, return_val_loss, . . .])	proximalFlat computes proximal operators.
proximalTree(U, tree[, return_val_loss, . . .])	proximalTree computes a proximal operator.
proximalGraph(U, graph[, return_val_loss, . . .])	proximalGraph computes a proximal operator.
trainDL(X[, return_model, model, D, . . .])	trainDL is an efficient implementation of the dictionary learning technique presented in
structTrainDL(X[, return_model, model, D, . . .])	structTrainDL is an efficient implementation of the dictionary learning technique presented in
trainDL_Memory(X[, D, numThreads, . . .])	trainDL_Memory is an efficient but memory consuming variant of the dictionary learning technique presented in
nmf(X[, return_lasso, model, numThreads, . . .])	trainDL is an efficient implementation of the non-negative matrix factorization technique presented in
nnsc(X[, return_lasso, model, lambda1, . . .])	trainDL is an efficient implementation of the non-negative sparse coding technique presented in
archetypalAnalysis(X[, p, Z0, returnAB, . . .])	documentation to appear soon
decompSimplex(X, Z[, computeXtX, numThreads])	documentation to appear soon
simpleGroupTree(degrees)	makes a structure representing a tree given the degree of each level.
readGroupStruct(file)	reads a text file describing “simply” the structure of groups of variables needed by proximalGraph, proximalTree, fistaGraph, fistaTree and structTrainDL and builds the corresponding group structure.
groupStructOfString(s)	decode a multi-line string describing “simply” the structure of groups of variables needed by proximalGraph, proximalTree, fistaGraph, fistaTree and structTrainDL and builds the corresponding group structure.
graphOfGroupStruct(gstruct)	converts a group structure into the graph structure used by proximalGraph, fistaGraph or structTrainDL
treeOfGroupStruct(gstruct)	converts a group structure into the tree structure used by proximalTree, fistaTree or structTrainDL

CHAPTER ONE

SORT

`spams.sort(X, mode=True)`

sort the elements of X using quicksort

Args: X: double vector of size n mode: false for decreasing order (true by default)

Returns: Y: double vector of size n

Authors: Julien MAIRAL, 2010 (spams, matlab interface and documentation) Jean-Paul CHIEZE 2011-2012 (python interface)

CHAPTER TWO

CALCAAT

`spams.calcAAt(A)`

Compute efficiently AAt = A*A', when A is sparse

and has a lot more columns than rows. In some cases, it is up to 20 times faster than the equivalent python expression AAt=A*A';

Args: A: double sparse m x n matrix

Returns: AAt: double m x m matrix

Authors: Julien MAIRAL, 2009 (spams, matlab interface and documentation) Jean-Paul CHIEZE 2011-2012 (python interface)

CHAPTER

THREE

CALCXAT

`spams.calcXAt(X, A)`

Compute efficiently XAt = X*A', when A is sparse and has a

lot more columns than rows. In some cases, it is up to 20 times faster than the equivalent python expression;

Args: X: double m x n matrix A: double sparse p x n matrix

Returns: XAt: double m x p matrix

Authors: Julien MAIRAL, 2009 (spams, matlab interface and documentation) Jean-Paul CHIEZE 2011-2012 (python interface)

CHAPTER

FOUR

CALCXY

`spams.calcXY(X, Y)`

Compute Z=XY using the BLAS library used by SPAMS.

Args: X: double m x n matrix Y: double n x p matrix

Returns: Z: double m x p matrix

Authors: Julien MAIRAL, 2009 (spams, matlab interface and documentation) Jean-Paul CHIEZE 2011-2012 (python interface)

CHAPTER

FIVE

CALCXYT

`spams.calcXYt(X, Y)`

Compute Z=XY' using the BLAS library used by SPAMS.

Args: X: double m x n matrix Y: double p x n matrix

Returns: Z: double m x p matrix

Authors: Julien MAIRAL, 2009 (spams, matlab interface and documentation) Jean-Paul CHIEZE 2011-2012 (python interface)

CHAPTER

SIX

CALCXTY

`spams.calcXtY(X, Y)`

Compute Z=X'Y using the BLAS library used by SPAMS.

Args: X: double n x m matrix Y: double n x p matrix

Returns: Z: double m x p matrix

Authors: Julien MAIRAL, 2009 (spams, matlab interface and documentation) Jean-Paul CHIEZE 2011-2012 (python interface)

CHAPTER SEVEN

BAYER

`spams.bayer(X, offset)`

bayer applies a Bayer pattern to an image X. There are four possible offsets.

Args: X: double m x n matrix offset: scalar, 0,1,2 or 3

Returns: Y: double m x m matrix

Authors: Julien MAIRAL, 2009 (spams, matlab interface and documentation) Jean-Paul CHIEZE 2011-2012 (python interface)

CHAPTER

EIGHT

CONJGRAD

`spams.conjGrad(A, b, x0=None, tol=1e-10, itermax=None)`

Conjugate gradient algorithm, sometimes faster than the

equivalent python function solve. In order to solve Ax=b;

Args: A: double square n x n matrix. HAS TO BE POSITIVE DEFINITE b: double vector of length n. x0: double vector of length n. (optional) initial guess. tol: (optional) tolerance. itermax: (optional) maximum number of iterations.

Returns: x: double vector of length n.

Authors: Julien MAIRAL, 2009 (spams, matlab interface and documentation) Jean-Paul CHIEZE 2011-2012 (python interface)

CHAPTER

NINE

INVSYM

`spams.invSym(A)`

returns the inverse of a symmetric matrix A

Args: A: double n x n matrix

Returns: B: double n x n matrix

Authors: Julien MAIRAL, 2009 (spams, matlab interface and documentation) Jean-Paul CHIEZE 2011-2012 (python interface)

CHAPTER

TEN

NORMALIZE

spams.**normalize**(*A*)

rescale the columns of X so that they have unit l2-norm.

Args: X: double m x n matrix

Returns: Y: double m x n matrix

Authors: Julien MAIRAL, 2010 (spams, matlab interface and documentation) Jean-Paul CHIEZE 2011-2012 (python interface)

CHAPTER

ELEVEN

SPARSEPROJECT

`spams.`**`sparseProject`**`(`*U*, *thrs=1.0*, *mode=1*, *lambda1=0.0*, *lambda2=0.0*, *lambda3=0.0*, *pos=0*, *numThreads=-1*`)`

sparseProject solves various optimization problems, including projections on a few convex sets.

It aims at addressing the following problems for all columns u of U in parallel

1. **when mode=1 (projection on the l1-ball)** min_v ||u-v||_2^2 s.t. ||v||_1 <= thrs
2. **when mode=2** min_v ||u-v||_2^2 s.t. ||v||_2^2 + lamuda1||v||_1 <= thrs
3. **when mode=3** min_v ||u-v||_2^2 s.t ||v||_1 + 0.5lamuda1||v||_2^2 <= thrs
4. **when mode=4** min_v 0.5||u-v||_2^2 + lamuda1||v||_1 s.t ||v||_2^2 <= thrs
5. **when mode=5**

   min_v 0.5||u-v||_2^2 + lamuda1||v||_1 +lamuda2 FL(v) + … 0.5lamuda_3 ||v||_2^2

   where FL denotes a “fused lasso” regularization term.
6. when mode=6 min_v ||u-v||_2^2 s.t lamuda1||v||_1 +lamuda2 FL(v) + …

   0.5lamuda3||v||_2^2 <= thrs

   When pos=true and mode <= 4, it solves the previous problems with positivity constraints

Args:

U: double m x n matrix (input signals) m is the signal size n is the number of signals to project

Kwargs: thrs: (parameter) lambda1: (parameter) lambda2: (parameter) lambda3: (parameter) mode: (see above) pos: (optional, false by default) numThreads: (optional, number of threads for exploiting

multi-core / multi-cpus. By default, it takes the value -1, which automatically selects all the available CPUs/cores).

Returns: V: double m x n matrix (output matrix)

Authors: Julien MAIRAL, 2009 (spams, matlab interface and documentation) Jean-Paul CHIEZE 2011-2012 (python interface)

Note: this function admits a few experimental usages, which have not been extensively tested:

- single precision setting

CHAPTER

TWELVE

LASSO

spams.**lasso**(*X*, *D=None*, *Q=None*, *q=None*, *return_reg_path=False*, *L=-1*, *lambda1=None*, *lambda2=0.0*, *mode=2*, *pos=False*, *ols=False*, *numThreads=-1*, *max_length_path=-1*, *verbose=False*, *cholesky=False*)

lasso is an efficient implementation of the homotopy-LARS algorithm for solving the Lasso.

If the function is called this way spams.lasso(X,D = D, Q = None,…), it aims at addressing the following problems for all columns x of X, it computes one column alpha of A that solves

1. when mode=0

   min_{alpha} ||x-Dalpha||_2^2 s.t. ||alpha||_1 <= lambda1

2. when mode=1

   min_{alpha} ||alpha||_1 s.t. ||x-Dalpha||_2^2 <= lambda1

3. when mode=2

   min_{alpha} 0.5||x-Dalpha||_2^2 + lambda1||alpha||_1 +0.5 lambda2||alpha||_2^2

If the function is called this way spams.lasso(X,D = None, Q = Q, q = q,…), it solves the above optimisation problem, when Q=D'D and q=D'x.

Possibly, when pos=true, it solves the previous problems with positivity constraints on the vectors alpha

Args:

X: double m x n matrix (input signals) m is the signal size n is the number of signals to decompose

D: double m x p matrix (dictionary) p is the number of elements in the dictionary

Q: p x p double matrix (Q = D'D) q: p x n double matrix (q = D'X) verbose: verbose mode return_reg_path:

if true the function will return a tuple of matrices.

Kwargs: lambda1: (parameter) lambda2: (optional parameter for solving the Elastic-Net)

for mode=0 and mode=1, it adds a ridge on the Gram Matrix

L: (optional), maximum number of steps of the homotopy algorithm (can be used as a stopping criterion)

pos: (optional, adds non-negativity constraints on the coefficients, false by default)

mode: (see above, by default: 2) numThreads: (optional, number of threads for exploiting multi-core / multi-cpus. By default, it takes the value -1, which automatically selects all the available CPUs/cores).

cholesky: (optional, default false), choose between Cholesky implementation or one based on the matrix inversion Lemma

ols: (optional, default false), perform an orthogonal projection before returning the solution.

max_length_path: (optional) maximum length of the path, by default 4*p

Returns: A: double sparse p x n matrix (output coefficients) path: optional, returns the regularisation path for the first signal

A = spams.lasso(X,return_reg_path = False,…) (A,path) = spams.lasso(X,return_reg_path = True,…)

Authors: Julien MAIRAL, 2009 (spams, matlab interface and documentation) Jean-Paul CHIEZE 2011-2012 (python interface)

Note: this function admits a few experimental usages, which have not been extensively tested:

- single precision setting (even though the output alpha is double precision)

Examples:

```
import numpy as np
m = 5;n = 10;nD = 5
np.random.seed(0)
X = np.asfortranarray(np.random.normal(size=(m,n)))
X = np.asfortranarray(X / np.tile(np.sqrt((X*X).sum(axis=0)),(X.shape[0],1)))
D = np.asfortranarray(np.random.normal(size=(100,200)))
D = np.asfortranarray(D / np.tile(np.sqrt((D*D).sum(axis=0)),(D.shape[0],1)))
alpha = spams.lasso(X,D = D,return_reg_path = FALSE,lambda1 = 0.15)
```

CHAPTER THIRTEEN

LASSOMASK

`spams.`**`lassoMask`**(*X*, *D*, *B*, *L=-1*, *lambda1=None*, *lambda2=0.0*, *mode=2*, *pos=False*, *numThreads=-1*, *verbose=False*)

lasso is a variant of lasso that handles binary masks. It aims at addressing the following problems for all columns x of X, and beta of B, it computes one column alpha of A that solves

1. when mode=0

   min_{alpha} ||diag(beta)(x-Dalpha)||_2^2 s.t. ||alpha||_1 <= lambda1

2. when mode=1

   min_{alpha} ||alpha||_1 s.t. ||diag(beta)(x-Dalpha)||_2^2 <=
   lambda1*||beta||_0/m

3. when mode=2

   min_{alpha} 0.5||diag(beta)(x-Dalpha)||_2^2 +
   lambda1*(||beta||_0/m)*||alpha||_1 + (lambda2/2)||alpha||_2^2

Possibly, when pos=true, it solves the previous problems with positivity constraints on the vectors alpha

Args:

X: double m x n matrix (input signals) m is the signal size n is the number of signals to decompose

D: double m x p matrix (dictionary) p is the number of elements in the dictionary

B: boolean m x n matrix (mask) p is the number of elements in the dictionary

verbose: verbose mode

Kwargs: lambda1: (parameter) L: (optional, maximum number of elements of each

decomposition)

pos: (optional, adds positivity constraints on the coefficients, false by default)

mode: (see above, by default: 2) lambda2: (optional parameter for solving the Elastic-Net)

for mode=0 and mode=1, it adds a ridge on the Gram Matrix

numThreads: (optional, number of threads for exploiting multi-core / multi-cpus. By default, it takes the value -1, which automatically selects all the available CPUs/cores).

Returns: A: double sparse p x n matrix (output coefficients)

Authors: Julien MAIRAL, 2010 (spams, matlab interface and documentation) Jean-Paul CHIEZE 2011-2012 (python interface)

Note: this function admits a few experimental usages, which have not been extensively tested:

- single precision setting (even though the output alpha is double precision)

CHAPTER

FOURTEEN

LASSOWEIGHTED

`spams.`**`lassoWeighted`**`(`*X*, *D*, *W*, *L=-1*, *lambda1=None*, *mode=2*, *pos=False*, *numThreads=-1*, *verbose=False*`)`

lassoWeighted is an efficient implementation of the LARS algorithm for solving the weighted Lasso. It is optimized for solving a large number of small or medium-sized decomposition problem (and not for a single large one).

It first computes the Gram matrix D'D and then perform a Cholesky-based OMP of the input signals in parallel. For all columns x of X, and w of W, it computes one column alpha of A which is the solution of

1. when mode=0

   min_{alpha} ||x-Dalpha||_2^2 s.t. ||diag(w)alpha||_1 <= lambda1

2. when mode=1

   min_{alpha} ||diag(w)alpha||_1 s.t. ||x-Dalpha||_2^2 <= lambda1

3. when mode=2

   min_{alpha} 0.5||x-Dalpha||_2^2 + lambda1||diag(w)alpha||_1

Possibly, when pos=true, it solves the previous problems with positivity constraints on the vectors alpha

Args:

X: double m x n matrix (input signals) m is the signal size n is the number of signals to decompose

D: double m x p matrix (dictionary) p is the number of elements in the dictionary

W: double p x n matrix (weights) verbose: verbose mode

Kwargs: lambda1: (parameter) L: (optional, maximum number of elements of each

decomposition)

pos: (optional, adds positivity constraints on the coefficients, false by default)

mode: (see above, by default: 2) numThreads: (optional, number of threads for exploiting

multi-core / multi-cpus. By default, it takes the value -1, which automatically selects all the available CPUs/cores).

Returns: A: double sparse p x n matrix (output coefficients)

Authors: Julien MAIRAL, 2009 (spams, matlab interface and documentation) Jean-Paul CHIEZE 2011-2012 (python interface)

Note: this function admits a few experimental usages, which have not been extensively tested:

- single precision setting (even though the output alpha is double precision)

CHAPTER

FIFTEEN

OMP

`spams.omp(X, D, L=None, eps=None, lambda1=None, return_reg_path=False, numThreads=-1)`

omp is an efficient implementation of the Orthogonal Matching Pursuit algorithm. It is optimized for solving a large number of small or medium-sized decomposition problem (and not for a single large one).

It first computes the Gram matrix D'D and then perform a Cholesky-based OMP of the input signals in parallel. X=[x^1,. . . ,x^n] is a matrix of signals, and it returns a matrix A=[alpha^1,. . . ,alpha^n] of coefficients.

it addresses for all columns x of X, min_{alpha} ||alpha||_0 s.t ||x-Dalpha||_2^2 <= eps or min_{alpha} ||x-Dalpha||_2^2 s.t. ||alpha||_0 <= L or min_{alpha} 0.5||x-Dalpha||_2^2 + lambda1||alpha||_0

Args:

X: double m x n matrix (input signals) m is the signal size n is the number of signals to decompose

D: double m x p matrix (dictionary) p is the number of elements in the dictionary All the columns of D should have unit-norm !

return_reg_path: if true the function will return a tuple of matrices.

Kwargs:

L: (optional, maximum number of elements in each decomposition, min(m,p) by default)

eps: (optional, threshold on the squared l2-norm of the residual, 0 by default

lambda1: (optional, penalty parameter, 0 by default numThreads: (optional, number of threads for exploiting

multi-core / multi-cpus. By default, it takes the value -1, which automatically selects all the available CPUs/cores).

Returns:

A: double sparse p x n matrix (output coefficients) path (optional): double dense p x L matrix (regularization path of the first signal) A = spams.omp(X,D,L,eps,return_reg_path = False,. . .) (A,path) = spams.omp(X,D,L,eps,return_reg_path = True,. . .)

Authors: Julien MAIRAL, 2009 (spams, matlab interface and documentation) Jean-Paul CHIEZE 2011-2012 (python interface)

Note: this function admits a few experimental usages, which have not been extensively tested:

- single precision setting (even though the output alpha is double precision)
- Passing an int32 vector of length n to L provides a different parameter L for each input signal x_i

- Passing a double vector of length n to eps and or lambda1 provides a different parameter eps (or lambda1) for each input signal x_i

CHAPTER

SIXTEEN

OMPMASK

`spams.ompMask`(*X*, *D*, *B*, *L=None*, *eps=None*, *lambda1=None*, *return_reg_path=False*, *numThreads=-1*)

ompMask is a variant of omp that allow using a binary mask B

for all columns x of X, and columns beta of B, it computes a column alpha of A by addressing
min_{alpha} ||alpha||_0 s.t ||diag(beta)*(x-Dalpha)||_2^2

<= eps*||beta||_0/m

or min_{alpha} ||diag(beta)*(x-Dalpha)||_2^2 s.t. ||alpha||_0 <= L or min_{alpha} 0.5||diag(beta)*(x-Dalpha)||_2^2 + lambda1||alpha||_0

Args:

X: double m x n matrix (input signals) m is the signal size n is the number of signals to decompose

D: double m x p matrix (dictionary) p is the number of elements in the dictionary All the columns of D should have unit-norm !

B: boolean m x n matrix (mask) p is the number of elements in the dictionary

return_reg_path: if true the function will return a tuple of matrices.

Kwargs:

L: (optional, maximum number of elements in each decomposition, min(m,p) by default)

eps: (optional, threshold on the squared l2-norm of the residual, 0 by default

lambda1: (optional, penalty parameter, 0 by default numThreads: (optional, number of threads for exploiting

multi-core / multi-cpus. By default, it takes the value -1, which automatically selects all the available CPUs/cores).

Returns:

A: double sparse p x n matrix (output coefficients) path (optional): double dense p x L matrix (regularization path of the first signal) A = spams.ompMask(X,D,B,L,eps,return_reg_path = False,...) (A,path) = spams.ompMask(X,D,B,L,eps,return_reg_path = True,...)

Authors: Julien MAIRAL, 2010 (spams, matlab interface and documentation) Jean-Paul CHIEZE 2011-2012 (python interface)

Note: this function admits a few experimental usages, which have not been extensively tested:

- single precision setting (even though the output alpha is double precision)
- Passing an int32 vector of length n to L provides a different parameter L for each input signal x_i

- Passing a double vector of length n to eps and or lambda1 provides a different parameter eps (or lambda1) for each input signal x_i

CHAPTER SEVENTEEN

CD

`spams.cd`(*X*, *D*, *A0*, *lambda1=None*, *mode=2*, *itermax=100*, *tol=0.001*, *numThreads=-1*)

cd addresses l1-decomposition problem with a coordinate descent type of approach.

It is optimized for solving a large number of small or medium-sized decomposition problem (and not for a single large one). It first computes the Gram matrix D'D. This method is particularly well adapted when there is low correlation between the dictionary elements and when one can benefit from a warm restart. It aims at addressing the two following problems for all columns x of X, it computes a column alpha of A such that

2. when mode=1

   min_{alpha} ||alpha||_1 s.t. ||x-Dalpha||_2^2 <= lambda1 For this constraint setting, the method solves a sequence of penalized problems (corresponding to mode=2) and looks for the corresponding Lagrange multplier with a simple but efficient heuristic.

3. when mode=2

   min_{alpha} 0.5||x-Dalpha||_2^2 + lambda1||alpha||_1

Args:

X: double m x n matrix (input signals) m is the signal size n is the number of signals to decompose

D: double m x p matrix (dictionary) p is the number of elements in the dictionary All the columns of D should have unit-norm !

A0: double sparse p x n matrix (initial guess)

Kwargs: lambda1: (parameter) mode: (optional, see above, by default 2) itermax: (maximum number of iterations) tol: (tolerance parameter) numThreads: (optional, number of threads for exploiting

multi-core / multi-cpus. By default, it takes the value -1, which automatically selects all the available CPUs/cores).

Returns: A: double sparse p x n matrix (output coefficients)

Authors: Julien MAIRAL, 2009 (spams, matlab interface and documentation) Jean-Paul CHIEZE 2011-2012 (python interface)

Note: this function admits a few experimental usages, which have not been extensively tested:

- single precision setting (even though the output alpha is double precision)

CHAPTER

EIGHTEEN

SOMP

`spams.`**`somp`**(*X*, *D*, *list_groups*, *L=None*, *eps=0.0*, *numThreads=-1*)

somp is an efficient implementation of a Simultaneous Orthogonal Matching Pursuit algorithm. It is optimized for solving a large number of small or medium-sized decomposition problem (and not for a single large one).

It first computes the Gram matrix D'D and then perform a Cholesky-based OMP of the input signals in parallel. It aims at addressing the following NP-hard problem

X is a matrix structured in groups of signals, which we denote by X=[X_1,. . . ,X_n]

for all matrices X_i of X, min_{A_i} ||A_i||_{0,infty} s.t ||X_i-D A_i||_2^2 <= eps*n_i where n_i is the number of columns of X_i

or

min_{A_i} ||X_i-D A_i||_2^2 s.t. ||A_i||_{0,infty} <= L

Args:

X: double m x N matrix (input signals) m is the signal size N is the total number of signals

D: double m x p matrix (dictionary) p is the number of elements in the dictionary All the columns of D should have unit-norm !

list_groups: int32 vector containing the indices (starting at 0) of the first elements of each groups.

Kwargs: L: (maximum number of elements in each decomposition) eps: (threshold on the squared l2-norm of the residual numThreads: (optional, number of threads for exploiting

multi-core / multi-cpus. By default, it takes the value -1, which automatically selects all the available CPUs/cores).

Returns: alpha: double sparse p x N matrix (output coefficients)

Authors: Julien MAIRAL, 2010 (spams, matlab interface and documentation) Jean-Paul CHIEZE 2011-2012 (python interface)

Note: this function admits a few experimental usages, which have not been extensively tested:

- single precision setting (even though the output alpha is double precision)

CHAPTER

NINETEEN

L1L2BCD

spams.**l1L2BCD**(*X*, *D*, *alpha0*, *list_groups*, *lambda1=None*, *mode=2*, *itermax=100*, *tol=0.001*, *numThreads=-1*)

l1L2BCD is a solver for a Simultaneous signal decomposition formulation based on block coordinate descent.

X is a matrix structured in groups of signals, which we denote by X=[X_1,. . . ,X_n]

if mode=2, it solves for all matrices X_i of X, min_{A_i} 0.5||X_i-D A_i||_2^2 + lambda1/sqrt(n_i)||A_i||_{1,2} where n_i is the number of columns of X_i

if mode=1, it solves min_{A_i} ||A_i||_{1,2} s.t. ||X_i-D A_i||_2^2 <= n_i lambda1

Args:

X: double m x N matrix (input signals) m is the signal size N is the total number of signals

D: double m x p matrix (dictionary) p is the number of elements in the dictionary

alpha0: double dense p x N matrix (initial solution) list_groups: int32 vector containing the indices (starting at 0)

of the first elements of each groups.

Kwargs: lambda1: (regularization parameter) mode: (see above, by default 2) itermax: (maximum number of iterations, by default 100) tol: (tolerance parameter, by default 0.001) numThreads: (optional, number of threads for exploiting

multi-core / multi-cpus. By default, it takes the value -1, which automatically selects all the available CPUs/cores).

Returns: alpha: double sparse p x N matrix (output coefficients)

Authors: Julien MAIRAL, 2010 (spams, matlab interface and documentation) Jean-Paul CHIEZE 2011-2012 (python interface)

Note: this function admits a few experimental usages, which have not been extensively tested:

- single precision setting (even though the output alpha is double precision)

CHAPTER TWENTY

FISTAFLAT

`spams.`**`fistaFlat`**(*Y*, *X*, *W0*, *return_optim_info=False*, *numThreads=-1*, *max_it=1000*, *L0=1.0*, *fixed_step=False*, *gamma=1.5*, *lambda1=1.0*, *delta=1.0*, *lambda2=0.0*, *lambda3=0.0*, *a=1.0*, *b=0.0*, *c=1.0*, *tol=1e-06*, *it0=100*, *max_iter_backtracking=1000*, *compute_gram=False*, *lin_admm=False*, *admm=False*, *intercept=False*, *resetflow=False*, *regul=''*, *loss=''*, *verbose=False*, *pos=False*, *clever=False*, *log=False*, *ista=False*, *subgrad=False*, *logName=''*, *is_inner_weights=False*, *inner_weights=None*, *size_group=1*, *groups=None*, *sqrt_step=True*, *transpose=False*, *linesearch_mode=0*)

fistaFlat solves sparse regularized problems.

X is a design matrix of size m x p X=[x^1,…,x^n]', where the x_i's are the rows of X Y=[y^1,…,y^n] is a matrix of size m x n It implements the algorithms FISTA, ISTA and subgradient descent.

- if loss='square' and regul is a regularization function for vectors, the entries of Y are real-valued, W = [w^1,…,w^n] is a matrix of size p x n For all column y of Y, it computes a column w of W such that

  w = argmin 0.5||y- X w||_2^2 + lambda1 psi(w)

- if loss='square' and regul is a regularization function for matrices the entries of Y are real-valued, W is a matrix of size p x n. It computes the matrix W such that

  W = argmin 0.5||Y- X W||_F^2 + lambda1 psi(W)

- loss='square-missing' same as loss='square', but handles missing data represented by NaN (not a number) in the matrix Y

- if loss='logistic' and regul is a regularization function for vectors, the entries of Y are either -1 or +1, W = [w^1,…,w^n] is a matrix of size p x n For all column y of Y, it computes a column w of W such that

  w = argmin (1/m)sum_{j=1}^m log(1+e^(-y_j x^j' w)) + lambda1 psi(w),

  where x^j is the j-th row of X.

- if loss='logistic' and regul is a regularization function for matrices the entries of Y are either -1 or +1, W is a matrix of size p x n

  W = argmin sum_{i=1}^n(1/m)sum_{j=1}^m log(1+e^(-y^i_j x^j' w^i)) + lambda1 psi(W)

- if loss='multi-logistic' and regul is a regularization function for vectors, the entries of Y are in {0,1,…,N} where N is the total number of classes W = [W^1,…,W^n] is a matrix of size p x Nn, each submatrix W^i is of size p x N for all submatrix WW of W, and column y of Y, it computes

WW = argmin (1/m)sum_{j=1}^m log(sum_{j=1}^r e^(x^j'(ww^j-ww^{y_j}))) + lambda1 sum_{j=1}^N psi(ww^j),

where ww^j is the j-th column of WW.

- if loss='multi-logistic' and regul is a regularization function for matrices, the entries of Y are in {0,1,...,N} where N is the total number of classes W is a matrix of size p x N, it computes

  W = argmin (1/m)sum_{j=1}^m log(sum_{j=1}^r e^(x^j'(w^j-w^{y_j}))) + lambda1 psi(W)

  where ww^j is the j-th column of WW.

- **loss='cur' useful to perform sparse CUR matrix decompositions,** W = argmin 0.5||Y-X*W*X||_F^2 + lambda1 psi(W)

The function psi are those used by proximalFlat (see documentation)

This function can also handle intercepts (last row of W is not regularized), and/or non-negativity constraints on W, and sparse matrices for X

Args: Y: double dense m x n matrix X: double dense or sparse m x p matrix W0: double dense p x n matrix or p x Nn matrix (for multi-logistic loss)

initial guess

return_optim_info: if true the function will return a tuple of matrices.

Kwargs: loss: (choice of loss, see above) regul: (choice of regularization, see function proximalFlat) lambda1: (regularization parameter) lambda2: (optional, regularization parameter, 0 by default) lambda3: (optional, regularization parameter, 0 by default) verbose: (optional, verbosity level, false by default) pos: (optional, adds positivity constraints on the

coefficients, false by default)

transpose: (optional, transpose the matrix in the regularization function) size_group: (optional, for regularization functions assuming a group

structure)

groups: (int32, optional, for regularization functions assuming a group structure, see proximalFlat)

numThreads: (optional, number of threads for exploiting multi-core / multi-cpus. By default, it takes the value -1, which automatically selects all the available CPUs/cores).

max_it: (optional, maximum number of iterations, 100 by default) it0: (optional, frequency for computing duality gap, every 10 iterations by default) tol: (optional, tolerance for stopping criteration, which is a relative duality gap

if it is available, or a relative change of parameters).

gamma: (optional, multiplier for increasing the parameter L in fista, 1.5 by default) L0: (optional, initial parameter L in fista, 0.1 by default, should be small enough) fixed_step: (deactive the line search for L in fista and use L0 instead) linesearch_mode: (line-search scheme when ista=true: 0: default, monotonic backtracking scheme 1: monotonic backtracking scheme, with restart at each iteration 2: Barzilai-Borwein step sizes (similar to SparSA by Wright et al.) 3: non-monotonic backtracking compute_gram: (optional, pre-compute X^TX, false by default). intercept: (optional, do not regularize last row of W, false by default). ista: (optional, use ista instead of fista, false by default). subgrad: (optional, if not ista, use subradient descent instead of fista, false by default). a: b: (optional, if subgrad, the gradient step is a/(t+b) also similar options as proximalFlat

the function also implements the ADMM algorithm via an option admm=true. It is not documented and you need to look at the source code to use it.

delta: undocumented; modify at your own risks! c: undocumented; modify at your own risks! max_iter_backtracking: undocumented; modify at your own risks! lin_admm: undocumented; modify at your own risks! admm: undocumented; modify at your own risks! resetflow: undocumented; modify at your own risks! clever: undocumented; modify at your own risks! log: undocumented; modify at your own risks! logName: undocumented; modify at your own risks! is_inner_weights: undocumented; modify at your own risks! inner_weights: undocumented; modify at your own risks! sqrt_step: undocumented; modify at your own risks!

Returns: W: double dense p x n matrix or p x Nn matrix (for multi-logistic loss) optim: optional, double dense 4 x n matrix.

first row: values of the objective functions. third row: values of the relative duality gap (if available) fourth row: number of iterations

optim_info: vector of size 4, containing information of the optimization. W = spams.fistaFlat(Y,X,W0,return_optim_info = False,…) (W,optim_info) = spams.fistaFlat(Y,X,W0,return_optim_info = True,…)

Authors: Julien MAIRAL, 2010 (spams, matlab interface and documentation) Jean-Paul CHIEZE 2011-2012 (python interface)

Note:

Valid values for the regularization parameter (regul) are: "l0", "l1", "l2", "linf", "l2-not-squared", "elastic-net", "fused-lasso", "group-lasso-l2", "group-lasso-linf", "sparse-group-lasso-l2", "sparse-group-lasso-linf", "l1l2", "l1linf", "l1l2+l1", "l1linf+l1", "tree-l0", "tree-l2", "tree-linf", "graph", "graph-ridge", "graph-l2", "multi-task-tree", "multi-task-graph", "l1linf-row-column", "trace-norm", "trace-norm-vec", "rank", "rank-vec", "none"

CHAPTER TWENTYONE

FISTATREE

spams.**fistaTree**(*Y*, *X*, *W0*, *tree*, *return_optim_info=False*, *numThreads=-1*, *max_it=1000*, *L0=1.0*, *fixed_step=False*, *gamma=1.5*, *lambda1=1.0*, *delta=1.0*, *lambda2=0.0*, *lambda3=0.0*, *a=1.0*, *b=0.0*, *c=1.0*, *tol=1e-06*, *it0=100*, *max_iter_backtracking=1000*, *compute_gram=False*, *lin_admm=False*, *admm=False*, *intercept=False*, *resetflow=False*, *regul=''*, *loss=''*, *verbose=False*, *pos=False*, *clever=False*, *log=False*, *ista=False*, *subgrad=False*, *logName=''*, *is_inner_weights=False*, *inner_weights=None*, *size_group=1*, *sqrt_step=True*, *transpose=False*, *linesearch_mode=0*)

fistaTree solves sparse regularized problems.

X is a design matrix of size m x p X=[x^1,…,x^n]', where the x_i's are the rows of X Y=[y^1,…,y^n] is a matrix of size m x n It implements the algorithms FISTA, ISTA and subgradient descent for solving

min_W loss(W) + lambda1 psi(W)

The function psi are those used by proximalTree (see documentation) for the loss functions, see the documentation of fistaFlat

This function can also handle intercepts (last row of W is not regularized), and/or non-negativity constraints on W and sparse matrices X

Args: Y: double dense m x n matrix X: double dense or sparse m x p matrix W0: double dense p x n matrix or p x Nn matrix (for multi-logistic loss)

initial guess

tree: named list (see documentation of proximalTree) return_optim_info:

if true the function will return a tuple of matrices.

Kwargs: loss: (choice of loss, see above) regul: (choice of regularization, see function proximalFlat) lambda1: (regularization parameter) lambda2: (optional, regularization parameter, 0 by default) lambda3: (optional, regularization parameter, 0 by default) verbose: (optional, verbosity level, false by default) pos: (optional, adds positivity constraints on the

coefficients, false by default)

transpose: (optional, transpose the matrix in the regularization function) size_group: (optional, for regularization functions assuming a group

structure)

numThreads: (optional, number of threads for exploiting multi-core / multi-cpus. By default, it takes the value -1, which automatically selects all the available CPUs/cores).

max_it: (optional, maximum number of iterations, 100 by default) it0: (optional, frequency for computing duality gap, every 10 iterations by default) tol: (optional, tolerance for stopping criteration, which is a relative duality gap

if it is available, or a relative change of parameters).

gamma: (optional, multiplier for increasing the parameter L in fista, 1.5 by default) L0: (optional, initial parameter L in fista, 0.1 by default, should be small enough) fixed_step: (deactive the line search for L in fista and use L0 instead) compute_gram: (optional, pre-compute X^TX, false by default). intercept: (optional, do not regularize last row of W, false by default). ista: (optional, use ista instead of fista, false by default). subgrad: (optional, if not ista, use subradient descent instead of fista, false by default). a: b: (optional, if subgrad, the gradient step is a/(t+b)

also similar options as proximalTree

the function also implements the ADMM algorithm via an option admm=true. It is not documented and you need to look at the source code to use it.

delta: undocumented; modify at your own risks! c: undocumented; modify at your own risks! max_iter_backtracking: undocumented; modify at your own risks! lin_admm: undocumented; modify at your own risks! admm: undocumented; modify at your own risks! resetflow: undocumented; modify at your own risks! clever: undocumented; modify at your own risks! log: undocumented; modify at your own risks! logName: undocumented; modify at your own risks! is_inner_weights: undocumented; modify at your own risks! inner_weights: undocumented; modify at your own risks! sqrt_step: undocumented; modify at your own risks!

Returns: W: double dense p x n matrix or p x Nn matrix (for multi-logistic loss) optim: optional, double dense 4 x n matrix.

first row: values of the objective functions. third row: values of the relative duality gap (if available) fourth row: number of iterations

optim_info: vector of size 4, containing information of the optimization. W = spams.fistaTree(Y,X,W0,tree,return_optim_info = False,…) (W,optim_info) = spams.fistaTree(Y,X,W0,tree,return_optim_info = True,…)

Authors: Julien MAIRAL, 2010 (spams, matlab interface and documentation) Jean-Paul CHIEZE 2011-2012 (python interface)

Note:

Valid values for the regularization parameter (regul) are: "l0", "l1", "l2", "linf", "l2-not-squared", "elastic-net", "fused-lasso", "group-lasso-l2", "group-lasso-linf", "sparse-group-lasso-l2", "sparse-group-lasso-linf", "l1l2", "l1linf", "l1l2+l1", "l1linf+l1", "tree-l0", "tree-l2", "tree-linf", "graph", "graph-ridge", "graph-l2", "multi-task-tree", "multi-task-graph", "l1linf-row-column", "trace-norm", "trace-norm-vec", "rank", "rank-vec", "none"

CHAPTER

TWENTYTWO

FISTAGRAPH

spams.**fistaGraph**(*Y*, *X*, *W0*, *graph*, *return_optim_info=False*, *numThreads=-1*, *max_it=1000*, *L0=1.0*, *fixed_step=False*, *gamma=1.5*, *lambda1=1.0*, *delta=1.0*, *lambda2=0.0*, *lambda3=0.0*, *a=1.0*, *b=0.0*, *c=1.0*, *tol=1e-06*, *it0=100*, *max_iter_backtracking=1000*, *compute_gram=False*, *lin_admm=False*, *admm=False*, *intercept=False*, *resetflow=False*, *regul=''*, *loss=''*, *verbose=False*, *pos=False*, *clever=False*, *log=False*, *ista=False*, *subgrad=False*, *logName=''*, *is_inner_weights=False*, *inner_weights=None*, *size_group=1*, *sqrt_step=True*, *transpose=False*, *linesearch_mode=0*)

fistaGraph solves sparse regularized problems.

X is a design matrix of size m x p X=[x^1,…,x^n]', where the x_i's are the rows of X Y=[y^1,…,y^n] is a matrix of size m x n It implements the algorithms FISTA, ISTA and subgradient descent.

It implements the algorithms FISTA, ISTA and subgradient descent for solving

min_W loss(W) + lambda1 psi(W)

The function psi are those used by proximalGraph (see documentation) for the loss functions, see the documentation of fistaFlat

This function can also handle intercepts (last row of W is not regularized), and/or non-negativity constraints on W.

Args: Y: double dense m x n matrix X: double dense or sparse m x p matrix W0: double dense p x n matrix or p x Nn matrix (for multi-logistic loss)

initial guess

graph: struct (see documentation of proximalGraph) return_optim_info:

if true the function will return a tuple of matrices.

Kwargs: loss: (choice of loss, see above) regul: (choice of regularization, see function proximalFlat) lambda1: (regularization parameter) lambda2: (optional, regularization parameter, 0 by default) lambda3: (optional, regularization parameter, 0 by default) verbose: (optional, verbosity level, false by default) pos: (optional, adds positivity constraints on the

coefficients, false by default)

numThreads: (optional, number of threads for exploiting multi-core / multi-cpus. By default, it takes the value -1, which automatically selects all the available CPUs/cores).

max_it: (optional, maximum number of iterations, 100 by default) it0: (optional, frequency for computing duality gap, every 10 iterations by default) tol: (optional, tolerance for stopping criteration, which is a relative duality gap if it is available, or a relative change of parameters).

gamma: (optional, multiplier for increasing the parameter L in fista, 1.5 by default) L0: (optional, initial parameter L in fista, 0.1 by default, should be small enough) fixed_step: (deactive the line search for L in fista and use L0 instead) compute_gram: (optional, pre-compute X^TX, false by default). intercept: (optional, do not regularize last row of W, false by default). ista: (optional, use ista instead of fista, false by default). subgrad: (optional, if not ista, use subradient descent instead of fista, false by default). a: b: (optional, if subgrad, the gradient step is a/(t+b)

also similar options as proximalTree

the function also implements the ADMM algorithm via an option admm=true. It is not documented and you need to look at the source code to use it.

delta: undocumented; modify at your own risks! c: undocumented; modify at your own risks! max_iter_backtracking: undocumented; modify at your own risks! lin_admm: undocumented; modify at your own risks! admm: undocumented; modify at your own risks! resetflow: undocumented; modify at your own risks! clever: undocumented; modify at your own risks! log: undocumented; modify at your own risks! logName: undocumented; modify at your own risks! is_inner_weights: undocumented; modify at your own risks! inner_weights: undocumented; modify at your own risks! sqrt_step: undocumented; modify at your own risks! size_group: undocumented; modify at your own risks! transpose: undocumented; modify at your own risks!

Returns: W: double dense p x n matrix or p x Nn matrix (for multi-logistic loss) optim: optional, double dense 4 x n matrix.

first row: values of the objective functions. third row: values of the relative duality gap (if available) fourth row: number of iterations

optim_info: vector of size 4, containing information of the optimization. W = spams.fistaGraph(Y,X,W0,graph,return_optim_info = False,…) (W,optim_info) = spams.fistaGraph(Y,X,W0,graph,return_optim_info = True,…)

Authors: Julien MAIRAL, 2010 (spams, matlab interface and documentation) Jean-Paul CHIEZE 2011-2012 (python interface)

Note:

Valid values for the regularization parameter (regul) are: "l0", "l1", "l2", "linf", "l2-not-squared", "elastic-net", "fused-lasso", "group-lasso-l2", "group-lasso-linf", "sparse-group-lasso-l2", "sparse-group-lasso-linf", "l1l2", "l1linf", "l1l2+l1", "l1linf+l1", "tree-l0", "tree-l2", "tree-linf", "graph", "graph-ridge", "graph-l2", "multi-task-tree", "multi-task-graph", "l1linf-row-column", "trace-norm", "trace-norm-vec", "rank", "rank-vec", "none"

CHAPTER

TWENTYTHREE

PROXIMALFLAT

spams.**proximalFlat**(*U*, *return_val_loss=False*, *numThreads=-1*, *lambda1=1.0*, *lambda2=0.0*, *lambda3=0.0*, *intercept=False*, *resetflow=False*, *regul=''*, *verbose=False*, *pos=False*, *clever=True*, *size_group=1*, *groups=None*, *transpose=False*)

proximalFlat computes proximal operators. Depending on the value of regul, it computes

Given an input matrix U=[u^1,ldots,u^n], it computes a matrix V=[v^1,ldots,v^n] such that if one chooses a regularization functions on vectors, it computes for each column u of U, a column v of V solving if regul='l0'

argmin 0.5||u-v||_2^2 + lambda1||v||_0

if regul='l1' argmin 0.5||u-v||_2^2 + lambda1||v||_1

if regul='l2' argmin 0.5||u-v||_2^2 + 0.5lambda1||v||_2^2

if regul='elastic-net' argmin 0.5||u-v||_2^2 + lambda1||v||_1 + lambda1_2||v||_2^2

if regul='fused-lasso'

argmin 0.5||u-v||_2^2 + lambda1 FL(v) + ... lambda1_2||v||_1 + lambda1_3||v||_2^2

if regul='linf' argmin 0.5||u-v||_2^2 + lambda1||v||_inf

if regul='l1-constraint' argmin 0.5||u-v||_2^2 s.t. ||v||_1 <= lambda1

if regul='l2-not-squared' argmin 0.5||u-v||_2^2 + lambda1||v||_2

if regul='group-lasso-l2' argmin 0.5||u-v||_2^2 + lambda1 sum_g ||v_g||_2 where the groups are either defined by groups or by size_group,

if regul='group-lasso-linf' argmin 0.5||u-v||_2^2 + lambda1 sum_g ||v_g||_inf

if regul='sparse-group-lasso-l2' argmin 0.5||u-v||_2^2 + lambda1 sum_g ||v_g||_2 + lambda1_2 ||v||_1 where the groups are either defined by groups or by size_group,

if regul='sparse-group-lasso-linf' argmin 0.5||u-v||_2^2 + lambda1 sum_g ||v_g||_inf + lambda1_2 ||v||_1

if regul='trace-norm-vec'

argmin 0.5||u-v||_2^2 + lambda1 ||mat(v)||_*

where mat(v) has size_group rows

if one chooses a regularization function on matrices if regul='l1l2', V=

argmin 0.5||U-V||_F^2 + lambda1||V||_{1/2}

if regul='l1linf', V= argmin 0.5||U-V||_F^2 + lambda1||V||_{1/inf}

if regul='l1l2+l1', V= argmin 0.5||U-V||_F^2 + lambda1||V||_{1/2} + lambda1_2||V||_{1/1}

if regul='l1linf+l1', V= argmin 0.5||U-V||_F^2 + lambda1||V||_{1/inf} + lambda1_2||V||_{1/1}

if regul='l1linf+row-column', V= argmin 0.5||U-V||_F^2 + lambda1||V||_{1/inf} + lambda1_2||V'||_{1/inf}

if regul='trace-norm', V= argmin 0.5||U-V||_F^2 + lambda1||V||_*

if regul='rank', V= argmin 0.5||U-V||_F^2 + lambda1 rank(V)

if regul='none', V= argmin 0.5||U-V||_F^2

for all these regularizations, it is possible to enforce non-negativity constraints with the option pos, and to prevent the last row of U to be regularized, with the option intercept

Args:

U: double m x n matrix (input signals) m is the signal size

return_val_loss: if true the function will return a tuple of matrices.

Kwargs: lambda1: (regularization parameter) regul: (choice of regularization, see above) lambda2: (optional, regularization parameter) lambda3: (optional, regularization parameter) verbose: (optional, verbosity level, false by default) intercept: (optional, last row of U is not regularized,

false by default)

transpose: (optional, transpose the matrix in the regularization function) size_group: (optional, for regularization functions assuming a group

structure). It is a scalar. When groups is not specified, it assumes that the groups are the sets of consecutive elements of size size_group

groups: (int32, optional, for regularization functions assuming a group structure. It is an int32 vector of size m containing the group indices of the variables (first group is 1).

pos: (optional, adds positivity constraints on the coefficients, false by default)

numThreads: (optional, number of threads for exploiting multi-core / multi-cpus. By default, it takes the value -1, which automatically selects all the available CPUs/cores).

resetflow: undocumented; modify at your own risks! clever: undocumented; modify at your own risks!

Returns: V: double m x n matrix (output coefficients) val_regularizer: double 1 x n vector (value of the regularization

term at the optimum).

val_loss: vector of size U.shape[1] alpha = spams.proximalFlat(U,return_val_loss = False,…) (alpha,val_loss) = spams.proximalFlat(U,return_val_loss = True,…)

Authors: Julien MAIRAL, 2010 (spams, matlab interface and documentation) Jean-Paul CHIEZE 2011-2012 (python interface)

Note:

Valid values for the regularization parameter (regul) are: "l0", "l1", "l2", "linf", "l2-not-squared", "elastic-net", "fused-lasso", "group-lasso-l2", "group-lasso-linf", "sparse-group-lasso-l2", "sparse-group-lasso-linf", "l1l2", "l1linf", "l1l2+l1", "l1linf+l1", "tree-l0", "tree-l2", "tree-linf", "graph", "graph-ridge", "graph-l2", "multi-task-tree", "multi-task-graph", "l1linf-row-column", "trace-norm", "trace-norm-vec", "rank", "rank-vec", "none"

CHAPTER

TWENTYFOUR

PROXIMALTREE

spams.**proximalTree**(*U*, *tree*, *return_val_loss=False*, *numThreads=-1*, *lambda1=1.0*, *lambda2=0.0*, *lambda3=0.0*, *intercept=False*, *resetflow=False*, *regul=”*, *verbose=False*, *pos=False*, *clever=True*, *size_group=1*, *transpose=False*)

proximalTree computes a proximal operator. Depending on the value of regul, it computes

Given an input matrix U=[u^1,\ldots,u^n], and a tree-structured set of groups T, it returns a matrix V=[v^1,\ldots,v^n]:

when the regularization function is for vectors, for every column u of U, it compute a column v of V solving if regul='tree-l0'

argmin 0.5||u-v||_2^2 + lambda1 sum_{g in T} delta^g(v)

if regul='tree-l2'

for all i, v^i = argmin 0.5||u-v||_2^2 + lambda1sum_{g in T} eta_g||v_g||_2

if regul='tree-linf'

for all i, v^i = argmin 0.5||u-v||_2^2 + lambda1sum_{g in T} eta_g||v_g||_inf

when the regularization function is for matrices: if regul='multi-task-tree'

V=argmin 0.5||U-V||_F^2 + lambda1 sum_{i=1}^nsum_{g in T} eta_g||v^i_g||_inf + ...
lambda1_2 sum_{g in T} eta_g max_{j in g}||V_j||_{inf}

it can also be used with any non-tree-structured regularization addressed by proximalFlat

for all these regularizations, it is possible to enforce non-negativity constraints with the option pos, and to prevent the last row of U to be regularized, with the option intercept

Args:

U: double m x n matrix (input signals) m is the signal size

tree: named list with four fields, eta_g, groups, own_variables and N_own_variables.

The tree structure requires a particular organization of groups and variables * Let us denote by N = |T|, the number of groups. the groups should be ordered T={g1,g2,\ldots,gN} such that if gi is included in gj, then j <= i. g1 should be the group at the root of the tree and contains every variable. * Every group is a set of contiguous indices for instance gi={3,4,5} or gi={4,5,6,7} or gi={4}, but not {3,5}; * We define root(gi) as the indices of the variables that are in gi, but not in its descendants. For instance for T={ g1={1,2,3,4},g2={2,3},g3={4} }, then, root(g1)={1}, root(g2)={2,3}, root(g3)={4}, We assume that for all i, root(gi) is a set of contigous variables * We assume that the smallest of root(gi) is also the smallest index of gi.

For instance, T={ g1={1,2,3,4},g2={2,3},g3={4} }, is a valid set of groups. but we can not have T={ g1={1,2,3,4},g2={1,2},g3={3} }, since root(g1)={4} and 4 is not the smallest element in g1.

We do not lose generality with these assumptions since they can be fullfilled for any tree-structured set of groups after a permutation of variables and a correct ordering of the groups. see more examples in test_ProximalTree.m of valid tree-structured sets of groups.

The first fields sets the weights for every group tree['eta_g'] double N vector

The next field sets inclusion relations between groups (but not between groups and variables): tree['groups'] sparse (double or boolean) N x N matrix the (i,j) entry is non-zero if and only if i is different than j and gi is included in gj. the first column corresponds to the group at the root of the tree.

The next field define the smallest index of each group gi, which is also the smallest index of root(gi) tree['own_variables'] int32 N vector

The next field define for each group gi, the size of root(gi) tree['N_own_variables'] int32 N vector

examples are given in test_ProximalTree.m

return_val_loss: if true the function will return a tuple of matrices.

Kwargs: lambda1: (regularization parameter) regul: (choice of regularization, see above) lambda2: (optional, regularization parameter) lambda3: (optional, regularization parameter) verbose: (optional, verbosity level, false by default) intercept: (optional, last row of U is not regularized,

false by default)

pos: (optional, adds positivity constraints on the coefficients, false by default)

transpose: (optional, transpose the matrix in the regularization function) size_group: (optional, for regularization functions assuming a group

structure). It is a scalar. When groups is not specified, it assumes that the groups are the sets of consecutive elements of size size_group

numThreads: (optional, number of threads for exploiting multi-core / multi-cpus. By default, it takes the value -1, which automatically selects all the available CPUs/cores).

resetflow: undocumented; modify at your own risks! clever: undocumented; modify at your own risks!

Returns: V: double m x n matrix (output coefficients) val_regularizer: double 1 x n vector (value of the regularization

term at the optimum).

val_loss: vector of size U.shape[1] alpha = spams.proximalTree(U,tree,return_val_loss = False,…) (alpha,val_loss) = spams.proximalTree(U,tree,return_val_loss = True,…)

Authors: Julien MAIRAL, 2010 (spams, matlab interface and documentation) Jean-Paul CHIEZE 2011-2012 (python interface)

Note:

Valid values for the regularization parameter (regul) are: "l0", "l1", "l2", "linf", "l2-not-squared", "elastic-net", "fused-lasso", "group-lasso-l2", "group-lasso-linf", "sparse-group-lasso-l2", "sparse-group-lasso-linf", "l1l2", "l1linf", "l1l2+l1", "l1linf+l1", "tree-l0", "tree-l2", "tree-linf", "graph", "graph-ridge", "graph-l2", "multi-task-tree", "multi-task-graph", "l1linf-row-column", "trace-norm", "trace-norm-vec", "rank", "rank-vec", "none"

CHAPTER

TWENTYFIVE

PROXIMALGRAPH

spams.**proximalGraph**(*U*, *graph*, *return_val_loss=False*, *numThreads=-1*, *lambda1=1.0*, *lambda2=0.0*, *lambda3=0.0*, *intercept=False*, *resetflow=False*, *regul=”*, *verbose=False*, *pos=False*, *clever=True*, *eval=None*, *size_group=1*, *transpose=False*)

proximalGraph computes a proximal operator. Depending on the value of regul, it computes

Given an input matrix U=[u^1,ldots,u^n], and a set of groups G, it computes a matrix V=[v^1,ldots,v^n] such that

if regul='graph' for every column u of U, it computes a column v of V solving

argmin 0.5||u-v||_2^2 + lambda1sum_{g in G} eta_g||v_g||_inf

if regul='graph+ridge' for every column u of U, it computes a column v of V solving

argmin 0.5||u-v||_2^2 + lambda1sum_{g in G} eta_g||v_g||_inf + lambda1_2||v||_2^2

if regul='multi-task-graph'

V=argmin 0.5||U-V||_F^2 + lambda1 sum_{i=1}^nsum_{g in G} eta_g||v^i_g||_inf + ...
lambda1_2 sum_{g in G} eta_g max_{j in g}||V_j||_{inf}

it can also be used with any regularization addressed by proximalFlat

for all these regularizations, it is possible to enforce non-negativity constraints with the option pos, and to prevent the last row of U to be regularized, with the option intercept

Args:

U: double p x n matrix (input signals) m is the signal size

graph: struct with three fields, eta_g, groups, and groups_var

The first fields sets the weights for every group graph.eta_g double N vector

The next field sets inclusion relations between groups (but not between groups and variables): graph.groups sparse (double or boolean) N x N matrix the (i,j) entry is non-zero if and only if i is different than j and gi is included in gj.

The next field sets inclusion relations between groups and variables graph.groups_var sparse (double or boolean) p x N matrix the (i,j) entry is non-zero if and only if the variable i is included in gj, but not in any children of gj.

examples are given in test_ProximalGraph.m

return_val_loss: if true the function will return a tuple of matrices.

Kwargs: lambda1: (regularization parameter) regul: (choice of regularization, see above) lambda2: (optional, regularization parameter) lambda3: (optional, regularization parameter) verbose: (optional, verbosity level, false by default) intercept: (optional, last row of U is not regularized,

false by default)

pos: (optional, adds positivity constraints on the coefficients, false by default)

numThreads: (optional, number of threads for exploiting multi-core / multi-cpus. By default, it takes the value -1, which automatically selects all the available CPUs/cores).

resetflow: undocumented; modify at your own risks! clever: undocumented; modify at your own risks! size_group: undocumented; modify at your own risks! transpose: undocumented; modify at your own risks!

Returns: V: double p x n matrix (output coefficients) val_regularizer: double 1 x n vector (value of the regularization

term at the optimum).

val_loss: vector of size U.shape[1] alpha = spams.proximalGraph(U,graph,return_val_loss = False,…) (alpha,val_loss) = spams.proximalGraph(U,graph,return_val_loss = True,…)

Authors: Julien MAIRAL, 2010 (spams, matlab interface and documentation) Jean-Paul CHIEZE 2011-2012 (python interface)

Note:

Valid values for the regularization parameter (regul) are: "l0", "l1", "l2", "linf", "l2-not-squared", "elastic-net", "fused-lasso", "group-lasso-l2", "group-lasso-linf", "sparse-group-lasso-l2", "sparse-group-lasso-linf", "l1l2", "l1linf", "l1l2+l1", "l1linf+l1", "tree-l0", "tree-l2", "tree-linf", "graph", "graph-ridge", "graph-l2", "multi-task-tree", "multi-task-graph", "l1linf-row-column", "trace-norm", "trace-norm-vec", "rank", "rank-vec", "none"

CHAPTER

TWENTYSIX

TRAINDL

`spams.trainDL`(*X*, *return_model=False*, *model=None*, *D=None*, *numThreads=-1*, *batchsize=-1*, *K=-1*, *lambda1=None*, *lambda2=1e-09*, *iter=-1*, *t0=1e-05*, *mode=2*, *posAlpha=False*, *posD=False*, *expand=False*, *modeD=0*, *whiten=False*, *clean=True*, *verbose=True*, *gamma1=0.0*, *gamma2=0.0*, *rho=1.0*, *iter_updateD=None*, *stochastic_deprecated=False*, *modeParam=0*, *batch=False*, *log_deprecated=False*, *logName=''*)

trainDL is an efficient implementation of the dictionary learning technique presented in

"Online Learning for Matrix Factorization and Sparse Coding" by Julien Mairal, Francis Bach, Jean Ponce and Guillermo Sapiro arXiv:0908.0050

"Online Dictionary Learning for Sparse Coding" by Julien Mairal, Francis Bach, Jean Ponce and Guillermo Sapiro ICML 2009.

Note that if you use mode=1 or 2, if the training set has a reasonable size and you have enough memory on your computer, you should use trainDL_Memory instead.

It addresses the dictionary learning problems

1. if mode=0

min_{D in C} (1/n) sum_{i=1}^n (1/2)||x_i-Dalpha_i||_2^2 s.t. ...

||alpha_i||_1 <= lambda1

2. if mode=1

min_{D in C} (1/n) sum_{i=1}^n ||alpha_i||_1 s.t. ...

||x_i-Dalpha_i||_2^2 <= lambda1

3. if mode=2

min_{D in C} (1/n) sum_{i=1}^n (1/2)||x_i-Dalpha_i||_2^2 + ...

lambda1||alpha_i||_1 + lambda1_2||alpha_i||_2^2

4. if mode=3, the sparse coding is done with OMP

min_{D in C} (1/n) sum_{i=1}^n (1/2)||x_i-Dalpha_i||_2^2 s.t. ...

||alpha_i||_0 <= lambda1

5. if mode=4, the sparse coding is done with OMP

min_{D in C} (1/n) sum_{i=1}^n ||alpha_i||_0 s.t. ...

||x_i-Dalpha_i||_2^2 <= lambda1

6. if mode=5, the sparse coding is done with OMP

min_{D in C} (1/n) sum_{i=1}^n 0.5||x_i-Dalpha_i||_2^2 +lambda1||alpha_i||_0

C is a convex set verifying

1. if modeD=0 C={ D in Real^{m x p} s.t. forall j, ||d_j||_2^2 <= 1 }
2. if modeD=1 C={ D in Real^{m x p} s.t. forall j, ||d_j||_2^2 + ...

   gamma1||d_j||_1 <= 1 }
3. if modeD=2 C={ D in Real^{m x p} s.t. forall j, ||d_j||_2^2 + ...

   gamma1||d_j||_1 + gamma2 FL(d_j) <= 1 }
4. if modeD=3 C={ D in Real^{m x p} s.t. forall j, (1-gamma1)||d_j||_2^2 + ...

   gamma1||d_j||_1 <= 1 }

Potentially, n can be very large with this algorithm.

Args:

X: double m x n matrix (input signals) m is the signal size n is the number of signals to decompose

return_model: if true the function will return the model as a named list ('A' = A, 'B' = B, 'iter' = n)

model: None or model (as A,B,iter) to use as initialisation

Kwargs:

D: (optional) double m x p matrix (dictionary) p is the number of elements in the dictionary When D is not provided, the dictionary is initialized with random elements from the training set.

K: (size of the dictionary, optional is D is provided) lambda1: (parameter) lambda2: (optional, by default 0) iter: (number of iterations). If a negative number is

provided it will perform the computation during the corresponding number of seconds. For instance iter=-5 learns the dictionary during 5 seconds.

mode: (optional, see above, by default 2) posAlpha: (optional, adds positivity constraints on the

coefficients, false by default, not compatible with mode =3,4)

modeD: (optional, see above, by default 0) posD: (optional, adds positivity constraints on the

dictionary, false by default, not compatible with modeD=2)

gamma1: (optional parameter for modeD >= 1) gamma2: (optional parameter for modeD = 2) batchsize: (optional, size of the minibatch, by default

512.

iter_updateD: (optional, number of BCD iterations for the dictionary update step, by default 1)

modeParam: (optimization mode). 1) if modeParam=0, the optimization uses the parameter free strategy of the ICML paper 2) if modeParam=1, the optimization uses the parameters rho as in arXiv:0908.0050 3) if modeParam=2, the optimization uses exponential decay weights with updates of the form A_{t} <- rho A_{t-1} + alpha_t alpha_t^T

rho: (optional) tuning parameter (see paper arXiv:0908.0050) t0: (optional) tuning parameter (see paper arXiv:0908.0050) clean: (optional, true by default. prunes automatically the dictionary from unused elements).

verbose: (optional, true by default, increase verbosity) numThreads: (optional, number of threads for exploiting

multi-core / multi-cpus. By default, it takes the value -1, which automatically selects all the available CPUs/cores).

expand: undocumented; modify at your own risks! whiten: undocumented; modify at your own risks! stochastic_deprecated: undocumented; modify at your own risks! batch: undocumented; modify at your own risks! log_deprecated: undocumented; modify at your own risks! logName: undocumented; modify at your own risks!

Returns: D: double m x p matrix (dictionary) model: the model as A B iter

D = spams.trainDL(X,return_model = False,. . .) (D,model) = spams.trainDL(X,return_model = True,. . .)

Authors: Julien MAIRAL, 2009 (spams, matlab interface and documentation) Jean-Paul CHIEZE 2011-2012 (python interface)

Note: this function admits a few experimental usages, which have not been extensively tested:

- single precision setting

CHAPTER

TWENTYSEVEN

STRUCTTRAINDL

spams.**structTrainDL**(*X*, *return_model=False*, *model=None*, *D=None*, *graph=None*, *tree=None*, *numThreads=-1*, *tol=1e-06*, *fixed_step=True*, *ista=False*, *batchsize=-1*, *K=-1*, *lambda1=None*, *lambda2=1e-09*, *lambda3=0.0*, *iter=-1*, *t0=1e-05*, *regul='none'*, *posAlpha=False*, *posD=False*, *expand=False*, *modeD=0*, *whiten=False*, *clean=True*, *verbose=True*, *gamma1=0.0*, *gamma2=0.0*, *rho=1.0*, *iter_updateD=None*, *stochastic_deprecated=False*, *modeParam=0*, *batch=False*, *log_deprecated=False*, *logName="*)

structTrainDL is an efficient implementation of the dictionary learning technique presented in

"Online Learning for Matrix Factorization and Sparse Coding" by Julien Mairal, Francis Bach, Jean Ponce and Guillermo Sapiro arXiv:0908.0050

"Online Dictionary Learning for Sparse Coding" by Julien Mairal, Francis Bach, Jean Ponce and Guillermo Sapiro ICML 2009.

It addresses the dictionary learning problems min_{D in C} (1/n) sum_{i=1}^n 0.5||x_i-Dalpha_i||_2^2 + lambda1 psi(alpha) where the regularization function psi depends on regul (see proximalFlat for the description of psi,

and regul below for allowed values of regul)

C is a convex set verifying

1. if modeD=0 C={ D in Real^{m x p} s.t. forall j, ||d_j||_2^2 <= 1 }
2. if modeD=1 C={ D in Real^{m x p} s.t. forall j, ||d_j||_2^2 + ...

   gamma1||d_j||_1 <= 1 }
3. if modeD=2 C={ D in Real^{m x p} s.t. forall j, ||d_j||_2^2 + ...

   gamma1||d_j||_1 + gamma2 FL(d_j) <= 1 }
4. if modeD=3 C={ D in Real^{m x p} s.t. forall j, (1-gamma1)||d_j||_2^2 + ...

   gamma1||d_j||_1 <= 1 }

Potentially, n can be very large with this algorithm.

Args:

X: double m x n matrix (input signals) m is the signal size n is the number of signals to decompose

return_model: if true the function will return the model as a named list ('A' = A, 'B' = B, 'iter' = n)

model: None or model (as A,B,iter) to use as initialisation

Kwargs:

D: (optional) double m x p matrix (dictionary) p is the number of elements in the dictionary When D is not provided, the dictionary is initialized with random elements from the training set.

K: (size of the dictionary, optional is D is provided) lambda1: (parameter) lambda2: (optional, by default 0) lambda3: (optional, regularization parameter, 0 by default) iter: (number of iterations). If a negative number is

provided it will perform the computation during the corresponding number of seconds. For instance iter=-5 learns the dictionary during 5 seconds.

regul: choice of regularization [one of] ‘l0’ ‘l1’ ‘l2’ ‘linf’ ‘none’ ‘elastic-net’ ‘fused-lasso’ ‘graph’ ‘graph-ridge’ ‘graph-l2’ ‘tree-l0’ ‘tree-l2’ ‘tree-linf’

tree: struct (see documentation of proximalTree); needed for regul of graph kind.

graph: struct (see documentation of proximalGraph); needed for regul of tree kind.

posAlpha: (optional, adds positivity constraints on the coefficients, false by default.

modeD: (optional, see above, by default 0) posD: (optional, adds positivity constraints on the

dictionary, false by default, not compatible with modeD=2)

gamma1: (optional parameter for modeD >= 1) gamma2: (optional parameter for modeD = 2) batchsize: (optional, size of the minibatch, by default

512.

iter_updateD: (optional, number of BCD iterations for the dictionary update step, by default 1)

modeParam: (optimization mode). 1) if modeParam=0, the optimization uses the parameter free strategy of the ICML paper 2) if modeParam=1, the optimization uses the parameters rho as in arXiv:0908.0050 3) if modeParam=2, the optimization uses exponential decay weights with updates of the form A_{t} <- rho A_{t-1} + alpha_t alpha_t^T

ista: (optional, use ista instead of fista, false by default). tol: (optional, tolerance for stopping criteration, which is a relative duality gap fixed_step: (deactive the line search for L in fista and use K instead) rho: (optional) tuning parameter (see paper arXiv:0908.0050) t0: (optional) tuning parameter (see paper arXiv:0908.0050) clean: (optional, true by default. prunes

automatically the dictionary from unused elements).

verbose: (optional, true by default, increase verbosity) numThreads: (optional, number of threads for exploiting

multi-core / multi-cpus. By default, it takes the value -1, which automatically selects all the available CPUs/cores).

expand: undocumented; modify at your own risks! whiten: undocumented; modify at your own risks! stochastic_deprecated: undocumented; modify at your own risks! batch: undocumented; modify at your own risks! log_deprecated: undocumented; modify at your own risks! logName: undocumented; modify at your own risks!

Returns: D: double m x p matrix (dictionary) model: the model as A B iter

D = spams.structTrainDL(X,return_model = False,...) (D,model) = spams.structTrainDL(X,return_model = True,...)

Authors: Julien MAIRAL, 2009 (spams, matlab interface and documentation) Jean-Paul CHIEZE 2011-2012 (python interface)

Note: this function admits a few experimental usages, which have not been extensively tested:

- single precision setting

CHAPTER TWENTYEIGHT

TRAINDL_MEMORY

spams.**trainDL_Memory**(*X*, *D=None*, *numThreads=-1*, *batchsize=-1*, *K=-1*, *lambda1=None*, *iter=-1*, *t0=1e-05*, *mode=2*, *posD=False*, *expand=False*, *modeD=0*, *whiten=False*, *clean=True*, *gamma1=0.0*, *gamma2=0.0*, *rho=1.0*, *iter_updateD=1*, *stochastic_deprecated=False*, *modeParam=0*, *batch=False*, *log_deprecated=False*, *logName="*)

trainDL_Memory is an efficient but memory consuming variant of the dictionary learning technique presented in

"Online Learning for Matrix Factorization and Sparse Coding" by Julien Mairal, Francis Bach, Jean Ponce and Guillermo Sapiro arXiv:0908.0050

"Online Dictionary Learning for Sparse Coding" by Julien Mairal, Francis Bach, Jean Ponce and Guillermo Sapiro ICML 2009.

Contrary to the approaches above, the algorithm here does require to store all the coefficients from all the training signals. For this reason this variant can not be used with large training sets, but is more efficient than the regular online approach for training sets of reasonable size.

It addresses the dictionary learning problems

1. if mode=1

min_{D in C} (1/n) sum_{i=1}^n ||alpha_i||_1 s.t. . . .

||x_i-Dalpha_i||_2^2 <= lambda1

2. if mode=2

min_{D in C} (1/n) sum_{i=1}^n (1/2)||x_i-Dalpha_i||_2^2 + . . . lambda1||alpha_i||_1

C is a convex set verifying

1. if modeD=0 C={ D in Real^{m x p} s.t. forall j, ||d_j||_2^2 <= 1 }

1. if modeD=1 C={ D in Real^{m x p} s.t. forall j, ||d_j||_2^2 + . . .

   gamma1||d_j||_1 <= 1 }

1. if modeD=2 C={ D in Real^{m x p} s.t. forall j, ||d_j||_2^2 + . . .

   gamma1||d_j||_1 + gamma2 FL(d_j) <= 1 }

Potentially, n can be very large with this algorithm.

Args:

X: double m x n matrix (input signals) m is the signal size n is the number of signals to decompose

Kwargs:

D: (optional) double m x p matrix (dictionary) p is the number of elements in the dictionary When D is not provided, the dictionary is initialized with random elements from the training set.

K: (size of the dictionary, optional is D is provided) lambda1: (parameter) iter: (number of iterations). If a negative number is

provided it will perform the computation during the corresponding number of seconds. For instance iter=-5 learns the dictionary during 5 seconds.

mode: (optional, see above, by default 2) modeD: (optional, see above, by default 0) posD: (optional, adds positivity constraints on the

dictionary, false by default, not compatible with modeD=2)

gamma1: (optional parameter for modeD >= 1) gamma2: (optional parameter for modeD = 2) batchsize: (optional, size of the minibatch, by default

512.

iter_updateD: (optional, number of BCD iterations for the dictionary update step, by default 1)

modeParam: (optimization mode). 1) if modeParam=0, the optimization uses the parameter free strategy of the ICML paper 2) if modeParam=1, the optimization uses the parameters rho as in arXiv:0908.0050 3) if modeParam=2, the optimization uses exponential decay weights with updates of the form A_{t} <- rho A_{t-1} + alpha_t alpha_t^T

rho: (optional) tuning parameter (see paper arXiv:0908.0050) t0: (optional) tuning parameter (see paper arXiv:0908.0050) clean: (optional, true by default. prunes

automatically the dictionary from unused elements).

numThreads: (optional, number of threads for exploiting multi-core / multi-cpus. By default, it takes the value -1, which automatically selects all the available CPUs/cores).

expand: undocumented; modify at your own risks! whiten: undocumented; modify at your own risks! stochastic_deprecated: undocumented; modify at your own risks! batch: undocumented; modify at your own risks! log_deprecated: undocumented; modify at your own risks! logName: undocumented; modify at your own risks!

Returns: D: double m x p matrix (dictionary) model: the model as A B iter

D = spams.trainDL_Memory(X,. . .)

Authors: Julien MAIRAL, 2009 (spams, matlab interface and documentation) Jean-Paul CHIEZE 2011-2012 (python interface)

Note: this function admits a few experimental usages, which have not been extensively tested:

- single precision setting (even though the output alpha is double precision)

CHAPTER

TWENTYNINE

NMF

spams.**nmf**(*X*, *return_lasso=False*, *model=None*, *numThreads=-1*, *batchsize=-1*, *K=-1*, *iter=-1*, *t0=1e-05*, *clean=True*, *rho=1.0*, *modeParam=0*, *batch=False*)

trainDL is an efficient implementation of the non-negative matrix factorization technique presented in

"Online Learning for Matrix Factorization and Sparse Coding" by Julien Mairal, Francis Bach, Jean Ponce and Guillermo Sapiro arXiv:0908.0050

"Online Dictionary Learning for Sparse Coding" by Julien Mairal, Francis Bach, Jean Ponce and Guillermo Sapiro ICML 2009.

Potentially, n can be very large with this algorithm.

Args:

X: double m x n matrix (input signals) m is the signal size n is the number of signals to decompose

return_lasso: if true the function will return a tuple of matrices.

Kwargs: K: (number of required factors) iter: (number of iterations). If a negative number

is provided it will perform the computation during the corresponding number of seconds. For instance iter=-5 learns the dictionary during 5 seconds.

batchsize: (optional, size of the minibatch, by default

512.

modeParam: (optimization mode). 1) if modeParam=0, the optimization uses the parameter free strategy of the ICML paper 2) if modeParam=1, the optimization uses the parameters rho as in arXiv:0908.0050 3) if modeParam=2, the optimization uses exponential decay weights with updates of the form A_{t} <- rho A_{t-1} + alpha_t alpha_t^T

rho: (optional) tuning parameter (see paper arXiv:0908.0050)

t0: (optional) tuning parameter (see paper arXiv:0908.0050)

clean: (optional, true by default. prunes automatically the dictionary from unused elements).

batch: (optional, false by default, use batch learning instead of online learning)

numThreads: (optional, number of threads for exploiting multi-core / multi-cpus. By default, it takes the value -1, which automatically selects all the available CPUs/cores).

model: struct (optional) learned model for "retraining" the data.

Returns: U: double m x p matrix V: double p x n matrix (optional) model: struct (optional) learned model to be used for "retraining" the data. U = spams.nmf(X,return_lasso = False,...) (U,V) = spams.nmf(X,return_lasso = True,...)

Authors: Julien MAIRAL, 2009 (spams, matlab interface and documentation) Jean-Paul CHIEZE 2011-2012 (python interface)

CHAPTER

THIRTY

NNSC

spams.**nnsc**(*X*, *return_lasso=False*, *model=None*, *lambda1=None*, *numThreads=-1*, *batchsize=-1*, *K=-1*, *iter=-1*, *t0=1e-05*, *clean=True*, *rho=1.0*, *modeParam=0*, *batch=False*)

trainDL is an efficient implementation of the non-negative sparse coding technique presented in

"Online Learning for Matrix Factorization and Sparse Coding" by Julien Mairal, Francis Bach, Jean Ponce and Guillermo Sapiro arXiv:0908.0050

"Online Dictionary Learning for Sparse Coding" by Julien Mairal, Francis Bach, Jean Ponce and Guillermo Sapiro ICML 2009.

Potentially, n can be very large with this algorithm.

Args:

X: double m x n matrix (input signals) m is the signal size n is the number of signals to decompose

return_lasso: if true the function will return a tuple of matrices.

Kwargs: K: (number of required factors) lambda1: (parameter) iter: (number of iterations). If a negative number

is provided it will perform the computation during the corresponding number of seconds. For instance iter=-5 learns the dictionary during 5 seconds.

batchsize: (optional, size of the minibatch, by default

512.

modeParam: (optimization mode). 1) if modeParam=0, the optimization uses the parameter free strategy of the ICML paper 2) if modeParam=1, the optimization uses the parameters rho as in arXiv:0908.0050 3) if modeParam=2, the optimization uses exponential decay weights with updates of the form A_{t} <- rho A_{t-1} + alpha_t alpha_t^T

rho: (optional) tuning parameter (see paper arXiv:0908.0050)

t0: (optional) tuning parameter (see paper arXiv:0908.0050)

clean: (optional, true by default. prunes automatically the dictionary from unused elements).

batch: (optional, false by default, use batch learning instead of online learning)

numThreads: (optional, number of threads for exploiting multi-core / multi-cpus. By default, it takes the value -1, which automatically selects all the available CPUs/cores).

model: struct (optional) learned model for "retraining" the data.

Returns: U: double m x p matrix V: double p x n matrix (optional) model: struct (optional) learned model to be used for "retraining" the data. U = spams.nnsc(X,return_lasso = False,. . .) (U,V) = spams.nnsc(X,return_lasso = True,. . .)

Authors: Julien MAIRAL, 2009 (spams, matlab interface and documentation) Jean-Paul CHIEZE 2011-2012 (python interface)

CHAPTER

THIRTYONE

ARCHETYPALANALYSIS

`spams.`**`archetypalAnalysis`**(*X*, *p=10*, *Z0=None*, *returnAB=False*, *robust=False*, *epsilon=0.001*, *computeXtX=False*, *stepsFISTA=3*, *stepsAS=50*, *randominit=False*, *numThreads=-1*)

documentation to appear soon

Args:

X: double m x n matrix (input signals) m is the signal size n is the number of signals to decompose

Returns: Z: double %

Authors: Yuansi Chen and Julien MAIRAL, 2014 (spams, matlab interface and documentation) Jean-Paul CHIEZE 2011-2012 (python interface)

CHAPTER

THIRTYTWO

DECOMPSIMPLEX

`spams.decompSimplex(X, Z, computeXtX=False, numThreads=-1)`

documentation to appear soon

Args:

X: double m x n matrix (input signals) m is the signal size n is the number of signals to decompose

Returns: Z: double %

Authors: Yuansi Chen and Julien MAIRAL, 2014 (spams, matlab interface and documentation) Jean-Paul CHIEZE 2011-2012 (python interface)

CHAPTER

THIRTYTHREE

SIMPLEGROUPTREE

`spams.`**`simpleGroupTree`**`(`*degrees*`)`

makes a structure representing a tree given the

degree of each level.

Args: degrees: int vector; degrees(i) is the number of children of each node at level i

Returns:

group_struct: list, one element for each node an element is itsel a 4 elements list : nodeid (int >= 0), weight (double), array of vars attached to the node (here equal to [nodeid]), array of children (nodeid's)

Authors: Jean-Paul CHIEZE, 2012

CHAPTER

THIRTYFOUR

READGROUPSTRUCT

`spams.`**`readGroupStruct`**(*file*)

reads a text file describing "simply" the structure of groups

of variables needed by proximalGraph, proximalTree, fistaGraph, fistaTree and structTrainDL and builds the corresponding group structure.

weight is a float variables-list : a space separated list of integers, maybe empty,

but '[' and '] must be present. Numbers in the range (0 - Nv-1)

children-list [a space separated list of node-id's] If the list is empty, '->' may be omitted.

The data must obey some rules []

- A group contains the variables of the corresponding node and of the whole subtree.
- Variables attached to a node are those that are not int the subtree.
- **If the data destination is a Graph, there may be several independant trees,** and a varibale may appear in several trees.

If the destination is a Tree, there must be only one tree, the root node must have id == 0 and each variable must appear only once.

Args: file: the file name

Returns:

groups: list, one element for each node an element is itsel a 4 elements list: nodeid (int >= 0), weight (double), array of vars of the node, array of children (nodeid's)

Authors: Jean-Paul CHIEZE, 2012

CHAPTER

THIRTYFIVE

GROUPSTRUCTOFSTRING

spams.**groupStructOfString**(*s*)

decode a multi-line string describing "simply" the structure of groups

of variables needed by proximalGraph, proximalTree, fistaGraph, fistaTree and structTrainDL and builds the corresponding group structure.

Each line describes a group of variables as a node of a tree. It has up to 4 fields separated by spaces:

node-id node-weight [variables-list] -> children-list

Let's define Ng = number of groups, and Nv = number of variables. node-id must be in the range (0 - Ng-1), and there must be Ng nodes weight is a float variables-list : a space separated list of integers, maybe empty,

but '[' and '] must be present. Numbers in the range (0 - Nv-1)

children-list [a space separated list of node-id's] If the list is empty, '->' may be omitted.

The data must obey some rules []

- A group contains the variables of the corresponding node and of the whole subtree.
- Variables attached to a node are those that are not int the subtree.
- If the data destination is a Graph, there may be several independant trees, and a varibale may appear in several trees.

If the destination is a Tree, there must be only one tree, the root node must have id == 0 and each variable must appear only once.

Args: s: the multi-lines string

Returns:

groups: list, one element for each node an element is itsel a 4 elements list: nodeid (int >= 0), weight (double), array of vars of the node, array of children (nodeid's)

Authors: Jean-Paul CHIEZE, 2012

CHAPTER

THIRTYSIX

GRAPHOFGROUPSTRUCT

`spams.`**`graphOfGroupStruct`**`(`*`gstruct`*`)`

converts a group structure into the graph structure

used by proximalGraph, fistaGraph or structTrainDL

Args:

gstruct: the structure of groups as a list, one element per node an element is itself a 4 elements list: nodeid (>= 0), weight (double), array of vars associated to the node, array of children (nodeis's)

Returns: graph: struct (see documentation of proximalGraph)

Authors: Jean-Paul CHIEZE, 2012

CHAPTER

THIRTYSEVEN

TREEOFGROUPSTRUCT

`spams.`**`treeOfGroupStruct`**(*gstruct*)

converts a group structure into the tree structure

used by proximalTree, fistaTree or structTrainDL

Args:

gstruct: the structure of groups as a list, one element per node an element is itself a 4 lements list: nodeid (>= 0), weight (double), array of vars associated to the node, array of children (nodeis's)

Returns:

permutations: permutation vector that must be applied to the result of the programm using the tree. Empty if no permutation is needed.

tree: named list (see documentation of proximalTree) nbvars: number of variables in the tree

(permutations,tree,nbvars) = spams.treeOfGroupStruct(gstruct)

Authors: Jean-Paul CHIEZE, 2012

CHAPTER

THIRTYEIGHT

INDICES AND TABLES

- genindex
- modindex
- search

INDEX