SLIM

1.0

Generated by Doxygen 1.8.2

Wed Dec 19 2012 12:19:48

Contents

Chapter 1

Welcome to SLIM

SLIM is a library which implements the Sparse LInear Methods (SLIM) for top-n recommendation. The algorithm is described in the paper

Xia Ning and George Karypis, "SLIM: Sparse Linear Models for Top-N Recommender Systems", Proceedings of the 2011 IEEE 11th International Conference on Data Mining, 497–506.

This manual is divided in the following sections:

- Download
- Installation
- Running SLIM
- Input Files
- Output Files
- Examples
- Credits & Contact Information
- Copyright Information

1.1 Download

SLIM is an open-source software and also provided as a binary distribution with pre-built executables for Linux (64 bit architecture). Additional binaries can be provided upon request. The source code can be downloaded here.

`slim-1.0.tar.gz` Linux (x86_64)

A pdf version of the manual is available `here`

1.2 Installation

Once you download SLIM, you need to uncompress and untar it using the following commands:

```
> tar -xzf slim-1.0.tar.gz
```

This will create a directory named `slim-1.0` with the following structure:

```
slim-1.0\
  build\
  examples\
  include\
  src\
```

In order to compile the source code and build the SLIM library, it requires CMake 2.8 (http://www.cmake.org/) and gcc 4.4. Assumming CMake and gcc are installed, do the following commands to compile and build:

```
> cd slim-1.0
> cd build
> cmake ..
> make
> make install
```

And if you want to clean all the objects generated from `make`, do the following command:

```
> make clean
```

After you do the above commands, a `libSLIM.a` library will be generated within `build/lib` directory, all the *.h files are in `build/include` directory, and two executables `slim_learn` and `slim_predict` will be generated within `build/examples` directory. You can use `slim_learn` and `slim_predict` as stand-alone programs, or you can use the library by properly linking it and including the header files.

1.3 Running SLIM

The name of the SLIM executable is `slim_learn` and `slim_predict` and they are located under `build/examples`. The `slim_learn` and slim_predict programs are invoked at the command-line within a shell window (e.g., Gnome terminal, etc).

1.3.1 Manpage

The manpage for `SLIM` is the following (can be obtained by typing `slim_learn -help`):

```
Usage
slim_learn [options]

-train_file=string
Specifies the input file which contains the training data. This file should be
in .csr format.

-test_file=string
Specifies the input file which contains the testing data. This file should be
  in .csr format.

-model_file=string
Specifies the output file which will contains a model matrix. The output file will be in
  .csr format.

-fs_file=string
Specifies the input file which contains a matrix for feature selection purpose. This input
                file should be in .csr format. This option takes effect only when -fs option is specified.

-pred_file=string
Specifies the output file which will contain the top-n prediction for each user.  The output
                file wil be in .csr format. If this option is not specified, no prediction scores will be

-lambda=float
Specifies the regularization parameter for the $\ell_1$ norm

-beta=flat
```

```
    Specifies the regularizationi parameter for the $\ell_2$ norm

    -starti=int
    Specifies the index of the first column (C-style indexing) from
    which the sparse coefficient matrix will be calculated. The default
      value is 0.

    -endi=int
    Specifies the index of the last column (exclusively) up to which
      the sparse coefficient matrix will be calculated. The default value
      is the number of total columns.

    -transpose
    Specifies that the input feature selection matrix needs to be transposed.

    -fs
      Specifies that feature selection is required so as to accelerate the learning.

    -k=int
    Specifies the number of features if feature selection is applied. The default
    value is 50.

    -dbglvl=int
    Specifies the debug level. The default value is 0.

    -optTol=float
    Specifies the threshold which control the optimization. Once the error
      from two optimization iterations is smaller than this value, the optimization
      process will be terminated. The default value is 1e-5.

    -max_bcls_niters=int
    Specifies the maximum number of iterations that is allowed for optimization.
      Once the number of iterations reaches this value, the optimization process
      will be terminated. The default value is 1e5.

    -bsize=int
    Specifies the block size for output. Once the calculation for these bsize
    blocks are done, they are dumped into the output file. The default value is 1000.

    -nratings=int
    Specifies the number of unique rating values in the testing set. The rating values
    should be integers starting from 1. The default value is 1.

    -topn=int
    Specifies the number of recommendations to be produced for each user. The default
                  value is 10.

    -help
    Print this message.
```

1.4 Input Files

The `slim_learn` and `slim_predict` accept and produce a sparse matrix format (with extension .csr) which is specified as follows.

A sparse matrix A with n rows and m columns is stored in a plain text file that contains n lines, where the n lines contain information for each row of A. In SLIM's sparse matrix format only the non-zero entries of the matrix are stored. In particular, the i-st line of the file contains information about the non-zero entries of the i-th row of the matrix. The non-zero entries of each row are specified as a space-separated list of pairs. Each pair contains the column number followed by the value for that particular column. The column numbers are assumed to be integers and their corresponding values are assumed to be binary. Note that the columns are numbered starting from 1 (not from 0 as is often done in C). An example of SLIM's matrix format is shown as follows. This shows an example 7 × 8 matrix and its corresponding representation in SLIM's matrix format.

```
    matrix:
```

```
0 1 0 0 1 0 0 1
1 1 0 1 0 0 0 0
0 0 1 0 0 1 0 1
1 0 0 0 0 0 0 0
0 1 0 1 0 0 1 0
0 0 1 0 1 1 0 0
0 1 0 1 1 0 1 1

matrix .csr file
2 1 5 1 8 1
1 1 2 1 4 1
3 1 6 1 8 1
1 1
2 1 4 1 7 1
3 1 5 1 6 1
2 1 4 1 5 1 7 1 8 1
```

1.5 Output Files

The `slim_learn` generates a model file which will be in .csr format as specified above, and the contained matrix is actually the transpose the aggregation coefficient matrix.

The `slim_predict` generates a prediction file, if specified by `-pred_file`, in .csr format. In this file, each row corresponds to a testing user, the column values correspond to the items that have bean recommended, and the corresponding values are the recommendation scores. All the column values are order based on the scores in descreasing order.

1.6 Examples

The following shows how to run `slim_learn`

```
slim_learn -train_file=train.mat -model_file=model.mat -starti=0 -endi=1682 -lambda=2 -beta=5 -optTol=0.00001 -max_bcls_niters=10000
```

The model is printed into `model.mat`.

Note

The matrix output to `model.mat` is the transpose the sparse aggregation coefficient matrix.

The following shows how to run `slim_predict`

```
slim_predict -train_file=train.mat -test_file=test.mat -model_file=model.mat -pred_file=prediction.txt -topn=10
```

Note

If `model.mat` contains the tranpose of an aggregation coefficient matrix or an item-item similarity matrix, the option `-transpose` needs to be specified.

1.6.1 Running SLIM in parallel

You can run `slim_learn` to calculate only a chunck (i.e., a certain set of consecutive columns, specified by `-starti` and `-endi`) of the aggregation coefficent matrix. In this way, you can run multiple slim_learn programs in parallel (e.g., on a hadoop cluster) to calculate different chunks of the aggreegation coefficient matrix concurrently and then collect all the output and concatenate them in the right order so as to get the entire aggregation coefficient matrix.

1.7 Credits & Contact Information

SLIM was written by Xia Ning.

Thank Prof. Michael P. Friedlander for providing the BCLS library.

Thank Prof. George Karypis for providing the GKlib library.

If you encounter any problems or have any suggestions, please contact Xia Ning via email at xning@cs.umn.edu.

1.8 Copyright Information

Chapter 2

Data Structure Index

2.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 3

File Index

3.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 4

Data Structure Documentation

4.1 cs_sparse Struct Reference

A matrix structure used for BCLS. This is adopted from BCLS.

```
#include <struct.h>
```

Data Fields

- int nzmax
- int m
- int n
- int ∗ p
- int ∗ i
- float ∗ x
- int nz

4.1.1 Detailed Description

A matrix structure used for BCLS. This is adopted from BCLS.

Definition at line 99 of file struct.h.

4.1.2 Field Documentation

4.1.2.1 int∗ cs_sparse::i

row indices, size nzmax

Definition at line 110 of file struct.h.

Referenced by Aprod(), cs_spalloc(), cs_spfree(), and slim_learn().

4.1.2.2 int cs_sparse::m

number of rows

Definition at line 104 of file struct.h.

Referenced by cs_spalloc(), and slim_learn().

4.1.2.3 int cs_sparse::n

number of columns

Definition at line 106 of file struct.h.

Referenced by cs_spalloc(), and slim_learn().

4.1.2.4 int cs_sparse::nz

number of entries in triplet matrix, -1 for compressed-col

Definition at line 114 of file struct.h.

Referenced by cs_spalloc(), and slim_learn().

4.1.2.5 int cs_sparse::nzmax

maximum number of entries

Definition at line 102 of file struct.h.

Referenced by cs_spalloc(), and slim_learn().

4.1.2.6 int* cs_sparse::p

column pointers (size n+1) or col indices (size nzmax)

Definition at line 108 of file struct.h.

Referenced by Aprod(), cs_spalloc(), cs_spfree(), and slim_learn().

4.1.2.7 float* cs_sparse::x

numerical values, size nzmax

Definition at line 112 of file struct.h.

Referenced by Aprod(), cs_spalloc(), cs_spfree(), and slim_learn().

The documentation for this struct was generated from the following file:

- /home/xning/Project/SLIMLib/include/struct.h

4.2 ctimer_t Struct Reference

A data structure for timer.

```
#include <struct.h>
```

Data Fields

- clock_t start
- clock_t end

4.2.1 Detailed Description

A data structure for timer.

Definition at line 19 of file struct.h.

4.2.2 Field Documentation

4.2.2.1 clock_t ctimer_t::end

The end time

Definition at line 24 of file struct.h.

Referenced by display_timer(), and end_timer().

4.2.2.2 clock_t ctimer_t::start

The start time

Definition at line 22 of file struct.h.

Referenced by display_timer(), and start_timer().

The documentation for this struct was generated from the following file:

- /home/xning/Project/SLIMLib/include/struct.h

4.3 ctrl_t Struct Reference

A data structure for ctrl parameters.

```
#include <struct.h>
```

Data Fields

- char ∗ train_file
- char ∗ test_file
- char ∗ model_file
- char ∗ pred_file
- int dbglvl
- double lambda
- double beta
- int starti
- int endi
- double optTol
- int max_bcls_niters
- double bl
- double bu
- int fs
- char ∗ fs_file
- int k
- int bsize
- int nratings
- int topn
- int transpose

4.3.1 Detailed Description

A data structure for ctrl parameters.

Definition at line 35 of file struct.h.

4.3.2 Field Documentation

4.3.2.1 double ctrl_t::beta

the regularization parameter for L-2 norm

Definition at line 52 of file struct.h.

Referenced by create_ctrl(), parse_cmdline(), and slim_learn().

4.3.2.2 double ctrl_t::bl

lower bound for BCLS

Definition at line 65 of file struct.h.

Referenced by create_ctrl(), and slim_learn().

4.3.2.3 int ctrl_t::bsize

block size for data dump

Definition at line 78 of file struct.h.

Referenced by create_ctrl(), parse_cmdline(), and slim_learn().

4.3.2.4 double ctrl_t::bu

upper bound for BCLS

Definition at line 67 of file struct.h.

Referenced by create_ctrl(), and slim_learn().

4.3.2.5 int ctrl_t::dbglvl

debug level, default 0

Definition at line 47 of file struct.h.

Referenced by bcsol(), create_ctrl(), parse_cmdline(), preprocess(), and slim_test().

4.3.2.6 int ctrl_t::endi

the ending column index from which the coefficient matrix is calculated

Definition at line 57 of file struct.h.

Referenced by create_ctrl(), parse_cmdline(), and slim_learn().

4.3.2.7 int ctrl_t::fs

if feature selection is applied

Definition at line 70 of file struct.h.

Referenced by create_ctrl(), parse_cmdline(), and slim_learn().

4.3.2.8 char∗ ctrl_t::fs_file

a file name which contains a constraint matrix in csr format for feature selection

Definition at line 72 of file struct.h.

Referenced by create_ctrl(), free_ctrl(), parse_cmdline(), and slim_learn().

4.3.2.9 int ctrl_t::k

number of features to use if feature selection is applied

Definition at line 75 of file struct.h.

Referenced by create_ctrl(), parse_cmdline(), and slim_fs_learn().

4.3.2.10 double ctrl_t::lambda

the regularization parameter for L-1 norm

Definition at line 50 of file struct.h.

Referenced by create_ctrl(), parse_cmdline(), and slim_learn().

4.3.2.11 int ctrl_t::max_bcls_niters

max number of iterations allowed in BCLS solver

Definition at line 62 of file struct.h.

Referenced by bcsol(), create_ctrl(), parse_cmdline(), and slim_learn().

4.3.2.12 char∗ ctrl_t::model_file

a file name into which the model in csr format will be output

Definition at line 42 of file struct.h.

Referenced by create_ctrl(), free_ctrl(), main(), parse_cmdline(), and slim_learn().

4.3.2.13 int ctrl_t::nratings

number of ratings

Definition at line 81 of file struct.h.

Referenced by create_ctrl(), parse_cmdline(), slim_predict(), and slim_test().

4.3.2.14 double ctrl_t::optTol

optimality tolerance

Definition at line 60 of file struct.h.

Referenced by bcsol(), create_ctrl(), and parse_cmdline().

4.3.2.15 char∗ ctrl_t::pred_file

a file name into which the prediction will be output

Definition at line 44 of file struct.h.

Referenced by create_ctrl(), free_ctrl(), parse_cmdline(), and slim_test().

4.3.2.16 int ctrl_t::starti

the starting column index from which the coefficient matrix is calculated

Definition at line 55 of file struct.h.

Referenced by create_ctrl(), parse_cmdline(), and slim_learn().

4.3.2.17 char∗ ctrl_t::test_file

a file name that contains the testing data in csr format

Definition at line 40 of file struct.h.

Referenced by create_ctrl(), free_ctrl(), main(), and parse_cmdline().

4.3.2.18 int ctrl_t::topn

the number of recommendations to be recommended

Definition at line 84 of file struct.h.

Referenced by create_ctrl(), parse_cmdline(), and suggest_predict().

4.3.2.19 char∗ ctrl_t::train_file

a file name that contains the training data in csr format

Definition at line 38 of file struct.h.

Referenced by create_ctrl(), free_ctrl(), main(), and parse_cmdline().

4.3.2.20 int ctrl_t::transpose

need to transpose the matrix

Definition at line 87 of file struct.h.

Referenced by create_ctrl(), parse_cmdline(), and read_constraint().

The documentation for this struct was generated from the following file:

- /home/xning/Project/SLIMLib/include/struct.h

4.4 worksp Struct Reference

A workspace structure used for BCLS. This is adopated from BCLS.

```
#include <struct.h>
```

Data Fields

- cs ∗ A
- int max_bcls_niters
- int ∗ acol

4.4.1 Detailed Description

A workspace structure used for BCLS. This is adopated from BCLS.

Definition at line 124 of file struct.h.

4.4.2 Field Documentation

4.4.2.1 cs∗ worksp::A

a matrix

Definition at line 127 of file struct.h.

Referenced by Aprod(), and slim_learn().

4.4.2.2 int∗ worksp::acol

the active columns

Definition at line 131 of file struct.h.

Referenced by Aprod(), slim_fs_learn(), and slim_learn().

4.4.2.3 int worksp::max_bcls_niters

max number of iterations allowed in BCLS solver

Definition at line 129 of file struct.h.

Referenced by slim_learn().

The documentation for this struct was generated from the following file:

- /home/xning/Project/SLIMLib/include/struct.h

Chapter 5

File Documentation

5.1 /home/xning/Project/SLIMLib/examples/test_slim_learn.c File Reference

This is to test slim_learn.

```
#include <slim.h>
```

Functions

- int main (int argc, char ∗argv[ ])

  The main entry for the learning.

5.1.1 Detailed Description

This is to test slim_learn.

Author

Xia Ning

Version

1.0

Date

2011-2012

Copyright

GNU Public License

Definition in file test_slim_learn.c.

5.2 /home/xning/Project/SLIMLib/examples/test_slim_predict.c File Reference

This is to test slim_predict.

```
#include <slim.h>
```

Functions

- int main (int argc, char ∗argv[ ])

  The main entry for the testing.

5.2.1 Detailed Description

This is to test slim_predict.

Author

Xia Ning

Version

1.0

Date

2011-2012

Copyright

GNU Public License

Definition in file test_slim_predict.c.

5.3 /home/xning/Project/SLIMLib/include/def.h File Reference

This file contains all the defined macros.

Macros

- #define CMD_TRAIN_FILE 1
- #define CMD_TEST_FILE 2
- #define CMD_MODEL_FILE 3
- #define CMD_DBGLVL 4
- #define CMD_LAMBDA 5
- #define CMD_BETA 6
- #define CMD_STARTI 7
- #define CMD_ENDI 8
- #define CMD_OPTTOL 9
- #define CMD_MAX_BCLS_NITERS 10
- #define CMD_FS_FILE 11
- #define CMD_FS 12
- #define CMD_K 13
- #define CMD_BSIZE 14
- #define CMD_HELP 15
- #define CMD_NRATINGS 16
- #define CMD_PRED_FILE 17
- #define CMD_TOPN 18
- #define CMD_TRANSPOSE 19
- #define EPSILON (1e-5)
- #define EPSILON2 (1e-10)

5.3.1 Detailed Description

This file contains all the defined macros.

Definition in file def.h.

5.3.2 Macro Definition Documentation

5.3.2.1 #define CMD_BETA 6

beta

Definition at line 22 of file def.h.

Referenced by parse_cmdline().

5.3.2.2 #define CMD_BSIZE 14

block size

Definition at line 38 of file def.h.

Referenced by parse_cmdline().

5.3.2.3 #define CMD_DBGLVL 4

debug level

Definition at line 18 of file def.h.

Referenced by parse_cmdline().

5.3.2.4 #define CMD_ENDI 8

endi

Definition at line 26 of file def.h.

Referenced by parse_cmdline().

5.3.2.5 #define CMD_FS 12

feature selection

Definition at line 34 of file def.h.

Referenced by parse_cmdline().

5.3.2.6 #define CMD_FS_FILE 11

feature selection constrain file

Definition at line 32 of file def.h.

Referenced by parse_cmdline().

5.3.2.7 #define CMD_HELP 15

help

Definition at line 40 of file def.h.

Referenced by parse_cmdline().

5.3.2.8 #define CMD_K 13

number of features

Definition at line 36 of file def.h.

Referenced by parse_cmdline().

5.3.2.9 #define CMD_LAMBDA 5

lambda

Definition at line 20 of file def.h.

Referenced by parse_cmdline().

5.3.2.10 #define CMD_MAX_BCLS_NITERS 10

max BCLS iterations

Definition at line 30 of file def.h.

Referenced by parse_cmdline().

5.3.2.11 #define CMD_MODEL_FILE 3

model file

Definition at line 16 of file def.h.

Referenced by parse_cmdline().

5.3.2.12 #define CMD_NRATINGS 16

nratings

Definition at line 42 of file def.h.

Referenced by parse_cmdline().

5.3.2.13 #define CMD_OPTTOL 9

opttol

Definition at line 28 of file def.h.

Referenced by parse_cmdline().

5.3.2.14 #define CMD_PRED_FILE 17

predition file

Definition at line 44 of file def.h.

Referenced by parse_cmdline().

5.3.2.15 #define CMD_STARTI 7

starti

Definition at line 24 of file def.h.

Referenced by parse_cmdline().

5.3.2.16 #define CMD_TEST_FILE 2

test file

Definition at line 14 of file def.h.

Referenced by parse_cmdline().

5.3.2.17 #define CMD_TOPN 18

number of recommendations

Definition at line 46 of file def.h.

Referenced by parse_cmdline().

5.3.2.18 #define CMD_TRAIN_FILE 1

train file

Definition at line 12 of file def.h.

Referenced by parse_cmdline().

5.3.2.19 #define CMD_TRANSPOSE 19

transpose matrix

Definition at line 48 of file def.h.

Referenced by parse_cmdline().

5.3.2.20 #define EPSILON (1e-5)

epsilon

Definition at line 52 of file def.h.

Referenced by slim_learn().

5.3.2.21 #define EPSILON2 (1e-10)

epsilon2

Definition at line 54 of file def.h.

Referenced by count_nnz().

5.4 /home/xning/Project/SLIMLib/include/proto.h File Reference

This file contains all prototypes.

Functions

- void parse_cmdline (ctrl_t ∗ctrl, int argc, char ∗argv[])

  Entry point of the command-line argument parsing.
- ctrl_t ∗ create_ctrl ()

  Create a ctrl structure wich contains all the default parameters for SLIM.
- void free_ctrl (ctrl_t ∗ctrl)

  Free a ctrl structure.
- void start_timer (ctimer_t ∗ctimer)

  Start a timer to record current time.
- void end_timer (ctimer_t ∗ctimer)

  End a timer to record a length of a duration.
- void display_timer (ctimer_t ∗ctimer, char ∗msg)

  Display a user-defined message and a duration length recorded by a timer.
- int count_nnz (double ∗array, int narray)

  Count the number of non-zero values in an array.
- void find_topk (double ∗w, int n, int topk, double ∗map, int ∗topk2)

  Find the top-k values from an array.
- void get_column (gk_csr_t ∗constraint, int i, double ∗w)

  Get a column from a csr matrix.
- gk_csr_t ∗ read_constraint (ctrl_t ∗ctrl, char ∗file)

  Read in a constraint matrix for feature selection.
- void csr_Write (gk_csr_t ∗mat, char ∗filename, char ∗mode, int format, int writevals, int numbering)

  Dump the csr into a file.
- void check_train_test (ctrl_t ∗ctrl, gk_csr_t ∗train, gk_csr_t ∗test)

  Check if test data are already in train data.
- int call_back (BCLS ∗ls, void ∗UsrWrk)

  call_back function, periodically called by BCLS to test if the user wants to exit. This is from BCLS.
- int call_back_it (BCLS ∗ls, void ∗UsrWrk)

  call_back function, immediately terminate BCLS iterations based on how many iterations it runs
- int pretty_printer (void ∗io_file, char ∗msg)

  Pretty_printer, this is the print-routine that will be used by BCLS for its output. This is from BCLS.
- void ∗ cs_free (void ∗p)

  Wrapper for free.
- void ∗ cs_malloc (int n, size_t size)

  Wrapper for malloc.
- void ∗ cs_calloc (int n, size_t size)

  Wrapper for calloc.
- cs ∗ cs_spfree (cs ∗A)

  Free a sparse matrix.
- cs ∗ cs_spalloc (int m, int n, int nzmax, int values, int triplet)

  Allocate a sparse matrix (triplet form or compressed-column form)
- void dload (const int n, const double alpha, double x[])

  Load a constant into a vector.
- int Aprod (int mode, int m, int n, int nix, int ix[], double x[], double y[], void ∗UsrWrk)

  Aprod, matrix-vector products. This is from BCLS.
- void bcsol (ctrl_t ∗ctrl, gk_csr_t ∗AA, double ∗bb, double ∗x, worksp ∗Wrk, double ∗bl, double ∗bu, double beta, double ∗c)

  BCLS learning. This is from BCLS.
- void slim_learn (ctrl_t ∗ctrl, gk_csr_t ∗train)

  SLIM learning.

- void slim_fs_learn (ctrl_t ∗ctrl, gk_csr_t ∗A, double ∗b, double ∗w, float ∗∗A_colval, worksp ∗Wrk, double ∗bl, double ∗bu, double beta, double ∗c)

    SLIM learning with feature selction.
- void preprocess (ctrl_t ∗ctrl, gk_csr_t ∗train, gk_csr_t ∗test)

    Pre-process the data.
- double ∗ slim_test (ctrl_t ∗ctrl, gk_csr_t ∗model, gk_csr_t ∗train, gk_csr_t ∗test)

    Top-N recommendations and evaluations.
- int suggest_predict (ctrl_t ∗ctrl, gk_csr_t ∗model, int ∗∗iidx, gk_csr_t ∗train, int u, gk_dkv_t ∗∗rcmd)

    Top-N recommendation for a user.
- void slim_predict (ctrl_t ∗ctrl, gk_csr_t ∗train, gk_csr_t ∗test, gk_csr_t ∗model)

    SLIM testing.

5.4.1 Detailed Description

This file contains all prototypes.

Definition in file proto.h.

5.4.2 Function Documentation

5.4.2.1 int Aprod (int *mode,* int *m,* int *n,* int *nix,* int *ix[ ],* double *x[ ],* double *y[ ],* void ∗ *UsrWrk*)

Aprod, matrix-vector products. This is from BCLS.

If mode == BCLS_PROD_A (0), compute y <- A ∗x, with x untouched; and if mode == BCLS_PROD_At (1), compute x <- A'∗y, with y untouched.

Definition at line 163 of file bcsol.c.

References worksp::A, worksp::acol, cs_sparse::i, cs_sparse::p, and cs_sparse::x.

Referenced by bcsol().

```
                                             {

int      i, j, k, l;
double  aij;
double xj, sum;
worksp * Wrk = (worksp *)UsrWrk;
cs *A = (cs *)Wrk->A;
int * Ai = A->i;
int * Ap = A->p;
float * Ax = A->x;
int * acol = Wrk->acol;


if (mode == BCLS_PROD_A) {

  gk_dset(m, 0.0, y);

  for (l = 0; l < nix; l++) {
    j = ix[l];

    /* skip the inactive column */
    if (!acol[j]) continue;

    xj = x[j];
    if (xj == 0.0)
  ; // Relax.
    else
  for (k = Ap[j]; k < Ap[j+1]; k++) {
    aij    = Ax[k];


    /* this is to handle float-valued A matrix */
    i      = Ai[k];
    y[i] += aij * xj;


  }
```

```
    }
  }

  else if (mode == BCLS_PROD_At) {
    for (l = 0; l < nix; l++) {
      j = ix[l];
      sum = 0;

      /* skip the inactive column */
      if (!acol[j]){
    x[j] = sum;
    continue;
      }

      for (k = Ap[j]; k < Ap[j+1]; k++) {
    aij   = Ax[k];

    /* this is to handle float-valued A matrix */
    i     = Ai[k];
    sum += aij * y[i];


      }
      x[j] = sum;
    }
  }


  return 0;
}
```

5.4.2.2 void bcsol (ctrl_t * *ctrl*, gk_csr_t * *AA*, double * *bb*, double * *x*, worksp * *Wrk*, double * *bl*, double * *bu*, double *beta*, double * *c*)

BCLS learning. This is from BCLS.

This is to solve the problem

$$\begin{array}{ll} \underset{x}{\text{minimize}} & \frac{1}{2}\|Ax - a_i\|_2^2 + \frac{1}{2}\beta\|x\|_2^2 + \lambda\|x\|_1 \\ \text{subject to} & 0 \le x \\ & x_i = 0 \end{array}$$

Definition at line 247 of file bcsol.c.

References Aprod(), bcls_niters, call_back(), call_back_it(), ctrl_t::dbglvl, ctrl_t::max_bcls_niters, ctrl_t::optTol, and pretty_printer().

Referenced by slim_fs_learn(), and slim_learn().

```
                                {

  bcls_niters = 0;

  /*  Problem dimensions. */
  int m = AA->nrows;
  int n = AA->ncols;

  /* init a bcls problem */
  BCLS    *ls = bcls_create_prob( m, n );

  bcls_set_problem_data(ls, m, n, Aprod,  Wrk, beta, x, bb, c, bl, bu);


  /* set up tolerance */
  ls->optTol = ctrl->optTol;

  /* whatever */
  bcls_set_print_hook( ls, stdout, pretty_printer );
  ls->proj_search     = proj_search;
  ls->newton_step     = newton_step;
  /* if (ctrl->test_bcls) */
  if (ctrl->max_bcls_niters > 0)
    ls->CallBack       = call_back_it;
  else
    ls->CallBack       = call_back;

  /* call the solver */
  int err = bcls_solve_prob( ls );
  /* solution */
  if (ctrl->dbglvl > 1){
    int nnzx = 0;
    printf("\n Solution\n --------\n");
    printf("%4s  %18s %1s %18s %1s %18s  %18s\n",
       "Var","Lower","","Value","","Upper","Gradient");
    for (int j = 0; j < n; j++) {
      if (x[j] > 1e-10){
    nnzx ++;
    char * blActiv = "";
    char * buActiv = "";
    if (x[j] - bl[j] < ls->epsx) blActiv = "=";
    if (bu[j] - x[j] < ls->epsx) buActiv = "=";
    printf("%4d  %18.11e %1s %18.11e %1s %18.11e  %18.11e\n",
           j+1, bl[j], blActiv, x[j], buActiv, bu[j], (ls->g)[j]);
      }
    }
    printf("%d nnz solution values\n", nnzx);
  }


  /* free the problem */
  err = bcls_free_prob( ls );

}
```

5.4.2.3 void check_train_test (ctrl_t ∗ *ctrl*, gk_csr_t ∗ *train*, gk_csr_t ∗ *test*)

Check if test data are already in train data.

Parameters

in	*ctrl*	A ctrl structure
in	*train*	The training data
in	*test*	The testing data

Definition at line 19 of file check.c.

Referenced by preprocess().

```
                                                                       {

  int error = 0;

  int nrows_test = test->nrows;

  for (int i = 0; i < nrows_test; i ++){

    int nc_test = test->rowptr[i+1] - test->rowptr[i];
    int nc_train = train->rowptr[i+1] - train->rowptr[i];

    for (int j = 0; j < nc_test; j ++){

      int item_test = *(test->rowptr[i] + j + test->rowind);

      for (int k = 0; k < nc_train; k ++){

    int item_train = *(train->rowptr[i] + k + train->rowind);

    if (item_test == item_train){
      printf("ERROR: user %6d has item %6d in both train and test\n", i,
      item_train);
      error = 1;
      break;
    }

      }

    }

  }

  if (error)
    errexit("ERROR: train and test not disjoint\n");

}
```

5.4.2.4 int count_nnz (double ∗ *array,* int *narray*)

Count the number of non-zero values in an array.

Parameters

in	*array*	An array whose non-zero values will be counted
in	*narray*	The length of the array

Returns

int The number of non-zero values in the array

Definition at line 134 of file util.c.

References EPSILON2.

Referenced by slim_fs_learn().

```
                                          {

  int nnz = 0;

  for (int i = 0; i < narray; i ++){
    if (array[i] > EPSILON2 || array[i] < -EPSILON2)
      nnz ++;
  }

  return nnz;

}
```

5.4.2.5 ctrl_t∗ create_ctrl ( )

Create a ctrl structure wich contains all the default parameters for SLIM.

Returns

ctrl_t∗ A pointer to a created ctrl structure

Definition at line 21 of file util.c.

References ctrl_t::beta, ctrl_t::bl, ctrl_t::bsize, ctrl_t::bu, ctrl_t::dbglvl, ctrl_t::endi, ctrl_t::fs, ctrl_t::fs_file, ctrl_t::k, ctrl_t::lambda, ctrl_t::max_bcls_niters, ctrl_t::model_file, ctrl_t::nratings, ctrl_t::optTol, ctrl_t::pred_file, ctrl_t::starti, ctrl_t::test_file, ctrl_t::topn, ctrl_t::train_file, and ctrl_t::transpose.

Referenced by main(), and parse_cmdline().

```
                      {

  ctrl_t * ctrl = gk_malloc(sizeof(ctrl_t), "malloc ctrl");

  ctrl->train_file = NULL;
  ctrl->test_file  = NULL;
  ctrl->model_file = NULL;
  ctrl->fs_file = NULL;
  ctrl->pred_file = NULL;

  ctrl->dbglvl = 0;

  ctrl->beta    = 1.0;
  ctrl->lambda  = 1.0;

  ctrl->starti = -1;
  ctrl->endi = -1;

  ctrl->optTol = 1e-5;
  ctrl->max_bcls_niters = 100000;

  ctrl->bl = 0;
  ctrl->bu = 1e20;

  ctrl->fs = 0;

  ctrl->k = 50;

  ctrl->bsize = 1000;

  ctrl->nratings = 5;

  ctrl->topn = 10;

  ctrl->transpose = 0;

  return ctrl;

}
```

5.4.2.6 void display_timer (ctimer_t ∗ *ctimer,* char ∗ *msg*)

Display a user-defined message and a duration length recorded by a timer.

Parameters

in	*ctimer*	A timer with a length of a duration recorded
in	*msg*	A user-defined message to display

Definition at line 116 of file util.c.

References ctimer_t::end, and ctimer_t::start.

Referenced by slim_learn(), and slim_test().

```
                                                    {

  printf("----- elapsed CPU time for %s: %f s\n", msg,
         (double)(ctimer->end - ctimer->start)/CLOCKS_PER_SEC);
  fflush(stdout);

}
```

5.4.2.7 void end_timer (ctimer_t ∗ *ctimer*)

End a timer to record a length of a duration.

Parameters

in	*ctimer*	A timer to end

Definition at line 101 of file util.c.

References ctimer_t::end.

Referenced by slim_learn(), and slim_test().

```
                                     {

  ctimer->end = clock();

}
```

5.4.2.8 void find_topk (double ∗ *w,* int *n,* int *topk,* double ∗ *map,* int ∗ *topk2*)

Find the top-k values from an array.

Parameters

in	*w*	The array whose top-k values will be found
in	*n*	The length of the array w
in	*topk*	The number of top values to be found
out	*map*	The array of indices that correspond to the top-k values in the input array
out	*topk2*	The actual number of top values that are found

Definition at line 159 of file util.c.

Referenced by slim_fs_learn().

```
                                                                                        {

  gk_dkv_t * wkv = gk_malloc(sizeof(gk_dkv_t)*n, "malloc wkv");
  int k2 = 0;

  for (int i = 0; i < n; i ++){
    wkv[i].key = w[i];
    wkv[i].val = i;
    if (w[i] > 1e-10) k2 ++;
  }

  /* sort */
  gk_dkvsortd(n, wkv);

  for (int i = 0; i < ((topk <= k2)? topk:k2); i ++){
    map[i] = wkv[i].val;
  }

  *topk2 = ((topk <= k2)? topk:k2);
  gk_free((void **)&wkv, LTERM);

}
```

5.4.2.9 void free_ctrl (ctrl_t * *ctrl*)

Free a ctrl structure.

Parameters

in	*ctrl*	A pointer to a ctrl structure to be freed

Definition at line 68 of file util.c.

References ctrl_t::fs_file, ctrl_t::model_file, ctrl_t::pred_file, ctrl_t::test_file, and ctrl_t::train_file.

Referenced by main().

```
                             {

  gk_free((void **)&ctrl->model_file, LTERM);
  gk_free((void **)&ctrl->train_file, LTERM);
  gk_free((void **)&ctrl->test_file, LTERM);
  gk_free((void **)&ctrl->pred_file, LTERM);
  gk_free((void **)&ctrl->fs_file, LTERM);

  gk_free((void **)&ctrl, LTERM);

}
```

5.4.2.10 void get_column (gk_csr_t * *constraint,* int *i,* double * *w*)

Get a column from a csr matrix.

Parameters

in	*constraint*	A matrix from which one column is to be retrievd
in	*i*	The index of the column to be retrieved
out	*w*	The output vector which saves the retrieved column

Definition at line 194 of file util.c.

Referenced by slim_learn().

```
                                                               {
  if (i > constraint->ncols){
    gk_dset(constraint->nrows, 0, w);
  }
  else{
    int nnz = constraint->colptr[i+1] - constraint->colptr[i];
    for (int j = 0; j < nnz; j ++){
      int k = *(constraint->colptr[i] + j + constraint->colind);
      w[k]   = *(constraint->colptr[i] + j + constraint->colval);
    }
  }
}
```

5.4.2.11 void parse_cmdline (ctrl_t * *ctrl*, int *argc*, char * *argv[]*)

Entry point of the command-line argument parsing.

Parameters

out	*ctrl*	A ctrl structure to be filled out
in	*argc*	Number of arguments
in	*argv*	A list of arguments

Definition at line 146 of file cmd.c.

References ctrl_t::beta, ctrl_t::bsize, CMD_BETA, CMD_BSIZE, CMD_DBGLVL, CMD_ENDI, CMD_FS, CMD_FS_FILE, CMD_HELP, CMD_K, CMD_LAMBDA, CMD_MAX_BCLS_NITERS, CMD_MODEL_FILE, CMD_NRATINGS, CMD_OPTTOL, CMD_PRED_FILE, CMD_STARTI, CMD_TEST_FILE, CMD_TOPN, CMD_TRAIN_FILE, CMD_TRANSPOSE, create_ctrl(), ctrl_t::dbglvl, ctrl_t::endi, ctrl_t::fs, ctrl_t::fs_file, ctrl_t::k, ctrl_t::lambda, ctrl_t::max_bcls_niters, ctrl_t::model_file, ctrl_t::nratings, ctrl_t::optTol, ctrl_t::pred_file, ctrl_t::starti, ctrl_t::test_file, ctrl_t::topn, ctrl_t::train_file, and ctrl_t::transpose.

Referenced by main().

```
                                                               {
  int c = -1, option_index = -1;

  if (ctrl == NULL)
    ctrl = create_ctrl();

  while((c = gk_getopt_long_only(argc, argv, "", slim_options, &option_index))
        != -1){
    switch(c){


    case CMD_TRAIN_FILE:
      ctrl->train_file = gk_strdup(gk_optarg);
      break;

    case CMD_TEST_FILE:
      ctrl->test_file = gk_strdup(gk_optarg);
      break;

    case CMD_MODEL_FILE:
      ctrl->model_file = gk_strdup(gk_optarg);
      break;

    case CMD_PRED_FILE:
      ctrl->pred_file = gk_strdup(gk_optarg);
      break;
    case CMD_DBGLVL:
      ctrl->dbglvl = atoi(gk_optarg);
      break;

    case CMD_LAMBDA:
      ctrl->lambda = atof(gk_optarg);
      break;

    case CMD_BETA:
      ctrl->beta = atof(gk_optarg);
      break;

    case CMD_STARTI:
      ctrl->starti = atoi(gk_optarg);
      break;

    case CMD_ENDI:
      ctrl->endi = atoi(gk_optarg);
      break;

    case CMD_OPTTOL:
      ctrl->optTol = atof(gk_optarg);
      break;

    case CMD_MAX_BCLS_NITERS:
      ctrl->max_bcls_niters = atoi(gk_optarg);
      break;

    case CMD_FS:
      ctrl->fs = 1;
      break;

    case CMD_FS_FILE:
      ctrl->fs_file = gk_strdup(gk_optarg);
      break;

    case CMD_K:
      ctrl->k = atoi(gk_optarg);
      break;

    case CMD_BSIZE:
      ctrl->bsize = atoi(gk_optarg);
      break;

    case CMD_NRATINGS:
      ctrl->nratings = atoi(gk_optarg);
      break;

    case CMD_TOPN:
      ctrl->topn = atoi(gk_optarg);
      break;

    case CMD_TRANSPOSE:
      ctrl->transpose = 1;
      break;

    case CMD_HELP:
      for (int i=0; strlen(helpstr[i]) > 0; i++)
    printf("%s\n", helpstr[i]);
      exit(0);


    case '?':
    default:
      printf("Illegal command-line option(s) %s\n", gk_optarg);
      exit(0);

    }
  }


  if (argc-gk_optind != 0 || argc == 1) {
    for (int i=0; strlen(shorthelpstr[i]) > 0; i++)
      printf("%s\n", shorthelpstr[i]);
    exit(0);
  }


}
```

5.4.2.12 void slim_fs_learn (ctrl_t * *ctrl*, gk_csr_t * *A*, double * *b*, double * *w*, float ** *A_colval*, worksp * *Wrk*, double * *bl*, double * *bu*, double *beta*, double * *c*)

SLIM learning with feature selction.

Parameters

in	*ctrl*	A ctrl structure which contains all the parameters for SLIM Learning with feature selection
in	*A*	The A matrix
in	*b*	The RHS vector
in,out	*w*	The solution vector
in	*A_colval*	A temporary place for a column
in	*Wrk*	A workspace for BCLS
in	*bl*	The lower bound for BCLS
in	*bu*	The upper bound for BCLS
in	*beta*	The regularization parameter for L-2 norm
in	*c*	The vector for L-1 norm

Definition at line 31 of file slim_fs_learn.c.

References worksp::acol, bcsol(), count_nnz(), find_topk(), and ctrl_t::k.

Referenced by slim_learn().

```
                                   {
  int nnz = *(A->colptr + A->ncols);
  int * acol = Wrk->acol;

  /* count nnz */
  int kk = count_nnz(w, A->ncols);

  /* find topk nnz */
  int topk = 0;
  /* find the indices of topk entries, meanwhile w is over-written as the topk
       locations */
  find_topk(w, A->ncols, gk_min(ctrl->k, kk), w, &topk);

  /* back up original values, this is done only once */
  if (*A_colval == NULL){
    *A_colval = gk_malloc(sizeof(float)*nnz, "malloc *A_colval");
    memcpy((void *)*A_colval, (void *)A->colval, sizeof(float)*nnz);
  }

  /* remove all A nnz values, this will not affect the column under
       consideration */
  gk_fset(nnz, 0, A->colval);
  /* set all columns as inactive */
  gk_iset(A->ncols, 0, acol);
  /* recover all topk columns in A */
  for (int i = 0; i < topk; i ++){
    int j = (int)w[i];
    /* activate this column */
    acol[j] = 1;
    int nj = A->colptr[j+1] - A->colptr[j];
    for (int k = 0; k < nj; k ++){
      /* get the orignal values back */
      *(A->colptr[j] + k + A->colval) = *(A->colptr[j] + k + *A_colval);
    }
  }

  /* BCLS */
  gk_dset(A->ncols, 0, w);
  bcsol(ctrl, A, b, w, Wrk, bl, bu, beta, c);

  /* recover full A, specific to binary A, this will over-write the column of
       b,
     but will not matter */
  memcpy((void *)A->colval, (void *)*A_colval, sizeof(float)*nnz);
  /* activate all columns */
  gk_iset(A->ncols, 1, acol);

}
```

5.4.2.13 void slim_learn (ctrl_t * *ctrl,* gk_csr_t * *train*)

SLIM learning.

This routine contains the learning algorithm for SLIM

Parameters

in	*ctrl*	A ctrl structure which contains all the parameters for SLIM learning
in	*train*	The training data

Definition at line 22 of file slim_learn.c.

References worksp::A, worksp::acol, bcsol(), ctrl_t::beta, ctrl_t::bl, ctrl_t::bsize, ctrl_t::bu, csr_Write(), display_timer(), end_timer(), ctrl_t::endi, EPSILON, ctrl_t::fs, ctrl_t::fs_file, get_column(), cs_sparse::i, ctrl_t::lambda, cs_sparse::m, ctrl_t::max_bcls_niters, worksp::max_bcls_niters, ctrl_t::model_file, cs_sparse::n, cs_sparse::nz, cs_sparse::nzmax, cs_sparse::p, read_constraint(), slim_fs_learn(), start_timer(), ctrl_t::starti, and cs_sparse::x.

Referenced by main().

```
                                                   {

  /* set up timers */
  ctimer_t * timer = gk_malloc(sizeof(ctimer_t), "malloc timer"
      );
  ctimer_t * timer0 = gk_malloc(sizeof(ctimer_t), "malloc timer
      ");
  start_timer(timer0);



  /* constants used across all problems */
  int nr = train->nrows;
  int ni = train->ncols;

  /* lower/upper bound */
  double * bl = gk_malloc(sizeof(double)*ni, "malloc bl");
  gk_dset(ni, ctrl->bl, bl);
  double * bu = gk_malloc(sizeof(double)*ni, "malloc bu");
  gk_dset(ni, ctrl->bu, bu);

  /* RHS vector for all problems */
  double * b = gk_malloc(sizeof(double)*nr, "malloc b");
  gk_dset(nr, 0, b);
  /* c, linear vector */
  double * c = gk_malloc(sizeof(double)*ni, "malloc c");
  gk_dset(ni, ctrl->lambda, c);

  /* solution vector */
  double * w = gk_malloc(sizeof(double)*ni, "malloc w");
  gk_dset(ni, 0, w);

  /* the A matrix */
  gk_csr_t * A = train;
  cs * csA = gk_malloc(sizeof(cs), "malloc csA");

  /* Workspace for BCLS */
  worksp * Wrk = gk_malloc(sizeof(worksp), "malloc Wrk");
  Wrk->A = csA;
  csA->p = A->colptr;
  csA->i = A->colind;
  csA->x = A->colval;
  csA->m = A->nrows;
  csA->n = A->ncols;
  csA->nzmax = *(A->rowptr + A->nrows);
  csA->nz = -1; /* column-view */
  Wrk->max_bcls_niters = ctrl->max_bcls_niters;
  Wrk->acol = gk_malloc(sizeof(int)*ni, "malloc acol");
  gk_iset(ni, 1, Wrk->acol);

  /* temporary space for a column */
  float * A_colval = NULL;


  /* output data */
  int bsize = ctrl->bsize; /* output block size */
  gk_csr_t * mat = gk_csr_Create();
  mat->nrows = 0;
  mat->ncols = train->ncols;
  mat->rowptr = gk_malloc(sizeof(int)*(ni+1), "malloc mat->rowptr");
```

```
mat->rowptr[0] = 0;
mat->rowind = gk_malloc(sizeof(int)*ni*bsize, "malloc mat->rowind");
gk_iset(ni*bsize, 0, mat->rowind);
mat->rowval = gk_malloc(sizeof(float)*ni*bsize, "malloc mat->rowval");
gk_fset(ni*bsize, 0, mat->rowval);




/* constraint data */
gk_csr_t * constraint = NULL;
if (ctrl->fs){
  constraint = read_constraint(ctrl, ctrl->fs_file);
}


/* starting and ending columns */
int starti = (ctrl->starti >= 0)? ctrl->starti:0;
int endi    = (ctrl->endi    >= 0)? ctrl->endi:ni;

/* go through all columns  */
for (int i = starti; i < endi; i ++){

  start_timer(timer);
  printf("column %8d: ", i);

  /* the index is beyond the true boundary; this may happen due to cold start
     */
  if (i >= train->ncols){
    *(mat->rowptr + mat->nrows + 1) = *(mat->rowptr + mat->nrows);
    mat->nrows ++;
    end_timer(timer);
    display_timer(timer, "empty iter ");
    continue;
  }

  /* this column is totally empty */
  if (train->colptr[i+1] - train->colptr[i] == 0){
    *(mat->rowptr + mat->nrows + 1) = *(mat->rowptr + mat->nrows);
    mat->nrows ++;
    end_timer(timer);
    display_timer(timer, "empty iter ");
    continue;
  }
  /* in case in csr format, there are 0s recored */
  int allzeros = 1;
  for (int j = train->colptr[i]; j < train->colptr[i+1]; j ++){
    if (train->colval[j] != 0) {
  allzeros = 0; break;
    }
  }
  if (allzeros == 1){
    *(mat->rowptr + mat->nrows + 1) = *(mat->rowptr + mat->nrows);
    mat->nrows ++;
    end_timer(timer);
    display_timer(timer, "empty iter ");
    continue;
  }


  /**********************************************************/
  /*                BCLS learning                           */
  /**********************************************************/
  /* get the i-th column from A */
  gk_dset(nr, 0, b);
  get_column(A, i, b);
  /* /\* 0 <= w[i] <= 0 => w[i] = 0 *\/ */
  /* bu[i] = 0;  */
  gk_dset(ni, 0, w);
  /* disable  */
  Wrk->acol[i] = 0;

  if (!ctrl->fs){
    bcsol(ctrl, A, b, w, Wrk, bl, bu, ctrl->beta, c);
  }
  else{
    get_column(constraint, i, w);
    slim_fs_learn(ctrl, A, b, w, &A_colval, Wrk, bl, bu, ctrl->
    beta, c);
  }


  /* timing for this run */
  end_timer(timer);
  display_timer(timer, "iter ");
```

```
    /**********************************************************/
    /*              dump the data                             */
    /**********************************************************/
    /* many enough, dump the data */
    if (mat->nrows >= ctrl->bsize){
      printf("Dumping data...\n");
      csr_Write(mat, ctrl->model_file, "a", GK_CSR_FMT_CSR,
      1, 1);
      mat->nrows = 0;
    }

    /* fill out the matrix */
    *(mat->rowptr + mat->nrows + 1) = *(mat->rowptr + mat->nrows);
    for (int j = 0, k = 0; j < ni; j ++){
      if (w[j] > EPSILON){
    *(mat->rowind + mat->rowptr[mat->nrows] + k) = j;
    *(mat->rowval + mat->rowptr[mat->nrows] + k) = w[j];
    (*(mat->rowptr + mat->nrows + 1)) ++;
    k ++;
      }
    }
    mat->nrows ++;

    /* reset for the next run */
    Wrk->acol[i] = 1;
    /* bu[i] = ctrl->bu;  */


  } /* end of starti - endi */

  end_timer(timer0);
  display_timer(timer0, "BCLS");

  /* left-over data dump */
  printf("Dumping data...\n");
  csr_Write(mat, ctrl->model_file, "a", GK_CSR_FMT_CSR, 1, 1
      );



  /* finish up  */
  gk_free((void **)&timer, LTERM);   gk_free((void **)&timer0, LTERM);
  gk_csr_Free(&mat);   gk_free((void **)&w, LTERM);
  gk_free((void **)&bl, &bu, &b, &c, LTERM);
  gk_csr_Free(&constraint);
  gk_free((void **)&csA, LTERM);
  gk_free((void **)&Wrk->acol, LTERM); gk_free((void **)&Wrk, LTERM);
  gk_free((void **)&A_colval, LTERM);


}
```

5.4.2.14 void slim_predict (ctrl_t * *ctrl,* gk_csr_t * *train,* gk_csr_t * *test,* gk_csr_t * *model*)

SLIM testing.

This routine contains the testing method for SLIM

Parameters

in	*ctrl*	A ctrl structure which contains all the Parameters for SLIM testing
in	*train*	The training data, which has been used to learn the model
in	*test*	The testing data
in	*model*	The model

Definition at line 26 of file slim_predict.c.

References ctrl_t::nratings, and slim_test().

Referenced by main().

```
            {

  printf("model->nrows = %d, model->ncols = %d\n", model->nrows, model->ncols);
  /* sanity check */
  model->ncols = model->nrows;
  gk_csr_CreateIndex(model, GK_CSR_COL);

  double * eval = slim_test(ctrl, model, train, test);

  /* print the results */
  for (int j = 0; j < ctrl->nratings; j ++)
    printf("For rating value %3d HR = %.5f ARHR = %.5f cumulative HR = %.5f
       ARHR = %.5f\n",
       j+1, eval[j*4], eval[j*4+1], eval[j*4+2], eval[j*4+3]);


  /* clean up */
  gk_free((void **)&eval, LTERM);


}
```

5.4.2.15 double* slim_test (ctrl_t * *ctrl,* gk_csr_t * *model,* gk_csr_t * *train,* gk_csr_t * *test*)

Top-N recommendations and evaluations.

Parameters

in	*ctrl*	A ctrl structure
in	*model*	A model
in	*train*	The training data from which the model is learned
in	*test*	The testing data

Returns

eval A set of evaluations

Definition at line 60 of file slim_predict.c.

References ctrl_t::dbglvl, display_timer(), end_timer(), ctrl_t::nratings, ctrl_t::pred_file, start_timer(), and suggest_predict().

Referenced by slim_predict().

```
                                                   {

  int nu = test->nrows;
  int nhits = 0;
  double arh = 0;
  int n = 0;

  ctimer_t * timer = gk_malloc(sizeof(ctimer_t), "malloc timer"
      );
  start_timer(timer);

  /* evaluation results for return */
  double * eval = gk_malloc(sizeof(double)*(ctrl->nratings)*4, "malloc
       eval");
  gk_dset(ctrl->nratings*4, 0, eval);

  /* number of testing instances for each rating value */
  int * nr = gk_malloc(sizeof(int)*ctrl->nratings, "malloc nr");
  gk_iset(ctrl->nratings, 0, nr);

  int ncols = gk_max(train->ncols, model->ncols);
  int * nc = gk_malloc(sizeof(int)*ncols, "malloc nc");
  gk_iset(ncols, 0, nc);
  int * nhc = gk_malloc(sizeof(int)*ncols, "malloc nhc");
  gk_iset(ncols, 0, nhc);

  /* auxiliary_space */
  int * iidx = NULL;

  /* output file for predictions */
  FILE * pfile = NULL;
  if (ctrl->pred_file){
    pfile = gk_fopen(ctrl->pred_file, "w", "pred file");
    printf("Output predictions to %s file...\n", ctrl->pred_file);
  }

  /* predictions for all the users */
  for (int u = 0; u < nu; u ++){

    /* show the process */
    if (u % 1000 == 0) {
      if (ctrl->dbglvl == 0){
    printf("."); fflush(stdout);
      }
    }

    /* no testing instances for this user */
    if (test->rowptr[u+1] - test->rowptr[u] == 0) {
      if (ctrl->pred_file)
    fprintf(pfile, "\n");
      continue;
    }
    n ++;


    /* top-n recommendation */
    gk_dkv_t * rcmd = NULL;
    int nrcmd = 0;
    nrcmd = suggest_predict(ctrl, model, &iidx, train,  u, &rcmd
      );

    /* stats for the recommendation */
    for (int kk = test->rowptr[u]; kk < test->rowptr[u+1]; kk ++){

      int r = (int)(test->rowval[kk]); /* assume all ratings are integers [1,
       2, ..., nratings] */
      nr[r-1] ++ ;

      nc[test->rowind[kk]] ++;
    }


    /* evaluations */
    for (int jj = 0; jj < nrcmd; jj ++){

      /* output the predictions */
      if (ctrl->pred_file)
    fprintf(pfile, "%d %.5f ", (int)rcmd[jj].val+1, rcmd[jj].key);


      for (int kk = test->rowptr[u]; kk < test->rowptr[u+1]; kk ++){

    int r = (int)(test->rowval[kk]); /* assume all ratings are integers [1, 2,
       ..., nratings] */


    /* hit hit */
    if (rcmd[jj].val == test->rowind[kk]){

      nhc[test->rowind[kk]] ++;

      /* overall hit rates */
      nhits ++; arh += 1.0/(double)(jj + 1) ;
      /* hit rates on different ratings */
      eval[(r - 1)*4 + 0] += 1.0; /* hit rate on rating r */
      eval[(r - 1)*4 + 1] += 1.0/(double)(jj + 1) ; /* arh on rating r */
      eval[(r - 1)*4 + 2]  = eval[(r - 1)*4 + 0];
      eval[(r - 1)*4 + 3]  = eval[(r - 1)*4 + 1];

    }
      }

    }

    /* finalize the prediction output */
    if (ctrl->pred_file)
      fprintf(pfile, "\n");


    /* clean up */
    gk_free((void **)&rcmd, LTERM);

  }

  /* end timing */
  printf("\n");
  end_timer(timer);
  display_timer(timer, "SLIM prediction");
  /* all stats */
  for (int i = 0; i < ctrl->nratings; i ++){
    if (nr[i] > 0){
      eval[i*4 + 0] /= (double)nr[i];
      eval[i*4 + 1] /= (double)nr[i];
    }
  }
  /* cumulative stats */
  for (int i = ctrl->nratings - 2; i >= 0; i --){
    nr[i]         += nr[i+1]; /* cumulative counts */
    eval[i*4 + 2] += eval[(i+1)*4 + 2]; /* cumulative hit counts */
    eval[i*4 + 3] += eval[(i+1)*4 + 3];  /* cumulative rhr counts */
  }
  for (int i = 0; i < ctrl->nratings; i ++){
    if (nr[i] > 0){
      eval[i*4 + 2] /= (double)nr[i];
      eval[i*4 + 3] /= (double)nr[i];
    }
  }


  /* finish up */
  if (ctrl->pred_file)
    gk_fclose(pfile);
  gk_free((void **)&nc, LTERM);
  gk_free((void **)&nhc, LTERM);
  gk_free((void **)&nr, LTERM);
  gk_free((void **)&timer, LTERM);
  gk_free((void **)&iidx, LTERM);


  return eval;

}
```

5.4.2.16 void start_timer (ctimer_t * *ctimer*)

Start a timer to record current time.

Parameters

in	*ctimer*	A timer to start

Definition at line 88 of file util.c.

References ctimer_t::start.

Referenced by slim_learn(), and slim_test().

```
                                   {

  ctimer->start = clock();

}
```

5.4.2.17 int suggest_predict (ctrl_t * *ctrl,* gk_csr_t * *model,* int ** *iidx,* gk_csr_t * *train,* int *u,* gk_dkv_t ** *rcmd*)

Top-N recommendation for a user.

Parameters

in	*ctrl*	A ctrl structure
in	*model*	A model
in	*iidx*	An auxiliary array for efficient recommendations
in	*train*	Training data from which the model is learned
in	*u*	The index of the user for which the top-n recommendations are generated
out	*rcmd*	The list of recommendations, in which the keys are the recommendation scores and the values are the item indices

Returns

int The actual number of recommendations

Definition at line 228 of file slim_predict.c.

References ctrl_t::topn.

Referenced by slim_test().

```
                                                       {


  if (model->colptr == NULL)
    gk_csr_CreateIndex(model, GK_CSR_COL);

  int ni =  train->ncols;

  if (*iidx == NULL)
    *iidx = gk_malloc(sizeof(int)*ni, "malloc *iidx");

  gk_iset(ni, -1, *iidx);

  int nuitrn = train->rowptr[u+1] - train->rowptr[u];
  /* special case when no training data, thus no recommendations */
  if (nuitrn == 0){
    *rcmd = NULL;
    return 0;
  }

  for (int ii = 0; ii < nuitrn; ii ++)
    *(*iidx + *(train->rowptr[u] + ii + train->rowind)) -= 1;


  gk_dkv_t * ccandb = gk_malloc(sizeof(gk_dkv_t)*ni, "malloc ccandb");
  int nrcmd = 0;

  /* efficient recommendations */
  nuitrn = train->rowptr[u+1] - train->rowptr[u];
  for (int i = 0; i < nuitrn; i ++){
    int ii = *(train->rowptr[u] + i + train->rowind);
    for (int j = 0; j < model->colptr[ii+1] - model->colptr[ii]; j ++){
      int jj = *(model->colptr[ii] + j + model->colind);
      if ((*iidx)[jj] < -1) continue;
      if ((*iidx)[jj] == -1){
    (*iidx)[jj] = nrcmd;
    ccandb[nrcmd].key = *(model->colptr[ii] + j + model->colval) * 1.0;
    ccandb[nrcmd].val = jj;
    nrcmd ++;
      }else{
    ccandb[(*iidx)[jj]].key += *(model->colptr[ii] + j + model->colval) * 1.0;
      }
    }
  }

  /* sorting */
  gk_dkvsortd(nrcmd, ccandb);
  int nrcmd2 = gk_min(nrcmd, ctrl->topn);
  *rcmd = ccandb;


  return nrcmd2;

}
```

5.5 /home/xning/Project/SLIMLib/include/struct.h File Reference

This file contains all the necessary data structures.

Data Structures

- struct ctimer_t

  A data structure for timer.
- struct ctrl_t

  A data structure for ctrl parameters.

- struct cs_sparse

  A matrix structure used for BCLS. This is adopted from BCLS.
- struct worksp

  A workspace structure used for BCLS. This is adopated from BCLS.

Typedefs

- typedef struct cs_sparse cs

  A matrix structure used for BCLS. This is adopted from BCLS.

5.5.1 Detailed Description

This file contains all the necessary data structures.

Definition in file struct.h.

5.6 /home/xning/Project/SLIMLib/src/bcsol.c File Reference

This file contains all the routines needed for BCLS optimization.

```
#include <slim.h>
```

Macros

- #define CS_MAX(a, b) (((a) > (b)) ? (a) : (b))

  Compute the maximum of two.
- #define CS_MIN(a, b) (((a) < (b)) ? (a) : (b))

  Compute the minimum of two.
- #define CS_FLIP(i) (-(i)-2)

  Flip.
- #define CS_UNFLIP(i) (((i) < 0) ? CS_FLIP(i) : (i))

  Unflip.
- #define CS_MARKED(w, j) (w [j] < 0)

  Check if marked.
- #define CS_MARK(w, j) { w [j] = CS_FLIP (w [j]) ; }

  Mark.
- #define CS_CSC(A) (A && (A->nz == -1))

  CSC.
- #define CS_TRIPLET(A) (A && (A->nz >= 0))

  Triplet.

Functions

- int call_back (BCLS ∗ls, void ∗UsrWrk)

  call_back function, periodically called by BCLS to test if the user wants to exit. This is from BCLS.
- int call_back_it (BCLS ∗ls, void ∗UsrWrk)

  call_back function, immediately terminate BCLS iterations based on how many iterations it runs
- int pretty_printer (void ∗io_file, char ∗msg)

  Pretty_printer, this is the print-routine that will be used by BCLS for its output. This is from BCLS.

- void ∗ cs_free (void ∗p)

  Wrapper for free.
- void ∗ cs_malloc (int n, size_t size)

  Wrapper for malloc.
- void ∗ cs_calloc (int n, size_t size)

  Wrapper for calloc.
- cs ∗ cs_spfree (cs ∗A)

  Free a sparse matrix.
- cs ∗ cs_spalloc (int m, int n, int nzmax, int values, int triplet)

  Allocate a sparse matrix (triplet form or compressed-column form)
- void dload (const int n, const double alpha, double x[])

  Load a constant into a vector.
- int Aprod (int mode, int m, int n, int nix, int ix[], double x[], double y[], void ∗UsrWrk)

  Aprod, matrix-vector products. This is from BCLS.
- void bcsol (ctrl_t ∗ctrl, gk_csr_t ∗AA, double ∗bb, double ∗x, worksp ∗Wrk, double ∗bl, double ∗bu, double beta, double ∗c)

  BCLS learning. This is from BCLS.

Variables

- int bcls_niters = 0

  Number of iterations.

5.6.1 Detailed Description

This file contains all the routines needed for BCLS optimization.

Definition in file bcsol.c.

5.6.2 Function Documentation

5.6.2.1 int Aprod (int *mode,* int *m,* int *n,* int *nix,* int *ix[ ],* double *x[ ],* double *y[ ],* void ∗ *UsrWrk*)

Aprod, matrix-vector products. This is from BCLS.

If mode == BCLS_PROD_A (0), compute y <- A ∗x, with x untouched; and if mode == BCLS_PROD_At (1), compute x <- A'∗y, with y untouched.

Definition at line 163 of file bcsol.c.

References worksp::A, worksp::acol, cs_sparse::i, cs_sparse::p, and cs_sparse::x.

Referenced by bcsol().

```
                                                              {

  int      i, j, k, l;
  double  aij;
  double xj, sum;
  worksp * Wrk = (worksp *)UsrWrk;
  cs *A = (cs *)Wrk->A;
  int * Ai = A->i;
  int * Ap = A->p;
  float * Ax = A->x;
  int * acol = Wrk->acol;


  if (mode == BCLS_PROD_A) {

    gk_dset(m, 0.0, y);
    for (l = 0; l < nix; l++) {
      j = ix[l];

      /* skip the inactive column */
      if (!acol[j]) continue;

      xj = x[j];
      if (xj == 0.0)
    ; // Relax.
      else
    for (k = Ap[j]; k < Ap[j+1]; k++) {
      aij   = Ax[k];


      /* this is to handle float-valued A matrix */
      i     = Ai[k];
      y[i] += aij * xj;


    }
    }
  }

  else if (mode == BCLS_PROD_At) {
    for (l = 0; l < nix; l++) {
      j = ix[l];
      sum = 0;

      /* skip the inactive column */
      if (!acol[j]){
    x[j] = sum;
    continue;
      }

      for (k = Ap[j]; k < Ap[j+1]; k++) {
    aij  = Ax[k];

    /* this is to handle float-valued A matrix */
    i    = Ai[k];
    sum += aij * y[i];


      }
      x[j] = sum;
    }
  }


  return 0;
}
```

5.6.2.2 void bcsol (**ctrl_t** * *ctrl,* gk_csr_t * *AA,* double * *bb,* double * *x,* **worksp** * *Wrk,* double * *bl,* double * *bu,* double *beta,* double * *c*)

BCLS learning. This is from BCLS.

This is to solve the problem

$$\begin{array}{ll} \underset{x}{\text{minimize}} & \frac{1}{2}\|Ax - a_i\|_2^2 + \frac{1}{2}\beta\|x\|_2^2 + \lambda\|x\|_1 \\ \text{subject to} & 0 \le x \\ & x_i = 0 \end{array}$$

Definition at line 247 of file bcsol.c.

References Aprod(), bcls_niters, call_back(), call_back_it(), ctrl_t::dbglvl, ctrl_t::max_bcls_niters, ctrl_t::optTol, and pretty_printer().

Referenced by slim_fs_learn(), and slim_learn().

```
                                {

  bcls_niters = 0;

  /*  Problem dimensions. */
  int m = AA->nrows;
  int n = AA->ncols;

  /* init a bcls problem */
  BCLS   *ls = bcls_create_prob( m, n );
  bcls_set_problem_data(ls, m, n, Aprod,  Wrk, beta, x, bb, c, bl, bu);


  /* set up tolerance */
  ls->optTol = ctrl->optTol;

  /* whatever */
  bcls_set_print_hook( ls, stdout, pretty_printer );
  ls->proj_search     = proj_search;
  ls->newton_step     = newton_step;
  /* if (ctrl->test_bcls) */
  if (ctrl->max_bcls_niters > 0)
    ls->CallBack       = call_back_it;
  else
    ls->CallBack       = call_back;

  /* call the solver */
  int err = bcls_solve_prob( ls );

  /* solution */
  if (ctrl->dbglvl > 1){
    int nnzx = 0;
    printf("\n Solution\n --------\n");
    printf("%4s  %18s %1s %18s %1s %18s  %18s\n",
       "Var","Lower","","Value","","Upper","Gradient");
    for (int j = 0; j < n; j++) {
      if (x[j] > 1e-10){
    nnzx ++;
    char * blActiv = "";
    char * buActiv = "";
    if (x[j] - bl[j] < ls->epsx) blActiv = "=";
    if (bu[j] - x[j] < ls->epsx) buActiv = "=";
    printf("%4d  %18.11e %1s %18.11e %1s %18.11e  %18.11e\n",
           j+1, bl[j], blActiv, x[j], buActiv, bu[j], (ls->g)[j]);
      }
    }
    printf("%d nnz solution values\n", nnzx);
  }


  /* free the problem */
  err = bcls_free_prob( ls );

}
```

5.7 /home/xning/Project/SLIMLib/src/check.c File Reference

This file constains routines for data pre-processing.

`#include <slim.h>`

Functions

- void check_train_test (ctrl_t ∗ctrl, gk_csr_t ∗train, gk_csr_t ∗test)

  Check if test data are already in train data.

5.7.1 Detailed Description

This file constains routines for data pre-processing.

Definition in file check.c.

5.7.2 Function Documentation

5.7.2.1 void check_train_test (ctrl_t ∗ *ctrl,* gk_csr_t ∗ *train,* gk_csr_t ∗ *test*)

Check if test data are already in train data.

Parameters

in	*ctrl*	A ctrl structure
in	*train*	The training data
in	*test*	The testing data

Definition at line 19 of file check.c.

Referenced by preprocess().

```
                                                                        {

  int error = 0;

  int nrows_test = test->nrows;

  for (int i = 0; i < nrows_test; i ++){

    int nc_test = test->rowptr[i+1] - test->rowptr[i];
    int nc_train = train->rowptr[i+1] - train->rowptr[i];

    for (int j = 0; j < nc_test; j ++){

      int item_test = *(test->rowptr[i] + j + test->rowind);

      for (int k = 0; k < nc_train; k ++){

    int item_train = *(train->rowptr[i] + k + train->rowind);

    if (item_test == item_train){
      printf("ERROR: user %6d has item %6d in both train and test\n", i,
      item_train);
      error = 1;
      break;
    }

      }

    }

  }

  if (error)
    errexit("ERROR: train and test not disjoint\n");

}
```

5.8 /home/xning/Project/SLIMLib/src/cmd.c File Reference

This file contains all the routines for parameter setup from the user.

`#include <slim.h>`

Functions

- void parse_cmdline (ctrl_t ∗ctrl, int argc, char ∗argv[ ])

  Entry point of the command-line argument parsing.

5.8.1 Detailed Description

This file contains all the routines for parameter setup from the user.

Definition in file cmd.c.

5.8.2 Function Documentation

5.8.2.1 void parse_cmdline (ctrl_t * *ctrl,* int *argc,* char * *argv[]*)

Entry point of the command-line argument parsing.

Parameters

out	*ctrl*	A ctrl structure to be filled out
in	*argc*	Number of arguments
in	*argv*	A list of arguments

Definition at line 146 of file cmd.c.

References ctrl_t::beta, ctrl_t::bsize, CMD_BETA, CMD_BSIZE, CMD_DBGLVL, CMD_ENDI, CMD_FS, CMD_FS_FILE, CMD_HELP, CMD_K, CMD_LAMBDA, CMD_MAX_BCLS_NITERS, CMD_MODEL_FILE, CMD_NRATINGS, CMD_OPTTOL, CMD_PRED_FILE, CMD_STARTI, CMD_TEST_FILE, CMD_TOPN, CMD_TRAIN_FILE, CMD_TRANSPOSE, create_ctrl(), ctrl_t::dbglvl, ctrl_t::endi, ctrl_t::fs, ctrl_t::fs_file, ctrl_t::k, ctrl_t::lambda, ctrl_t::max_bcls_niters, ctrl_t::model_file, ctrl_t::nratings, ctrl_t::optTol, ctrl_t::pred_file, ctrl_t::starti, ctrl_t::test_file, ctrl_t::topn, ctrl_t::train_file, and ctrl_t::transpose.

Referenced by main().

```
                                                         {
  int c = -1, option_index = -1;

  if (ctrl == NULL)
    ctrl = create_ctrl();

  while((c = gk_getopt_long_only(argc, argv, "", slim_options, &option_index))
      != -1){
    switch(c){


    case CMD_TRAIN_FILE:
      ctrl->train_file = gk_strdup(gk_optarg);
      break;

    case CMD_TEST_FILE:
      ctrl->test_file = gk_strdup(gk_optarg);
      break;

    case CMD_MODEL_FILE:
      ctrl->model_file = gk_strdup(gk_optarg);
      break;

    case CMD_PRED_FILE:
      ctrl->pred_file = gk_strdup(gk_optarg);
      break;

    case CMD_DBGLVL:
      ctrl->dbglvl = atoi(gk_optarg);
      break;

    case CMD_LAMBDA:
      ctrl->lambda = atof(gk_optarg);
      break;

    case CMD_BETA:
      ctrl->beta = atof(gk_optarg);
      break;

    case CMD_STARTI:
      ctrl->starti = atoi(gk_optarg);
      break;

    case CMD_ENDI:
      ctrl->endi = atoi(gk_optarg);
      break;

    case CMD_OPTTOL:
      ctrl->optTol = atof(gk_optarg);
      break;

    case CMD_MAX_BCLS_NITERS:
      ctrl->max_bcls_niters = atoi(gk_optarg);
      break;

    case CMD_FS:
      ctrl->fs = 1;
      break;

    case CMD_FS_FILE:
      ctrl->fs_file = gk_strdup(gk_optarg);
      break;

    case CMD_K:
      ctrl->k = atoi(gk_optarg);
      break;

    case CMD_BSIZE:
      ctrl->bsize = atoi(gk_optarg);
      break;

    case CMD_NRATINGS:
      ctrl->nratings = atoi(gk_optarg);
      break;

    case CMD_TOPN:
      ctrl->topn = atoi(gk_optarg);
      break;

    case CMD_TRANSPOSE:
      ctrl->transpose = 1;
      break;

    case CMD_HELP:
      for (int i=0; strlen(helpstr[i]) > 0; i++)
    printf("%s\n", helpstr[i]);
      exit(0);


    case '?':
    default:
      printf("Illegal command-line option(s) %s\n", gk_optarg);
      exit(0);

    }
  }


  if (argc-gk_optind != 0 || argc == 1) {
    for (int i=0; strlen(shorthelpstr[i]) > 0; i++)
      printf("%s\n", shorthelpstr[i]);
    exit(0);
  }


}
```

5.9 /home/xning/Project/SLIMLib/src/io.c File Reference

This file contains all the I/O routines.

`#include <slim.h>`

Functions

- gk_csr_t ∗ read_constraint (ctrl_t ∗ctrl, char ∗file)

  Read in a constraint matrix for feature selection.

- void csr_Write (gk_csr_t ∗mat, char ∗filename, char ∗mode, int format, int writevals, int numbering)

  Dump the csr into a file.

5.9.1 Detailed Description

This file contains all the I/O routines.

Definition in file io.c.

5.10 /home/xning/Project/SLIMLib/src/process.c File Reference

This file constains routines for data pre-processing.

```
#include <slim.h>
```

Functions

- void preprocess (ctrl_t ∗ctrl, gk_csr_t ∗train, gk_csr_t ∗test)

  Pre-process the data.

5.10.1 Detailed Description

This file constains routines for data pre-processing.

Definition in file process.c.

5.11 /home/xning/Project/SLIMLib/src/slim_fs_learn.c File Reference

This file contains all the routines for SLIM learning with feature selection.

```
#include <slim.h>
```

Functions

- void slim_fs_learn (ctrl_t ∗ctrl, gk_csr_t ∗A, double ∗b, double ∗w, float ∗∗A_colval, worksp ∗Wrk, double ∗bl, double ∗bu, double beta, double ∗c)

  SLIM learning with feature selction.

5.11.1 Detailed Description

This file contains all the routines for SLIM learning with feature selection.

Definition in file slim_fs_learn.c.

5.11.2 Function Documentation

5.11.2.1 void slim_fs_learn (ctrl_t ∗ *ctrl,* gk_csr_t ∗ *A,* double ∗ *b,* double ∗ *w,* float ∗∗ *A_colval,* worksp ∗ *Wrk,* double ∗ *bl,* double ∗ *bu,* double *beta,* double ∗ *c*)

SLIM learning with feature selction.

Parameters

in	*ctrl*	A ctrl structure which contains all the parameters for SLIM Learning with feature selection
in	*A*	The A matrix
in	*b*	The RHS vector
in,out	*w*	The solution vector
in	*A_colval*	A temporary place for a column
in	*Wrk*	A workspace for BCLS
in	*bl*	The lower bound for BCLS

in	*bu*	The upper bound for BCLS
in	*beta*	The regularization parameter for L-2 norm
in	*c*	The vector for L-1 norm

Definition at line 31 of file slim_fs_learn.c.

References worksp::acol, bcsol(), count_nnz(), find_topk(), and ctrl_t::k.

Referenced by slim_learn().

```
                                   {

  int nnz = *(A->colptr + A->ncols);
  int * acol = Wrk->acol;

  /* count nnz */
  int kk = count_nnz(w, A->ncols);

  /* find topk nnz */
  int topk = 0;
  /* find the indices of topk entries, meanwhile w is over-written as the topk
       locations */
  find_topk(w, A->ncols, gk_min(ctrl->k, kk), w, &topk);

  /* back up original values, this is done only once */
  if (*A_colval == NULL){
    *A_colval = gk_malloc(sizeof(float)*nnz, "malloc *A_colval");
    memcpy((void *)*A_colval, (void *)A->colval, sizeof(float)*nnz);
  }

  /* remove all A nnz values, this will not affect the column under
       consideration */
  gk_fset(nnz, 0, A->colval);
  /* set all columns as inactive */
  gk_iset(A->ncols, 0, acol);
  /* recover all topk columns in A */
  for (int i = 0; i < topk; i ++){
    int j = (int)w[i];
    /* activate this column */
    acol[j] = 1;
    int nj = A->colptr[j+1] - A->colptr[j];
    for (int k = 0; k < nj; k ++){
      /* get the orignal values back */
      *(A->colptr[j] + k + A->colval) = *(A->colptr[j] + k + *A_colval);
    }
  }

  /* BCLS */
  gk_dset(A->ncols, 0, w);
  bcsol(ctrl, A, b, w, Wrk, bl, bu, beta, c);

  /* recover full A, specific to binary A, this will over-write the column of
       b,
     but will not matter */
  memcpy((void *)A->colval, (void *)*A_colval, sizeof(float)*nnz);
  /* activate all columns */
  gk_iset(A->ncols, 1, acol);

}
```

5.12 /home/xning/Project/SLIMLib/src/slim_learn.c File Reference

This file contains all the routines for SLIM learning.

`#include <slim.h>`

Functions

- void slim_learn (ctrl_t ∗ctrl, gk_csr_t ∗train)

  SLIM learning.

5.12.1 Detailed Description

This file contains all the routines for SLIM learning.

Definition in file slim_learn.c.

5.12.2 Function Documentation

5.12.2.1 void slim_learn (ctrl_t * *ctrl,* gk_csr_t * *train*)

SLIM learning.

This routine contains the learning algorithm for SLIM

Parameters

in	*ctrl*	A ctrl structure which contains all the parameters for SLIM learning
in	*train*	The training data

Definition at line 22 of file slim_learn.c.

References worksp::A, worksp::acol, bcsol(), ctrl_t::beta, ctrl_t::bl, ctrl_t::bsize, ctrl_t::bu, csr_Write(), display_timer(), end_timer(), ctrl_t::endi, EPSILON, ctrl_t::fs, ctrl_t::fs_file, get_column(), cs_sparse::i, ctrl_t::lambda, cs_sparse::m, ctrl_t::max_bcls_niters, worksp::max_bcls_niters, ctrl_t::model_file, cs_sparse::n, cs_sparse::nz, cs_sparse::nzmax, cs_sparse::p, read_constraint(), slim_fs_learn(), start_timer(), ctrl_t::starti, and cs_sparse::x.

Referenced by main().

```
                                                 {

/* set up timers */
ctimer_t * timer = gk_malloc(sizeof(ctimer_t), "malloc timer"
    );
ctimer_t * timer0 = gk_malloc(sizeof(ctimer_t), "malloc timer
    ");
start_timer(timer0);



/* constants used across all problems */
int nr = train->nrows;
int ni = train->ncols;

/* lower/upper bound */
double * bl = gk_malloc(sizeof(double)*ni, "malloc bl");
gk_dset(ni, ctrl->bl, bl);
double * bu = gk_malloc(sizeof(double)*ni, "malloc bu");
gk_dset(ni, ctrl->bu, bu);

/* RHS vector for all problems */
double * b = gk_malloc(sizeof(double)*nr, "malloc b");
gk_dset(nr, 0, b);
/* c, linear vector */
double * c = gk_malloc(sizeof(double)*ni, "malloc c");
gk_dset(ni, ctrl->lambda, c);

/* solution vector */
double * w = gk_malloc(sizeof(double)*ni, "malloc w");
gk_dset(ni, 0, w);

/* the A matrix */
gk_csr_t * A = train;
cs * csA = gk_malloc(sizeof(cs), "malloc csA");

/* Workspace for BCLS */
worksp * Wrk = gk_malloc(sizeof(worksp), "malloc Wrk");
Wrk->A = csA;
csA->p = A->colptr;
csA->i = A->colind;
csA->x = A->colval;
csA->m = A->nrows;
csA->n = A->ncols;
csA->nzmax = *(A->rowptr + A->nrows);
csA->nz = -1; /* column-view */
Wrk->max_bcls_niters = ctrl->max_bcls_niters;
```

```
Wrk->acol = gk_malloc(sizeof(int)*ni, "malloc acol");
gk_iset(ni, 1, Wrk->acol);

/* temporary space for a column */
float * A_colval = NULL;


/* output data */
int bsize = ctrl->bsize; /* output block size */
gk_csr_t * mat = gk_csr_Create();
mat->nrows = 0;
mat->ncols = train->ncols;
mat->rowptr = gk_malloc(sizeof(int)*(ni+1), "malloc mat->rowptr");
mat->rowptr[0] = 0;
mat->rowind = gk_malloc(sizeof(int)*ni*bsize, "malloc mat->rowind");
gk_iset(ni*bsize, 0, mat->rowind);
mat->rowval = gk_malloc(sizeof(float)*ni*bsize, "malloc mat->rowval");
gk_fset(ni*bsize, 0, mat->rowval);




/* constraint data */
gk_csr_t * constraint = NULL;
if (ctrl->fs){
  constraint = read_constraint(ctrl, ctrl->fs_file);
}


/* starting and ending columns */
int starti = (ctrl->starti >= 0)? ctrl->starti:0;
int endi   = (ctrl->endi   >= 0)? ctrl->endi:ni;

/* go through all columns  */
for (int i = starti; i < endi; i ++){

  start_timer(timer);
  printf("column %8d: ", i);

  /* the index is beyond the true boundary; this may happen due to cold start
     */
  if (i >= train->ncols){
    *(mat->rowptr + mat->nrows + 1) = *(mat->rowptr + mat->nrows);
    mat->nrows ++;
    end_timer(timer);
    display_timer(timer, "empty iter ");
    continue;
  }

  /* this column is totally empty */
  if (train->colptr[i+1] - train->colptr[i] == 0){
    *(mat->rowptr + mat->nrows + 1) = *(mat->rowptr + mat->nrows);
    mat->nrows ++;
    end_timer(timer);
    display_timer(timer, "empty iter ");
    continue;
  }
  /* in case in csr format, there are 0s recored */
  int allzeros = 1;
  for (int j = train->colptr[i]; j < train->colptr[i+1]; j ++){
    if (train->colval[j] != 0) {
  allzeros = 0; break;
    }
  }
  if (allzeros == 1){
    *(mat->rowptr + mat->nrows + 1) = *(mat->rowptr + mat->nrows);
    mat->nrows ++;
    end_timer(timer);
    display_timer(timer, "empty iter ");
    continue;
  }


  /**********************************************************/
  /*                BCLS learning                           */
  /**********************************************************/
  /* get the i-th column from A */
  gk_dset(nr, 0, b);
  get_column(A, i, b);
  /* /\* 0 <= w[i] <= 0 => w[i] = 0 *\/ */
  /* bu[i] = 0;  */
  gk_dset(ni, 0, w);
  /* disable  */
  Wrk->acol[i] = 0;

  if (!ctrl->fs){
    bcsol(ctrl, A, b, w, Wrk, bl, bu, ctrl->beta, c);
    }
    else{
      get_column(constraint, i, w);
      slim_fs_learn(ctrl, A, b, w, &A_colval, Wrk, bl, bu, ctrl->
      beta, c);
    }


    /* timing for this run */
    end_timer(timer);
    display_timer(timer, "iter ");



    /**********************************************************/
    /*              dump the data                             */
    /**********************************************************/
    /* many enough, dump the data */
    if (mat->nrows >= ctrl->bsize){
      printf("Dumping data...\n");
      csr_Write(mat, ctrl->model_file, "a", GK_CSR_FMT_CSR,
      1, 1);
      mat->nrows = 0;
    }

    /* fill out the matrix */
    *(mat->rowptr + mat->nrows + 1) = *(mat->rowptr + mat->nrows);
    for (int j = 0, k = 0; j < ni; j ++){
      if (w[j] > EPSILON){
    *(mat->rowind + mat->rowptr[mat->nrows] + k) = j;
    *(mat->rowval + mat->rowptr[mat->nrows] + k) = w[j];
    (*(mat->rowptr + mat->nrows + 1)) ++;
    k ++;
      }
    }
    mat->nrows ++;

    /* reset for the next run */
    Wrk->acol[i] = 1;
    /* bu[i] = ctrl->bu;  */


  } /* end of starti - endi */

  end_timer(timer0);
  display_timer(timer0, "BCLS");

  /* left-over data dump */
  printf("Dumping data...\n");
  csr_Write(mat, ctrl->model_file, "a", GK_CSR_FMT_CSR, 1, 1
      );



  /* finish up  */
  gk_free((void **)&timer, LTERM);   gk_free((void **)&timer0, LTERM);
  gk_csr_Free(&mat);   gk_free((void **)&w, LTERM);
  gk_free((void **)&bl, &bu, &b, &c, LTERM);
  gk_csr_Free(&constraint);
  gk_free((void **)&csA, LTERM);
  gk_free((void **)&Wrk->acol, LTERM); gk_free((void **)&Wrk, LTERM);
  gk_free((void **)&A_colval, LTERM);


}
```

5.13 /home/xning/Project/SLIMLib/src/slim_predict.c File Reference

This file contains all the routines for SLIM testing.

`#include <slim.h>`

Functions

- void slim_predict (ctrl_t ∗ctrl, gk_csr_t ∗train, gk_csr_t ∗test, gk_csr_t ∗model)

  SLIM testing.

- double ∗ slim_test (ctrl_t ∗ctrl, gk_csr_t ∗model, gk_csr_t ∗train, gk_csr_t ∗test)

  Top-N recommendations and evaluations.
- int suggest_predict (ctrl_t ∗ctrl, gk_csr_t ∗model, int ∗∗iidx, gk_csr_t ∗train, int u, gk_dkv_t ∗∗rcmd)

  Top-N recommendation for a user.

5.13.1 Detailed Description

This file contains all the routines for SLIM testing.

Definition in file slim_predict.c.

5.13.2 Function Documentation

5.13.2.1 void slim_predict (ctrl_t ∗ *ctrl,* gk_csr_t ∗ *train,* gk_csr_t ∗ *test,* gk_csr_t ∗ *model*)

SLIM testing.

This routine contains the testing method for SLIM

Parameters

in	*ctrl*	A ctrl structure which contains all the Parameters for SLIM testing
in	*train*	The training data, which has been used to learn the model
in	*test*	The testing data
in	*model*	The model

Definition at line 26 of file slim_predict.c.

References ctrl_t::nratings, and slim_test().

Referenced by main().

```
            {
  printf("model->nrows = %d, model->ncols = %d\n", model->nrows, model->ncols);

  /* sanity check */
  model->ncols = model->nrows;
  gk_csr_CreateIndex(model, GK_CSR_COL);

  double * eval = slim_test(ctrl, model, train, test);

  /* print the results */
  for (int j = 0; j < ctrl->nratings; j ++)
    printf("For rating value %3d HR = %.5f ARHR = %.5f cumulative HR = %.5f
       ARHR = %.5f\n",
       j+1, eval[j*4], eval[j*4+1], eval[j*4+2], eval[j*4+3]);


  /* clean up */
  gk_free((void **)&eval, LTERM);


}
```

5.13.2.2 double∗ slim_test (ctrl_t ∗ *ctrl,* gk_csr_t ∗ *model,* gk_csr_t ∗ *train,* gk_csr_t ∗ *test*)

Top-N recommendations and evaluations.

Parameters

in	*ctrl*	A ctrl structure
in	*model*	A model
in	*train*	The training data from which the model is learned
in	*test*	The testing data

Returns

eval A set of evaluations

Definition at line 60 of file slim_predict.c.

References ctrl_t::dbglvl, display_timer(), end_timer(), ctrl_t::nratings, ctrl_t::pred_file, start_timer(), and suggest_-predict().

Referenced by slim_predict().

```
                                           {

int nu = test->nrows;
int nhits = 0;
double arh = 0;
int n = 0;

ctimer_t * timer = gk_malloc(sizeof(ctimer_t), "malloc timer"
    );
start_timer(timer);

/* evaluation results for return */
double * eval = gk_malloc(sizeof(double)*(ctrl->nratings)*4, "malloc
     eval");
gk_dset(ctrl->nratings*4, 0, eval);

/* number of testing instances for each rating value */
int * nr = gk_malloc(sizeof(int)*ctrl->nratings, "malloc nr");
gk_iset(ctrl->nratings, 0, nr);

int ncols = gk_max(train->ncols, model->ncols);
int * nc = gk_malloc(sizeof(int)*ncols, "malloc nc");
gk_iset(ncols, 0, nc);
int * nhc = gk_malloc(sizeof(int)*ncols, "malloc nhc");
gk_iset(ncols, 0, nhc);

/* auxiliary_space */
int * iidx = NULL;

/* output file for predictions */
FILE * pfile = NULL;
if (ctrl->pred_file){
  pfile = gk_fopen(ctrl->pred_file, "w", "pred file");
  printf("Output predictions to %s file...\n", ctrl->pred_file);
}

/* predictions for all the users */
for (int u = 0; u < nu; u ++){

  /* show the process */
  if (u % 1000 == 0) {
    if (ctrl->dbglvl == 0){
  printf("."); fflush(stdout);
    }
  }

  /* no testing instances for this user */
  if (test->rowptr[u+1] - test->rowptr[u] == 0) {
    if (ctrl->pred_file)
  fprintf(pfile, "\n");
    continue;
  }
  n ++;


  /* top-n recommendation */
  gk_dkv_t * rcmd = NULL;
  int nrcmd = 0;
  nrcmd = suggest_predict(ctrl, model, &iidx, train,  u, &rcmd
    );

  /* stats for the recommendation */
  for (int kk = test->rowptr[u]; kk < test->rowptr[u+1]; kk ++){

    int r = (int)(test->rowval[kk]); /* assume all ratings are integers [1,
     2, ..., nratings] */
    nr[r-1] ++ ;

    nc[test->rowind[kk]] ++;
  }


  /* evaluations */
  for (int jj = 0; jj < nrcmd; jj ++){
      /* output the predictions */
      if (ctrl->pred_file)
    fprintf(pfile, "%d %.5f ", (int)rcmd[jj].val+1, rcmd[jj].key);


      for (int kk = test->rowptr[u]; kk < test->rowptr[u+1]; kk ++){

    int r = (int)(test->rowval[kk]); /* assume all ratings are integers [1, 2,
       ..., nratings] */


    /* hit hit */
    if (rcmd[jj].val == test->rowind[kk]){

      nhc[test->rowind[kk]] ++;

      /* overall hit rates */
      nhits ++; arh += 1.0/(double)(jj + 1) ;
      /* hit rates on different ratings */
      eval[(r - 1)*4 + 0] += 1.0; /* hit rate on rating r */
      eval[(r - 1)*4 + 1] += 1.0/(double)(jj + 1) ; /* arh on rating r */
      eval[(r - 1)*4 + 2]  = eval[(r - 1)*4 + 0];
      eval[(r - 1)*4 + 3]  = eval[(r - 1)*4 + 1];

    }
      }

    }

    /* finalize the prediction output */
    if (ctrl->pred_file)
      fprintf(pfile, "\n");


    /* clean up */
    gk_free((void **)&rcmd, LTERM);

  }

  /* end timing */
  printf("\n");
  end_timer(timer);
  display_timer(timer, "SLIM prediction");


  /* all stats */
  for (int i = 0; i < ctrl->nratings; i ++){
    if (nr[i] > 0){
      eval[i*4 + 0] /= (double)nr[i];
      eval[i*4 + 1] /= (double)nr[i];
    }
  }
  /* cumulative stats */
  for (int i = ctrl->nratings - 2; i >= 0; i --){
    nr[i]          += nr[i+1]; /* cumulative counts */
    eval[i*4 + 2] += eval[(i+1)*4 + 2]; /* cumulative hit counts */
    eval[i*4 + 3] += eval[(i+1)*4 + 3];  /* cumulative rhr counts */
  }
  for (int i = 0; i < ctrl->nratings; i ++){
    if (nr[i] > 0){
      eval[i*4 + 2] /= (double)nr[i];
      eval[i*4 + 3] /= (double)nr[i];
    }
  }


  /* finish up */
  if (ctrl->pred_file)
    gk_fclose(pfile);
  gk_free((void **)&nc, LTERM);
  gk_free((void **)&nhc, LTERM);
  gk_free((void **)&nr, LTERM);
  gk_free((void **)&timer, LTERM);
  gk_free((void **)&iidx, LTERM);


  return eval;

}
```

5.13.2.3 int suggest_predict (ctrl_t * *ctrl*, gk_csr_t * *model*, int ** *iidx*, gk_csr_t * *train*, int *u*, gk_dkv_t ** *rcmd*)

Top-N recommendation for a user.

Parameters

in	*ctrl*	A ctrl structure
in	*model*	A model
in	*iidx*	An auxiliary array for efficient recommendations
in	*train*	Training data from which the model is learned
in	*u*	The index of the user for which the top-n recommendations are generated
out	*rcmd*	The list of recommendations, in which the keys are the recommendation scores and the values are the item indices

Returns

int The actual number of recommendations

Definition at line 228 of file slim_predict.c.

References ctrl_t::topn.

Referenced by slim_test().

```
                                                      {


  if (model->colptr == NULL)
    gk_csr_CreateIndex(model, GK_CSR_COL);

  int ni =  train->ncols;

  if (*iidx == NULL)
    *iidx = gk_malloc(sizeof(int)*ni, "malloc *iidx");

  gk_iset(ni, -1, *iidx);

  int nuitrn = train->rowptr[u+1] - train->rowptr[u];
  /* special case when no training data, thus no recommendations */
  if (nuitrn == 0){
    *rcmd = NULL;
    return 0;
  }

  for (int ii = 0; ii < nuitrn; ii ++)
    *(*iidx + *(train->rowptr[u] + ii + train->rowind)) -= 1;


  gk_dkv_t * ccandb = gk_malloc(sizeof(gk_dkv_t)*ni, "malloc ccandb");
  int nrcmd = 0;

  /* efficient recommendations */
  nuitrn = train->rowptr[u+1] - train->rowptr[u];
  for (int i = 0; i < nuitrn; i ++){
    int ii = *(train->rowptr[u] + i + train->rowind);
    for (int j = 0; j < model->colptr[ii+1] - model->colptr[ii]; j ++){
      int jj = *(model->colptr[ii] + j + model->colind);
      if ((*iidx)[jj] < -1) continue;
      if ((*iidx)[jj] == -1){
    (*iidx)[jj] = nrcmd;
    ccandb[nrcmd].key = *(model->colptr[ii] + j + model->colval) * 1.0;
    ccandb[nrcmd].val = jj;
    nrcmd ++;
      }else{
    ccandb[(*iidx)[jj]].key += *(model->colptr[ii] + j + model->colval) * 1.0;
      }
    }
  }

  /* sorting */
  gk_dkvsortd(nrcmd, ccandb);
  int nrcmd2 = gk_min(nrcmd, ctrl->topn);
  *rcmd = ccandb;


  return nrcmd2;

}
```

5.14 /home/xning/Project/SLIMLib/src/util.c File Reference

This file contains all the utility routines.

```
#include <slim.h>
```

Functions

- ctrl_t ∗ create_ctrl ()

  Create a ctrl structure wich contains all the default parameters for SLIM.
- void free_ctrl (ctrl_t ∗ctrl)

  Free a ctrl structure.
- void start_timer (ctimer_t ∗ctimer)

  Start a timer to record current time.
- void end_timer (ctimer_t ∗ctimer)

  End a timer to record a length of a duration.
- void display_timer (ctimer_t ∗ctimer, char ∗msg)

  Display a user-defined message and a duration length recorded by a timer.
- int count_nnz (double ∗array, int narray)

  Count the number of non-zero values in an array.
- void find_topk (double ∗w, int n, int topk, double ∗map, int ∗topk2)

  Find the top-k values from an array.
- void get_column (gk_csr_t ∗constraint, int i, double ∗w)

  Get a column from a csr matrix.

5.14.1 Detailed Description

This file contains all the utility routines.

Definition in file util.c.

5.14.2 Function Documentation

5.14.2.1 int count_nnz (double ∗ *array,* int *narray*)

Count the number of non-zero values in an array.

Parameters

in	*array*	An array whose non-zero values will be counted
in	*narray*	The length of the array

Returns

int The number of non-zero values in the array

Definition at line 134 of file util.c.

References EPSILON2.

Referenced by slim_fs_learn().

{

```
  int nnz = 0;

  for (int i = 0; i < narray; i ++){
    if (array[i] > EPSILON2 || array[i] < -EPSILON2)
      nnz ++;
  }

  return nnz;

}
```

5.14.2.2 ctrl_t∗ create_ctrl ( )

Create a ctrl structure wich contains all the default parameters for SLIM.

Returns

ctrl_t∗ A pointer to a created ctrl structure

Definition at line 21 of file util.c.

References ctrl_t::beta, ctrl_t::bl, ctrl_t::bsize, ctrl_t::bu, ctrl_t::dbglvl, ctrl_t::endi, ctrl_t::fs, ctrl_t::fs_file, ctrl_t::k, ctrl_t::lambda, ctrl_t::max_bcls_niters, ctrl_t::model_file, ctrl_t::nratings, ctrl_t::optTol, ctrl_t::pred_file, ctrl_t::starti, ctrl_t::test_file, ctrl_t::topn, ctrl_t::train_file, and ctrl_t::transpose.

Referenced by main(), and parse_cmdline().

```
                      {

  ctrl_t * ctrl = gk_malloc(sizeof(ctrl_t), "malloc ctrl");

  ctrl->train_file = NULL;
  ctrl->test_file  = NULL;
  ctrl->model_file = NULL;
  ctrl->fs_file = NULL;
  ctrl->pred_file = NULL;

  ctrl->dbglvl = 0;

  ctrl->beta    = 1.0;
  ctrl->lambda  = 1.0;

  ctrl->starti = -1;
  ctrl->endi = -1;

  ctrl->optTol = 1e-5;
  ctrl->max_bcls_niters = 100000;

  ctrl->bl = 0;
  ctrl->bu = 1e20;

  ctrl->fs = 0;

  ctrl->k = 50;

  ctrl->bsize = 1000;

  ctrl->nratings = 5;

  ctrl->topn = 10;

  ctrl->transpose = 0;

  return ctrl;

}
```

5.14.2.3 void display_timer (ctimer_t ∗ *ctimer,* char ∗ *msg*)

Display a user-defined message and a duration length recorded by a timer.

Parameters

in	*ctimer*	A timer with a length of a duration recorded
in	*msg*	A user-defined message to display

Definition at line 116 of file util.c.

References ctimer_t::end, and ctimer_t::start.

Referenced by slim_learn(), and slim_test().

```
                                                 {
  printf("----- elapsed CPU time for %s: %f s\n", msg,
         (double)(ctimer->end - ctimer->start)/CLOCKS_PER_SEC);
  fflush(stdout);
}
```

5.14.2.4 void end_timer (ctimer_t * *ctimer*)

End a timer to record a length of a duration.

Parameters

in	*ctimer*	A timer to end

Definition at line 101 of file util.c.

References ctimer_t::end.

Referenced by slim_learn(), and slim_test().

```
                                 {
  ctimer->end = clock();
}
```

5.14.2.5 void find_topk (double * *w*, int *n*, int *topk*, double * *map*, int * *topk2*)

Find the top-k values from an array.

Parameters

in	*w*	The array whose top-k values will be found
in	*n*	The length of the array w
in	*topk*	The number of top values to be found
out	*map*	The array of indices that correspond to the top-k values in the input array
out	*topk2*	The actual number of top values that are found

Definition at line 159 of file util.c.

Referenced by slim_fs_learn().

```
                                                                      {

  gk_dkv_t * wkv = gk_malloc(sizeof(gk_dkv_t)*n, "malloc wkv");
  int k2 = 0;

  for (int i = 0; i < n; i ++){
    wkv[i].key = w[i];
    wkv[i].val = i;
    if (w[i] > 1e-10) k2 ++;
  }

  /* sort */
  gk_dkvsortd(n, wkv);

  for (int i = 0; i < ((topk <= k2)? topk:k2); i ++){
    map[i] = wkv[i].val;
  }

  *topk2 = ((topk <= k2)? topk:k2);
  gk_free((void **)&wkv, LTERM);

}
```

5.14.2.6 void free_ctrl (ctrl_t * *ctrl*)

Free a ctrl structure.

Parameters

in	*ctrl*	A pointer to a ctrl structure to be freed

Definition at line 68 of file util.c.

References ctrl_t::fs_file, ctrl_t::model_file, ctrl_t::pred_file, ctrl_t::test_file, and ctrl_t::train_file.

Referenced by main().

```
                            {

  gk_free((void **)&ctrl->model_file, LTERM);
  gk_free((void **)&ctrl->train_file, LTERM);
  gk_free((void **)&ctrl->test_file, LTERM);
  gk_free((void **)&ctrl->pred_file, LTERM);
  gk_free((void **)&ctrl->fs_file, LTERM);

  gk_free((void **)&ctrl, LTERM);

}
```

5.14.2.7 void get_column (gk_csr_t * *constraint,* int *i,* double * *w*)

Get a column from a csr matrix.

Parameters

in	*constraint*	A matrix from which one column is to be retrievd
in	*i*	The index of the column to be retrieved
out	*w*	The output vector which saves the retrieved column

Definition at line 194 of file util.c.

Referenced by slim_learn().

```
                                                          {

  if (i > constraint->ncols){
    gk_dset(constraint->nrows, 0, w);
  }
  else{
    int nnz = constraint->colptr[i+1] - constraint->colptr[i];
    for (int j = 0; j < nnz; j ++){
      int k = *(constraint->colptr[i] + j + constraint->colind);
      w[k]  = *(constraint->colptr[i] + j + constraint->colval);
    }
  }
}
```

5.14.2.8 void start_timer (ctimer_t * *ctimer*)

Start a timer to record current time.

Parameters

in	*ctimer*	A timer to start

Definition at line 88 of file util.c.

References ctimer_t::start.

Referenced by slim_learn(), and slim_test().

```
                                        {
  ctimer->start = clock();
}
```

Index