LIBIRWLS

2.0

Generated by Doxygen 1.8.12

Contents

Chapter 1

LIBIRWLS

Description:

LIBIRWLS is an integrated parallel library for Support Vector Machines (SVMs) that makes use of the IRWLS procedure. It implements the functions to run two different algorithms:

Parallel Iterative Re-Weighted Least Squares: A Parallel SVM solver based on the IRWLS algorithm

Parallel Semi-parametric Iterative Re-Weighted Least Squares: A Parallel Support Vector Machine (SVM) solver based on the IRWLS algorithm

Requirements:

This software is implemented in C and requires the following libraries:

- `OpenMP` To parallelize the software
- A Linear Algebra Package that implements the BLAS and Lapack standard routines, this software has been tested with these libraries (you need just one of them):
  - `BLAS` and `LAPACK`
  - `ATLAS`
  - `MKL`

License: MIT License

Copyright (c) 2015-2016 Roberto Diaz Morales

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, IN↩ CLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

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 model Struct Reference

It represents a trained model that has been obtained using PIRWLS or PSIRWLS.

```
#include <IOStructures.h>
```

Data Fields

- double Kgamma
- int sparse
- int nSVs
- int nElem
- double $*$ weights
- struct svm_sample $**$ x
- double $*$ quadratic_value
- int maxdim
- double bias

4.1.1 Detailed Description

It represents a trained model that has been obtained using PIRWLS or PSIRWLS.

This structures saves all the variables of a trained model needed to classify future data.

4.1.2 Field Documentation

4.1.2.1 bias

```
double model::bias
```

The bias term of the classification function.

4.1.2.2 Kgamma

```
double model::Kgamma
```

Gamma parameter of the kernel function.

4.1.2.3 maxdim

```
int model::maxdim
```

Number of dimensions of the dataset.

4.1.2.4 nElem

```
int model::nElem
```

Number of features distinct than zero in the dataset.

4.1.2.5 nSVs

```
int model::nSVs
```

To tell if the datasets are sparse or not.

4.1.2.6 quadratic_value

```
double* model::quadratic_value
```

Array that contains the norm L2 of every support vector.

4.1.2.7 sparse

```
int model::sparse
```

To tell if the datasets are sparse or not.

4.1.2.8 weights

```
double* model::weights
```

The weight associated to every support vector.

4.1.2.9 x

```
struct svm_sample** model::x
```

The support vectors.

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

- LIBIRWLS/include/IOStructures.h

4.2 predictProperties Struct Reference

Testing parameters of the IRWLS procedures.

```
#include <IOStructures.h>
```

Data Fields

- int Labels
- int Threads

4.2.1 Detailed Description

Testing parameters of the IRWLS procedures.

This struct stores the testing parameters of the IRWLS procedures.

4.2.2 Field Documentation

4.2.2.1 Labels

```
int predictProperties::Labels
```

If the dataset to test is labeled.

4.2.2.2 Threads

```
int predictProperties::Threads
```

Number of threads to make the predictions on the dataset.

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

- LIBIRWLS/include/IOStructures.h

4.3 properties Struct Reference

Training parameters of the IRWLS procedures.

```
#include <IOStructures.h>
```

Data Fields

- double Kgamma
- double C
- int Threads
- int MaxSize
- int size
- double Eta

4.3.1 Detailed Description

Training parameters of the IRWLS procedures.

This struct stores the training parameters of the IRWLS procedures.

4.3.2 Field Documentation

4.3.2.1 C

```
double properties::C
```

C parameter of the SVM cost function.

4.3.2.2 Eta

```
double properties::Eta
```

Convergence criteria of the SVM.

4.3.2.3 Kgamma

```
double properties::Kgamma
```

Gamma parameter of the kernel function.

4.3.2.4 MaxSize

```
int properties::MaxSize
```

Maximum size of the active set to calculate the SVM.

4.3.2.5 size

```
int properties::size
```

Size of semiparametric model (if we are executing the semiparametric version).

4.3.2.6 Threads

```
int properties::Threads
```

Number of threads to parallelize the operations.

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

- LIBIRWLS/include/IOStructures.h

4.4 svm_dataset Struct Reference

A dataset.

```
#include <IOStructures.h>
```

Data Fields

- int l
- int sparse
- int maxdim
- double $*$ y
- struct svm_sample $**$ x
- double $*$ quadratic_value

4.4.1 Detailed Description

A dataset.

This structure represents a dataset, a collection of samples and its associated labels.

4.4.2 Field Documentation

4.4.2.1 l

```
int svm_dataset::l
```

If the dataset is labeled or not.

4.4.2.2 maxdim

```
int svm_dataset::maxdim
```

The number of features of the dataset.

4.4.2.3 quadratic_value

```
double* svm_dataset::quadratic_value
```

The L2 norm of every sample. It is used to compute kernel functions faster.

4.4.2.4 sparse

```
int svm_dataset::sparse
```

If the dataset is sparse or not.

4.4.2.5 x

```
struct svm_sample** svm_dataset::x
```

The samples.

4.4.2.6 y

```
double* svm_dataset::y
```

The label of every sample.

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

- LIBIRWLS/include/IOStructures.h

4.5 svm_sample Struct Reference

A single feature of a data.

```
#include <IOStructures.h>
```

Data Fields

- int index
- double value

4.5.1 Detailed Description

A single feature of a data.

This structure represents a single feature of a data. It is composed of a features index and its value.

4.5.2 Field Documentation

4.5.2.1 index

```
int svm_sample::index
```

The feature index.

4.5.2.2 value

```
double svm_sample::value
```

The feature value.

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

- LIBIRWLS/include/IOStructures.h

Chapter 5

File Documentation

5.1 LIBIRWLS/include/IOStructures.h File Reference

Input and Output structures and procedures.

```
#include <stdio.h>
```

Data Structures

- struct properties

  Training parameters of the IRWLS procedures.
- struct predictProperties

  Testing parameters of the IRWLS procedures.
- struct model

  It represents a trained model that has been obtained using PIRWLS or PSIRWLS.
- struct svm_sample

  A single feature of a data.
- struct svm_dataset

  A dataset.

Typedefs

- typedef struct properties properties

  Training parameters of the IRWLS procedures.
- typedef struct predictProperties predictProperties

  Testing parameters of the IRWLS procedures.
- typedef struct model model

  It represents a trained model that has been obtained using PIRWLS or PSIRWLS.
- typedef struct svm_sample svm_sample

  A single feature of a data.
- typedef struct svm_dataset svm_dataset

  A dataset.

Functions

- svm_dataset readTrainFile (char filename[ ])

  It reads a file that contains a labeled dataset in libsvm format.
- svm_dataset readUnlabeledFile (char filename[ ])

  It reads a file that contains an unlabeled dataset in libsvm format.
- void storeModel (model ∗mod, FILE ∗Output)

  It stores a trained model into a file.
- void readModel (model ∗mod, FILE ∗Input)

  It loads a trained model from a file.
- void writeOutput (char fileoutput[ ], double ∗predictions, int size)

  It writes the content of a double array into a file.

5.1.1 Detailed Description

Input and Output structures and procedures.

Author

Roberto Diaz Morales

Date

23 Aug 2016 Input and Output structures and procedures.

5.1.2 Typedef Documentation

5.1.2.1 model

```
typedef struct model model
```

It represents a trained model that has been obtained using PIRWLS or PSIRWLS.

This structures saves all the variables of a trained model needed to classify future data.

5.1.2.2 predictProperties

```
typedef struct predictProperties predictProperties
```

Testing parameters of the IRWLS procedures.

This struct stores the testing parameters of the IRWLS procedures.

5.1.2.3 properties

```
typedef struct properties properties
```

Training parameters of the IRWLS procedures.

This struct stores the training parameters of the IRWLS procedures.

5.1.2.4 svm_dataset

```
typedef struct svm_dataset svm_dataset
```

A dataset.

This structure represents a dataset, a collection of samples and its associated labels.

5.1.2.5 svm_sample

```
typedef struct svm_sample svm_sample
```

A single feature of a data.

This structure represents a single feature of a data. It is composed of a features index and its value.

5.1.3 Function Documentation

5.1.3.1 readModel()

```
void readModel (
          model * mod,
          FILE * Input )
```

It loads a trained model from a file.

It loads a trained model (that has been obtained using PIRWLS or PSIRWLS) from a file.

Parameters

mod	The pointer with the struct to load results.
Input	The name of the file.

5.1.3.2 readTrainFile()

```
svm_dataset readTrainFile (
          char filename[ ] )
```

It reads a file that contains a labeled dataset in libsvm format.

It reads a file that contains a labeled dataset in libsvm format, the format is the following one: +1 1:5 7:2 15:6 +1 1:5 7:2 15:6 23:1 -1 2:4 3:2 10:6 11:4 ...

Parameters

filename	A string with the name of the file that contains the dataset.

Returns

The struct with the dataset information.

5.1.3.3 readUnlabeledFile()

```
svm_dataset readUnlabeledFile (
          char filename[] )
```

It reads a file that contains an unlabeled dataset in libsvm format.

It reads a file that contains an unlabeled dataset in libsvm format. The format si the following one: 1:5 7:2 15:6 1:5 7:2 15:6 23:1 2:4 3:2 10:6 11:4 ...

Parameters

filename	A string with the name of the file that contains the dataset.

Returns

The struct with the dataset information.

5.1.3.4 storeModel()

```
void storeModel (
          model * mod,
          FILE * Output )
```

It stores a trained model into a file.

It stores the struct of a trained model (that has been obtained using PIRWLS or PSIRWLS) into a file.

Parameters

mod	The struct with the model to store.
Output	The name of the file.

5.1.3.5 writeOutput()

```
void writeOutput (
          char fileoutput[],
          double * predictions,
          int size )
```

It writes the content of a double array into a file.

It writes the content of a double array into a file. It is used to save the predictions of a model on a dataset.

Parameters

fileoutput	The name of the file.
predictions	The array with the information to save.
size	The length of the array.

5.2 LIBIRWLS/include/kernels.h File Reference

Defition of the kernel functions used in the non linear SVM.

```
#include "IOStructures.h"
```

Functions

- double kernel (svm_dataset dataset, int index1, int index2, properties props)

  Radial Basis Function of two elements of the dataset.
- double kernelTest (svm_dataset dataset, int index1, model mymodel, int index2)

  Radial Basis Function of one element of the dataset and Support Vectro of a trained model.

5.2.1 Detailed Description

Defition of the kernel functions used in the non linear SVM.

Author

Roberto Diaz Morales

Date

23 Aug 2016 It defines the kernel function to use in the non linear SVM in this library.

5.2.2 Function Documentation

5.2.2.1 kernel()

```
double kernel (
            svm_dataset dataset,
            int index1,
            int index2,
            properties props )
```

Radial Basis Function of two elements of the dataset.

This function returns the kernel function among two elements of the same dataset.

It returns exp(-gamma||x1-x2||^2)

x1 and x2 are two elements of the dataset and gamma is a parameter whose value can be found in the struct props.

Parameters

dataset	The strut that contains the dataset information.
index1	The index of the first element of the dataset.
index2	The index of the second element of the dataset.
props	The list of properties to extract the kernel parameters.

Returns

The value of the Radial Basis Function of both elements.

5.2.2.2 kernelTest()

```
double kernelTest (
          svm_dataset dataset,
          int index1,
          model mymodel,
          int index2 )
```

Radial Basis Function of one element of the dataset and Support Vectro of a trained model.

This method returns the RBF Kernel function of one element of the dataset and Support Vectro of a trained model.

It returns exp(-gamma||x1-x2||^2)

x1 is an element of the dataset and x2 is a support vector of a trained model, gamma is a parameter whose value can be found in the struct props.

Parameters

dataset	The strut that contains the dataset information.
index1	The index of the sample of the dataset.
mymodel	The trained SVM model.
index2	The index of one of the Support Vectors of the trained model.

Returns

The value of the Radial Basis Function of both elements.

5.3 LIBIRWLS/include/LIBIRWLS-predict.h File Reference

Functions to classify data with a trained model.

```
#include "IOStructures.h"
```

Functions

- double ∗ test (svm_dataset dataset, model mymodel, predictProperties props)

  Function to classify data in a labeled dataset and to obtain the accuracy.
- void printPredictInstructions ()

  Print instructions.
- predictProperties parsePredictParameters (int ∗argc, char ∗∗∗argv)

  It parses the prediction parameters from the command line.

5.3.1 Detailed Description

Functions to classify data with a trained model.

Author

Roberto Diaz Morales

Date

23 Aug 2016 Functions to classify data with a trained model.

5.3.2 Function Documentation

5.3.2.1 parsePredictParameters()

```
predictProperties parsePredictParameters (
            int * argc,
            char *** argv )
```

It parses the prediction parameters from the command line.

It reads the command line, extract the parameters and creates a strict with the value of its values.

Parameters

argc	The number of words of the command line.
argv	The list of words of the command line.

Returns

A struct that contains the values of the test parameters.

5.3.2.2 printPredictInstructions()

```
void printPredictInstructions ( )
```

Print instructions.

It shows the command line instructions in the standard output.

5.3.2.3 test()

```
double∗ test (
            svm_dataset dataset,
            model mymodel,
            predictProperties props )
```

Function to classify data in a labeled dataset and to obtain the accuracy.

Function that uses a trained model on a dataset and obtains the class of every training sample.

Parameters

dataset	The test set.
mymodel	A trained SVM model.
props	The test properties.

Returns

The output of the classifier for every test sample (soft output).

5.4 LIBIRWLS/include/ParallelAlgorithms.h File Reference

Functions to perform parallel linear algebra tasks.

Functions

- void initMemory (int Threads, int size)

  Function to allocate auxiliar memory to be used in the algebra operations.
- void updateMemory (int Threads, int size)

  Function to update auxiliar memory to be used in the algebra operations.
- void getSubMatrix (double ∗matrix, int size1, int size2, int O1, int O2, double ∗A, int size3, int size4, int nCores)

  This function saves a piece of a matrix in the auxiliar memory.
- void putSubMatrix (double ∗matrix, int size1, int size2, int O1, int O2, double ∗A, int size3, int size4, int nCores)

  This function loads a piece of a matrix in the auxiliar memory and storage it in a matrix.
- void ParallelChol (double ∗matrix, int r, int c, int ro, int co, int n, int nCores, int deep)

  Main function that performs a parallel cholesky factorization.
- void ParallelLinearSystem (double ∗matrix1, int r1, int c1, int ro1, int co1, double ∗matrix2, int r2, int c2, int ro2, int co2, int n, int m, double ∗result, int rr, int cr, int ror, int cor, int nCores)

  Main function to solve a linear system in parallel.
- void ParallelVectorMatrixT (double ∗m1, int size, double ∗m2, double ∗result, int numThreads)

  This function performs a product of a vector and the transpose of a square matrix in parallel.
- void ParallelVectorMatrix (double ∗m1, int size, double ∗m2, double ∗result, int numThreads)

  This function performs a product of a vector and a square matrix in parallel.

5.4.1 Detailed Description

Functions to perform parallel linear algebra tasks.

Parallel procedures to solve linear systems, cholesky factorization, matrix products or triangular matrix inversion.

This library also contains many auxiliar functions that are not detailed here with the goal of obtained a readable documentation. Their are only designed to be called by the main functions described here. However, that functions are detailed described in the source code.

Author

Roberto Diaz Morales

Date

23 Aug 2016

5.4.2 Function Documentation

5.4.2.1 getSubMatrix()

```
void getSubMatrix (
          double * matrix,
          int size1,
          int size2,
          int O1,
          int O2,
          double * A,
          int size3,
          int size4,
          int nCores )
```

This function saves a piece of a matrix in the auxiliar memory.

This is an auxiliry function of the linear algebra funtions. It is used to save a submatrix of a matrix given as a parameter in the auxiliar memory.

Parameters

matrix	The original matrix.
size1	The number of rows fo the original matrix.
size2	The number of columns of the original matrix.
O1	The row where the submatrix starts.
O2	The column where the submatrix starts.
A	The pointer where the submatrix will be storaged.
size3	Number of rows of the submatrix to storage.
size4	Number of columns of the submatrix to storage.
nCores	Number of threads to perform this task.

5.4.2.2 initMemory()

```
void initMemory (
            int Threads,
            int size )
```

Function to allocate auxiliar memory to be used in the algebra operations.

The parallel lineal algebra functions of this module require some memory for every thead to allocate temporal results. This function must be called before any other function.

Parameters

Threads	The number of threads to parallelize the linear algebra functions.
size	The size of the dimensions of the matrices that will be handle. If a matrix has a rows and b columns, then n=max(a,b).

See also

updateMemory()

5.4.2.3 ParallelChol()

```
void ParallelChol (
            double * matrix,
            int r,
            int c,
            int ro,
            int co,
            int n,
            int nCores,
            int deep )
```

Main function that performs a parallel cholesky factorization.

This function performs a parallel cholesky factorization on a sqaure submatrix of a matrix recived as an argument. It uses openmp to create to parallelize the task using different threads and every one of them performs a subtask using the function Chol.

Parameters

matrix	The matrix to perform the cholesky factorization.
r	The number of rows of matrix
c	The number of columns of matrix.
ro	The row where the submatrix starts.
co	The column where the submatrix starts.
n	The order of the square submatrix
nCores	The number of threads to perform the task.
deep	It is a recursive function, this parameter tell the recursion deep to use.

See also

Chol()

5.4.2.4 ParallelLinearSystem()

```
void ParallelLinearSystem (
          double * matrix1,
          int r1,
          int c1,
          int ro1,
          int co1,
          double * matrix2,
          int r2,
          int c2,
          int ro2,
          int co2,
          int n,
          int m,
          double * result,
          int rr,
          int cr,
          int ror,
          int cor,
          int nCores )
```

Main function to solve a linear system in parallel.

This function solves a linear system of two submatrices of matrix1 and matrix2. It uses openmp to create different threads and every one of them performs a subtask using the function LinearSystem. If submatrix1 is the submatrix of matrix1 and submatrix2 is the submatrix of matrix2 then it performs (submatrix1)^-1∗submatrix2.

Parameters

matrix1	The first matrix.
r1	The number of rows of matrix1.
c1	The number of columns of matrix1.
ro1	The row where the submatrix starts.
co1	The column where the submatrix starts.
matrix2	The second matrix.
r2	The number of rows of matrix2.
c2	The number of columns of matrix2.
ro2	The row where the second submatrix starts.
co2	The column where the second submatrix starts.
n	The number of rows of the submatrix1
m	The number of columns of the submatrix2.
result	The matrix where the result should be storaged.
rr	The number of rows of matrix result.
cr	The number of columns of matrix result.
ror	The row where the result submatrix starts.
cor	The column where the result submatrix starts.
nCores	The number of threads to launch to solve the task.

5.4.2.5 ParallelVectorMatrix()

```
void ParallelVectorMatrix (
          double * m1,
          int size,
          double * m2,
          double * result,
          int numThreads )
```

This function performs a product of a vector and a square matrix in parallel.

This function performs a product of a vector and a matrix in parallel.

This function performs a vector matrix product in parallel.

Parameters

m1	The vector.
size	The length of the vector and the order of the square matrix.
m2	The square matrix.
result	The matrix to storage the result.
numThreads	Number of threads to pefrom this task.

5.4.2.6 ParallelVectorMatrixT()

```
void ParallelVectorMatrixT (
          double * m1,
          int size,
          double * m2,
          double * result,
          int numThreads )
```

This function performs a product of a vector and the transpose of a square matrix in parallel.

This function performs a product of a vector and the transpose of a matrix in parallel.

Parameters

m1	The vector.
size	The length of the vector and the order of the square matrix.
m2	The square matrix.
result	The matrix to storage the result.
numThreads	Number of threads to pefrom this task.

5.4.2.7 putSubMatrix()

```
void putSubMatrix (
          double * matrix,
          int size1,
          int size2,
          int O1,
          int O2,
          double * A,
          int size3,
          int size4,
          int nCores )
```

This function loads a piece of a matrix in the auxiliar memory and storage it in a matrix.

This is an auxiliry function of the linear algebra funtions. It is used to load a matrix from the auxiliar memory and storage it as a submatrix of another matrix.

Parameters

matrix	The original matrix where the submatrix will be allocated.
size1	The number of rows fo the original matrix.
size2	The number of columns of the original matrix.
O1	The row of the original matrix where the submatrix starts.
O2	The column of the original matrix where the submatrix starts.
A	The pointer where the submatrix is storaged.
size3	Number of rows of the submatrix that is storaged.
size4	Number of columns of the submatrix that is storaged.
nCores	Number of threads to perform this task.

5.4.2.8 updateMemory()

```
void updateMemory (
            int Threads,
            int size )
```

Function to update auxiliar memory to be used in the algebra operations.

The parallel lineal algebra functions of this module require some memory for every thead to allocate temporal results. This function must be called if we allocated the memory using the function initMemory and we are going to work with bigger matrices.

Parameters

Threads	The number of threads to parallelize the linear algebra functions.
size	The size of the dimensions of the matrices that will be handle. If a matrix has a rows and b columns, then n=max(a,b).

See also

initMemory()

5.5 LIBIRWLS/include/PIRWLS-train.h File Reference

Functions to train a full SVM using the IRWLS algorithm.

```
#include "IOStructures.h"
```

Functions

- int ∗ rpermute (int n)

  Random permutation of n elements.
- double ∗ subIRWLS (svm_dataset dataset, properties props, double ∗GIN, double ∗e, double ∗beta)

  IRWLS procedure on a Working Set.
- double ∗ trainFULL (svm_dataset dataset, properties props)

  It trains a full SVM with a training set.
- void printPIRWLSInstructions ()

  Print Instructions.
- properties parseTrainPIRWLSParameters (int ∗argc, char ∗∗∗argv)

  It parses the command line.
- model calculatePIRWLSModel (properties props, svm_dataset dataset, double ∗beta)

  It converts the result into a model struct.

5.5.1 Detailed Description

Functions to train a full SVM using the IRWLS algorithm.

For a detailed description of the algorithm and its parameters read the following paper:
Pérez-Cruz, F., Alarcón-Diana, P. L., Navia-Vázquez, A., & Artés-Rodríguez, A. (2001). Fast Training of Support Vector Classifiers. In Advances in Neural Information Processing Systems (pp. 734-740)

For a detailed description about the parallelization read the following paper:
Díaz-Morales, R., & Navia-Vázquez, Á. (2016). Efficient parallel implementation of kernel methods. Neurocomputing, 191, 175-186.

Author

Roberto Diaz Morales

Date

23 Aug 2016

5.5.2 Function Documentation

5.5.2.1 calculatePIRWLSModel()

```
model calculatePIRWLSModel (
          properties props,
          svm_dataset dataset,
          double * beta )
```

It converts the result into a model struct.

After the training of a SVM using the IRWLS procedure, this function build a struct with the information and returns it.

Parameters

props	The training parameters.
dataset	The training set.
beta	The weights of the classifier.

Returns

The struct that storages all the information of the classifier.

5.5.2.2 parseTrainPIRWLSParameters()

```
properties parseTrainPIRWLSParameters (
            int * argc,
            char *** argv )
```

It parses the command line.

It parses input command line to extract the parameters.

Parameters

argc	The number of words of the command line.
argv	The list of words of the command line.

Returns

A struct that contains the values of the training parameters.

5.5.2.3 printPIRWLSInstructions()

```
void printPIRWLSInstructions ( )
```

Print Instructions.

It shows PIRWLS-train command line instructions in the standard output.

5.5.2.4 rpermute()

```
int* rpermute (
            int n )
```

Random permutation of n elements.

It crates a random permutation of n elements.

Parameters

n	The number of elementos in the permutation.

Returns

The permutation.

5.5.2.5 subIRWLS()

```
double* subIRWLS (
            svm_dataset dataset,
            properties props,
            double * GIN,
            double * e,
            double * beta )
```

IRWLS procedure on a Working Set.

For a detailed description of the algorithm and its parameters read the following paper:
Pérez-Cruz, F., Alarcón-Diana, P. L., Navia-Vázquez, A., & Artés-Rodríguez, A. (2001). Fast Training of Support Vector Classifiers. In Advances in Neural Information Processing Systems (pp. 734-740)

For a detailed description about the parallelization read the following paper:
Díaz-Morales, R., & Navia-Vázquez, Á. (2016). Efficient parallel implementation of kernel methods. Neurocomputing, 191, 175-186.

Parameters

dataset	The training dataset.
props	The strut of training properties.
GIN	The classification effect of the inactive set.
e	The current error on every training data.
beta	The bias term of the classification function.

Returns

The new weights vector of the classifier.

5.5.2.6 trainFULL()

```
double* trainFULL (
            svm_dataset dataset,
            properties props )
```

It trains a full SVM with a training set.

For a detailed description of the algorithm and its parameters read the following paper:
Pérez-Cruz, F., Alarcón-Diana, P. L., Navia-Vázquez, A., & Artés-Rodríguez, A. (2001). Fast Training of Support Vector Classifiers. In Advances in Neural Information Processing Systems (pp. 734-740)

For a detailed description about the parallelization read the following paper:
Díaz-Morales, R., & Navia-Vázquez, Á. (2016). Efficient parallel implementation of kernel methods. Neurocomputing, 191, 175-186.

It trains a full SVM using a training set and the training parameters.

Parameters

dataset	The training set.
props	The values of the training parameters.

Returns

The weights of every Support Vector of the SVM.

5.6 LIBIRWLS/include/PSIRWLS-train.h File Reference

Functions to train a semi parametric SVM using the IRWLS algorithm.

```
#include "IOStructures.h"
```

Functions

- int ∗ SGMA (svm_dataset dataset, properties props)

    Sparse Greedy Matrix Approximation algorithm.
- double ∗ IRWLSpar (svm_dataset dataset, int ∗indexes, properties props)

    Iterative Re-Weighted Least Squares Algorithm.
- void printPSIRWLSInstructions ()

    Print Instructions.
- properties parseTrainParameters (int ∗argc, char ∗∗∗argv)

    It parses input command line to extract the parameters of the PSIRWLS algorithm.
- model calculatePSIRWLSModel (properties props, svm_dataset dataset, int ∗centroids, double ∗beta)

    It converts the result of the PSIRWLS algorithm into a model struct.

5.6.1 Detailed Description

Functions to train a semi parametric SVM using the IRWLS algorithm.

Author

Roberto Diaz Morales

Date

23 Aug 2016 Functions to train a semi parametric SVM using the IRWLS algorithm.

5.6.2 Function Documentation

5.6.2.1 calculatePSIRWLSModel()

```
model calculatePSIRWLSModel (
            properties props,
            svm_dataset dataset,
            int * centroids,
            double * beta )
```

It converts the result of the PSIRWLS algorithm into a model struct.

It converts the result into a model struct.

Parameters

props	The training parameters.
dataset	The training set.
centroids	of the selected centroids by the SGMA algorithm.
beta	The weights of every centroid obtained with the IRWLS algorithm.

Returns

The struct that storages all the information of the classifier.

It converts the result of the PSIRWLS algorithm into a model struct.

After the training of a SVM using the PSIRWLS procedure, this function build a struct with the information and returns it.

Parameters

props	The training parameters.
dataset	The training set.
centroids	of the selected centroids by the SGMA algorithm.
beta	The weights of every centroid obtained with the IRWLS algorithm.

Returns

The struct that storages all the information of the classifier.

5.6.2.2 IRWLSpar()

```
double* IRWLSpar (
            svm_dataset dataset,
            int * indexes,
            properties props )
```

Iterative Re-Weighted Least Squares Algorithm.

IRWLS procedure to obtain the weights of the semi parametric model.

Parameters

dataset	The training set.
indexes	The indexes of the centroids selected by the SGMA algorithm.
props	The struct with the training parameters.

Returns

The weights of every centroid.

IRWLS procedure to obtain the weights of the semi parametric model. For a detailed description of the algorithm and parallelization:

Díaz-Morales, R., & Navia-Vázquez, Á. (2016). Efficient parallel implementation of kernel methods. Neurocomputing, 191, 175-186.

Parameters

dataset	The training set.
indexes	The indexes of the centroids selected by the SGMA algorithm.
props	The struct with the training parameters.

Returns

The weights of every centroid.

5.6.2.3 parseTrainParameters()

```
properties parseTrainParameters (
            int * argc,
            char *** argv )
```

It parses input command line to extract the parameters of the PSIRWLS algorithm.

It parses input command line to extract the parameters.

Parameters

argc	The number of words of the command line.
argv	The list of words of the command line.

Returns

A struct that contains the values of the training parameters of the PSIRWLS algorithm.

5.6.2.4 printPSIRWLSInstructions()

```
void printPSIRWLSInstructions ( )
```

Print Instructions.

It shows PSIRWLS-train command line instructions in the standard output.

Print Instructions.

It shows PSIRWLS-train command line instructions in the standard output.

5.6.2.5 SGMA()

```
int* SGMA (
          svm_dataset dataset,
          properties props )
```

Sparse Greedy Matrix Approximation algorithm.

Sparse Greedy Matrix Approximation algorithm to select the basis elements of the semi parametric model.

Parameters

dataset	The training set.
props	The struct with the training parameters.

Sparse Greedy Matrix Approximation algorithm to select the basis elements of the semi parametric model. For a detailed description read:

Díaz-Morales, R., & Navia-Vázquez, Á. (2016). Efficient parallel implementation of kernel methods. Neurocomputing, 191, 175-186.

Parameters

dataset	The training set.
props	The struct with the training parameters.

5.7 LIBIRWLS/src/IOStructures.c File Reference

Implementation of the IO functions used in the application.

```
#include "../include/IOStructures.h"
#include <stdlib.h>
#include <math.h>
#include <string.h>
```

5.7.1 Detailed Description

Implementation of the IO functions used in the application.

It implements the interface defined by IOStructures.h. See IOStructures.h for a detailed description of functions and parameters.

Author

Roberto Diaz Morales

Date

23 Aug 2016

See also

IOStructures.h

5.8 LIBIRWLS/src/kernels.c File Reference

Implementation of the kernel functions used in the non linear SVM.

```
#include "../include/kernels.h"
#include "../include/IOStructures.h"
#include <math.h>
```

5.8.1 Detailed Description

Implementation of the kernel functions used in the non linear SVM.

It implements the interfaz defined by kernel.h with the kernel functions to use in the non linear SVM in this library. See kernels.h for a detailed description of its functions.

Author

Roberto Diaz Morales

Date

23 Aug 2016

See also

kernels.h

5.9 LIBIRWLS/src/LIBIRWLS-predict.c File Reference

Implementation of the predictions functions to classify data using a trained model.

```
#include <omp.h>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>
#include <sys/time.h>
#include <string.h>
#include "cblas.h"
#include "../include/kernels.h"
```

5.9.1 Detailed Description

Implementation of the predictions functions to classify data using a trained model.

See LIBIRWLS-predict.h for a detailed description of its functions and parameters.

Author

Roberto Diaz Morales

Date

23 Aug 2016

See also

LIBIRWLS-predict.h

5.10 LIBIRWLS/src/ParallelAlgorithms.c File Reference

Functions to perform some parallel linear algebra tasks.

```
#include "../include/ParallelAlgorithms.h"
#include <omp.h>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include "cblas.h"
```

5.10.1 Detailed Description

Functions to perform some parallel linear algebra tasks.

Parallel procedures to solve linear systems, cholesky factorization, matrix products or triangular matrix inversion.

Author

Roberto Diaz Morales

Date

23 Aug 2016

See also

ParallelAlgorithms.h

5.11 LIBIRWLS/src/PIRWLS-train.c File Reference

Implementation of the training functions of the PIRWLS algorithm.

```
#include <omp.h>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>
#include <sys/time.h>
#include <string.h>
#include "cblas.h"
#include "../include/PIRWLS-train.h"
```

5.11.1 Detailed Description

Implementation of the training functions of the PIRWLS algorithm.

See PIRWLS-train.h for a detailed description of its functions and parameters.

For a detailed description of the algorithm and its parameters read the following paper:
Pérez-Cruz, F., Alarcón-Diana, P. L., Navia-Vázquez, A., & Artés-Rodríguez, A. (2001). Fast Training of Support Vector Classifiers. In Advances in Neural Information Processing Systems (pp. 734-740)

For a detailed description about the parallelization read the following paper:
Díaz-Morales, R., & Navia-Vázquez, Á. (2016). Efficient parallel implementation of kernel methods. Neurocomputing, 191, 175-186.

Author

Roberto Diaz Morales

Date

23 Aug 2016

See also

PIRWLS-train.h

5.12 LIBIRWLS/src/PSIRWLS-train.c File Reference

Implementation of the training functions of the PSIRWLS algorithm.

```
#include <omp.h>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>
#include <sys/time.h>
#include <string.h>
#include "cblas.h"
#include "../include/PSIRWLS-train.h"
```

Functions

- int ∗ SGMA (svm_dataset dataset, properties props)

  Sparse Greedy Matrix Approximation algorithm.
- double ∗ IRWLSpar (svm_dataset dataset, int ∗indexes, properties props)

  Iterative Re-Weighted Least Squares Algorithm.
- model calculatePSIRWLSModel (properties props, svm_dataset dataset, int ∗centroids, double ∗beta)

  It converts the result into a model struct.
- properties parseTrainParameters (int ∗argc, char ∗∗∗argv)

  It parses input command line to extract the parameters of the PSIRWLS algorithm.
- void printPSIRWLSInstructions ()

  Print Instructios.
- int main (int argc, char ∗∗argv)

  Is the main function to build the executable file to train a SVM using the PSIRWLS procedure.

5.12.1 Detailed Description

Implementation of the training functions of the PSIRWLS algorithm.

See PSIRWLS-train.h for a detailed description of its functions and parameters.

For a detailed description of the algorithm and its parameters read the following paper:

Díaz-Morales, R., & Navia-Vázquez, Á. (2016). Efficient parallel implementation of kernel methods. Neurocomputing, 191, 175-186.

Author

Roberto Diaz Morales

Date

23 Aug 2016

See also

PSIRWLS-train.h

5.12.2 Function Documentation

5.12.2.1 calculatePSIRWLSModel()

```
model calculatePSIRWLSModel (
            properties props,
            svm_dataset dataset,
            int * centroids,
            double * beta )
```

It converts the result into a model struct.

It converts the result of the PSIRWLS algorithm into a model struct.

After the training of a SVM using the PSIRWLS procedure, this function build a struct with the information and returns it.

Parameters

props	The training parameters.
dataset	The training set.
centroids	of the selected centroids by the SGMA algorithm.
beta	The weights of every centroid obtained with the IRWLS algorithm.

Returns

The struct that storages all the information of the classifier.

5.12.2.2 IRWLSpar()

```
double* IRWLSpar (
            svm_dataset dataset,
            int * indexes,
            properties props )
```

Iterative Re-Weighted Least Squares Algorithm.

IRWLS procedure to obtain the weights of the semi parametric model. For a detailed description of the algorithm and parallelization:

Díaz-Morales, R., & Navia-Vázquez, Á. (2016). Efficient parallel implementation of kernel methods. Neurocomputing, 191, 175-186.

Parameters

dataset	The training set.
indexes	The indexes of the centroids selected by the SGMA algorithm.
props	The struct with the training parameters.

Returns

The weights of every centroid.

5.12.2.3 parseTrainParameters()

```
properties parseTrainParameters (
            int * argc,
            char *** argv )
```

It parses input command line to extract the parameters of the PSIRWLS algorithm.

It parses input command line to extract the parameters.

Parameters

argc	The number of words of the command line.
argv	The list of words of the command line.

Returns

A struct that contains the values of the training parameters of the PSIRWLS algorithm.

5.12.2.4 printPSIRWLSInstructions()

```
void printPSIRWLSInstructions ( )
```

Print Instructios.

Print Instructions.

It shows PSIRWLS-train command line instructions in the standard output.

5.12.2.5 SGMA()

```
int* SGMA (
            svm_dataset dataset,
            properties props )
```

Sparse Greedy Matrix Approximation algorithm.

Sparse Greedy Matrix Approximation algorithm to select the basis elements of the semi parametric model. For a detailed description read:

Díaz-Morales, R., & Navia-Vázquez, Á. (2016). Efficient parallel implementation of kernel methods. Neurocomputing, 191, 175-186.

Parameters

dataset	The training set.
props	The struct with the training parameters.

Index