BudgetedSVM

Generated by Doxygen 1.8.2

Mon May 6 2013 20:43:25

Contents

Chapter 1

BudgetedSVM Documentation

Thank you for using BudgetedSVM, a toolbox for training large-scale, non-linear classifiers. The toolbox implements the following four SVM/SVM-like algorithms for large-scale, non-linear classification:

- Pegasos (*Shalev-Shwartz, S., Singer, Y., Srebro, N. Pegasos: Primal Estimated sub-GrAdient SOlver for SVM*, ICML, 2007)
- AMM batch and AMM online (*Wang, Z., Djuric, N., Crammer, K., Vucetic, S., Trading Representability for Scalability: Adaptive Multi-Hyperplane Machine for Nonlinear Classification*, KDD, 2011)
- BSGD (*Wang, Z., Crammer, K., Vucetic, S., Breaking the Curse of Kernelization: Budgeted Stochastic Gradient Descent for Large-Scale SVM Training*, JMLR, 2012)
- LLSVM (*Zhang, K., Lan, L., Wang, Z., and Moerchen, F., Scaling up Kernel SVM on Limited Resources: A Low-rank Linearization Approach*, AISTATS, 2012)

Please report any comments/bugs/praises to nemanja.djuric@temple.edu. We hope you will find this toolbox useful!

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 3

Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 4

File Index

4.1 File List

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

Chapter 5

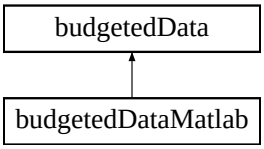
Class Documentation

5.1 budgetedData Class Reference

Class which handles manipulation of large data sets that cannot be fully loaded to memory (using a data structure similar to Matlab's sparse matrix structure).

`#include <budgetedSVM.h>`

Inheritance diagram for budgetedData:



Public Member Functions

- double getSparsity (void)

  Get the sparsity of the data set (i.e., percentage of non-zero features). It is a number between 0 and 100, showing the sparsity in percentage points.
- unsigned int getNumLoadedDataPointsSoFar (void)

  Get total number of data points loaded since the beginning of the epoch.
- budgetedData (bool keepAssignments=false, vector< int > *yLabels=NULL)

  Vanilla constructor, just initializes the variables.
- budgetedData (const char fileName[], unsigned int dimension, unsigned int chunkSize, bool keepAssignments=false, vector< int > *yLabels=NULL)

  Constructor that takes the data from LIBSVM-style .txt file.
- virtual ~budgetedData (void)

  Destructor, cleans up the memory.
- void saveAssignment (unsigned int *assigns)

  Saves the current assignments, used by AMM batch.
- void readChunkAssignments (bool endOfFile)

  Reads assignments for the current chunk, used by AMM batch.
- void flushData (void)

  Clears all data taken up by the current chunk.
- virtual bool readChunk (unsigned int size, bool assign=false)

  Reads the next data chunk.
- float getElementOfVector (unsigned int vector, unsigned int element)

Returns an element of a vector stored in budgetedData structure.

- long double getVectorSqrL2Norm (unsigned int vector, parameters *param)

  Returns a squared L2-norm of a vector stored in budgetedData structure.

- double distanceBetweenTwoPoints (unsigned int index1, unsigned int index2)

  Computes Euclidean distance between two data points from the input data.

Public Attributes

- unsigned long loadTime

  Measures the time spent to load the data.

- vector< float > an

  Vector of non-zero features of data points of the current data chunk. Where the data points start and end in this vector is specified by ai vector.

- vector< unsigned int > aj

  Vector of indices of non-zero features of data points of the current data chunk. Where the data points start and end in this vector is specified by ai vector.

- vector< unsigned int > ai

  Vector that tells us where the data point starts in vectors an and aj, always of length N.

- unsigned char * al

  Array of labels of the current data chunk, always of length N.

- vector< int > yLabels

  Vector of possible labels, either found during loading or initialized during testing phase by the learned model.

- unsigned int N

  Number of data points loaded.

- unsigned int * assignments

  Assignments for the current data chunk, used for AMM batch algorithm.

Protected Attributes

- FILE * ifile

  Pointer to a FILE object that identifies input data stream.

- FILE * fAssignFile

  Pointer to a FILE object that identifies data stream of current assignments, used for AMM batch algorithm.

- const char * ifileName

  Filename of LIBSVM-style .txt file with input data.

- const char * ifileNameAssign

  Filename of .txt file that keeps current assignments of weights to input data points, used for AMM batch algorithm.

- unsigned int dimension

  Dimensionality of the input data, does not include one additional dimension due to possible non-zero bias term.

- unsigned int dimensionHighestSeen

  Highest dimension seen during loading of the data, should always be smaller than the set dimension. Used to detect badly specified data dimension.

- unsigned int numNonZeroFeatures

  Number of non-zero features of the currently loaded chunk, found during loading of the data. Used to compute the sparsity of the data.

- unsigned int loadedDataPointsSoFar

  Total number of data points loaded so far.

- bool fileOpened

  Indicates that the input data .txt file is open.

- bool fileAssignOpened

  Indicates that the .txt file with current assignments is open, used for AMM batch algorithm.

- bool dataPartiallyLoaded

  Indicates that the data is only partially loaded to memory. It can also be fully loaded, e.g., when using data already loaded by some other application, Matlab for instance.
- bool keepAssignments

  Indicates that assignments should be kept, true only for AMM batch algorithm.

5.1.1 Detailed Description

Class which handles manipulation of large data sets that cannot be fully loaded to memory (using a data structure similar to Matlab's sparse matrix structure).

In order to handle large data sets, we do not load the entire data into memory, instead load it in smaller chunks. The loaded chunk is stored in a structure similar to Matlab's sparse matrix structure. Namely, only non-zero features and corresponding feature values of data points are stored in one budget vector for fast access, with additional vector that hold pointers to feature vector telling us where the information for each individual data point starts.

5.1.2 Constructor & Destructor Documentation

5.1.2.1 budgetedData::budgetedData (bool *keepAssignments* = `false`, vector< int > * *yLabels* = `NULL`)

Vanilla constructor, just initializes the variables.

Parameters

in	*keepAssignments*	True for AMM batch, otherwise false. File 'temp_assigns.txt' will be created and deleted to keep the assignments.
in	*yLabels*	Possible labels in the classification problem, for training data is NULL since they are inferred from data.

5.1.2.2 budgetedData::budgetedData (const char *fileName[]*, unsigned int *dimension*, unsigned int *chunkSize*, bool *keepAssignments* = `false`, vector< int > * *yLabels* = `NULL`)

Constructor that takes the data from LIBSVM-style .txt file.

Parameters

in	*fileName*	Path to the input .txt file.
in	*dimension*	Dimensionality of the classification problem.
in	*chunkSize*	Size of the input data chunk that is loaded.
in	*keepAssignments*	True for AMM batch, otherwise false. File 'temp_assigns.txt' will be created and deleted to keep the assignments.
in	*yLabels*	Possible labels in the classification problem, for training data is NULL since inferred from data.

5.1.3 Member Function Documentation

5.1.3.1 double budgetedData::distanceBetweenTwoPoints (unsigned int *index1*, unsigned int *index2*)

Computes Euclidean distance between two data points from the input data.

Parameters

in	*index1*	Index of the first data point.
in	*index2*	Index of the second data point.

Returns

Euclidean distance between the two points.

5.1.3.2 float budgetedData::getElementOfVector (unsigned int *vector*, unsigned int *element*)

Returns an element of a vector stored in budgetedData structure.

Parameters

in	*vector*	Index of the vector (C-style indexing used, starting from 0; note that LibSVM format indices start from 1).
in	*element*	Index of the element of the vector (C-style indexing used, starting from 0; note that LibSVM format indices start from 1).

Returns

Element of the vector specified as an input.

In the case that we need to read an element of a vector from currently loaded data chunk, we can use this function to access these vector elements.

5.1.3.3 unsigned int budgetedData::getNumLoadedDataPointsSoFar (void) [inline]

Get total number of data points loaded since the beginning of the epoch.

Returns

Number of data points loaded since the beginning of the epoch.

5.1.3.4 double budgetedData::getSparsity (void) [inline]

Get the sparsity of the data set (i.e., percentage of non-zero features). It is a number between 0 and 100, showing the sparsity in percentage points.

Returns

Returns the sparsity of the data set in percentage points.

5.1.3.5 long double budgetedData::getVectorSqrL2Norm (unsigned int *vector*, parameters * *param*)

Returns a squared L2-norm of a vector stored in budgetedData structure.

Parameters

in	*vector*	Index of the vector (C-style indexing used, starting from 0; note that LibSVM format indices start from 1).
in	*param*	The parameters of the algorithm.

Returns

Squared L2-norm of a vector.

This function returns squared L2-norm of a vector stored in the budgetedData structure. In particular, it is used to speed up the computation of Gaussian kernel.

5.1.3.6 bool budgetedData::readChunk (unsigned int *size*, bool *assign* = false) [virtual]

Reads the next data chunk.

Parameters

in	*size*	Size of the chunk (i.e., number of data points) to be loaded.
in	*assign*	True if assignments should be saved, false otherwise.

Returns

True if just read the last data chunk, false otherwise.

In order to handle large data sets, we do not load the entire data into memory, instead load it in smaller chunks. Once we have finished processing a loaded data chunk, we load a new one using this function. The return value tells us if there are more chunks left; while there is still data to be loaded the function returns false, if we are done with the data set the function returns true. In the case of the AMM_batch algorithm, we also need to store current assignments of data points to weights, if the input "assign" is true then the function also initializes a .txt file for purpose of storing these assignments when the first chunk is loaded.

Reimplemented in budgetedDataMatlab.

5.1.3.7 void budgetedData::readChunkAssignments (bool *endOfFile*)

Reads assignments for the current chunk, used by AMM batch.

Parameters

in	*endOfFile*	If the final chunk, close the assignment file.

During AMM batch training phase we need to keep track of the assignment of non-zero weights to data points. We store the assignments into a text file and load them together with the data chunk currently loaded, as it may be to expensive to store all assignments in memory when working with large data sets.

5.1.3.8 void budgetedData::saveAssignment (unsigned int * *assigns*)

Saves the current assignments, used by AMM batch.

Parameters

in	*assigns*	Current assignments.

5.1.4 Member Data Documentation

5.1.4.1 unsigned int * budgetedData::assignments

Assignments for the current data chunk, used for AMM batch algorithm.

See Also

fAssignFile

5.1.4.2 long budgetedData::dimension [protected]

Dimensionality of the input data, does not include one additional dimension due to possible non-zero bias term.

See Also

parameters::BIAS_TERM

5.1.4.3 FILE * budgetedData::fAssignFile [protected]

Pointer to a FILE object that identifies data stream of current assignments, used for AMM batch algorithm.

During AMM batch training phase we need to keep track of which non-zero weight is assigned to which data point. We store the assignments into text file and load them together with the data chunk currently loaded, as it might be to expensive to store all assignments in memory. In order to keep track of this weight-example mapping, each weight vector also has a unique budgetedVector::weightID, assigned to each vector upon creation.

See Also

parameters::CHUNK_SIZE
budgetedVector::weightID

5.1.4.4 const char * budgetedData::ifileNameAssign [protected]

Filename of .txt file that keeps current assignments of weights to input data points, used for AMM batch algorithm.

During AMM batch training phase we need to keep track of which non-zero weight is assigned to which data point. We store the assignments into text file and load them together with the data chunk currently loaded, as it might be to expensive to store all assignments in memory.

See Also

parameters::CHUNK_SIZE

The documentation for this class was generated from the following files:

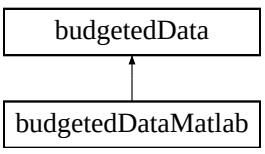
- budgetedSVM.h
- budgetedSVM.cpp

5.2 budgetedDataMatlab Class Reference

Class which manipulates sparse array of vectors (similarly to Matlab sparse matrix structure), with added functionality to load data directly from Matlab.

`#include <budgetedSVM_matlab.h>`

Inheritance diagram for budgetedDataMatlab:



Public Member Functions

- bool readChunk (unsigned int size, bool assign=false)

  Overrides virtual function from budgetedData, simply returns false regardless of inputs as the data is fully loaded from Matlab.

- budgetedDataMatlab (const mxArray *labelVec, const mxArray *instanceMat, parameters *param, bool keepAssignments=false, vector< int > *yLabels=NULL)

  Constructor, invokes readDataFromMatlab that loads Matlab data.
- ~budgetedDataMatlab (void)

  Destructor, cleans up the memory.

Protected Member Functions

- void readDataFromMatlab (const mxArray *labelVec, const mxArray *instanceMat, parameters *param)

  Loads the data from Matlab.

Additional Inherited Members

5.2.1 Detailed Description

Class which manipulates sparse array of vectors (similarly to Matlab sparse matrix structure), with added functionality to load data directly from Matlab.

Class which manipulates sparse array of vectors (similarly to Matlab sparse matrix structure), with added functionality to load data directly from Matlab. Unlike budgetedData, where we load the data in smaller chunks, in this class we assume that the entire data can be loaded into memory, as it is already loaded in Matlab.

5.2.2 Constructor & Destructor Documentation

5.2.2.1 budgetedDataMatlab::budgetedDataMatlab (const mxArray * *labelVec*, const mxArray * *instanceMat*, parameters * *param*, bool *keepAssignments* = `false`, vector< int > * *yLabels* = `NULL`) `[inline]`

Constructor, invokes readDataFromMatlab that loads Matlab data.

Parameters

in	*labelVec*	Vector of labels.
in	*instanceMat*	Matrix of data points, each row is a single data point.
in	*param*	The parameters of the algorithm.
in	*keepAssignments*	True for AMM batch, otherwise false. Unlike in budgetedData case, no file is created to store the assignments as it is assumed that the memory to hold the assignments can be allocated in whole.
in	*yLabels*	Possible labels in the classification problem, for training data is NULL since inferred from data.

5.2.3 Member Function Documentation

5.2.3.1 bool budgetedDataMatlab::readChunk (unsigned int *size*, bool *assign* = `false`) `[inline]`, `[virtual]`

Overrides virtual function from budgetedData, simply returns false regardless of inputs as the data is fully loaded from Matlab.

Parameters

in	*size*	Size of the chunk to be loaded.
in	*assign*	True if assignment should be saved, false otherwise.

Returns

False regardless of inputs, since the data is fully loaded from Matlab.

Reimplemented from budgetedData.

5.2.3.2 void budgetedDataMatlab::readDataFromMatlab (const mxArray * *labelVec*, const mxArray * *instanceMat*, parameters * *param*) [protected]

Loads the data from Matlab.

Parameters

in	*labelVec*	Vector of labels.
in	*instanceMat*	Matrix of data points, each row is a single data point.
in	*param*	The parameters of the algorithm.

The documentation for this class was generated from the following files:

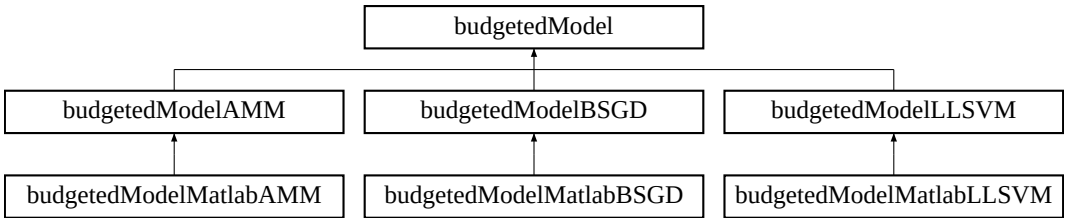
- budgetedSVM_matlab.h
- budgetedSVM_matlab.cpp

5.3 budgetedModel Class Reference

Interface which defines methods to load model from and save model to text file.

`#include <budgetedSVM.h>`

Inheritance diagram for budgetedModel:



Public Member Functions

- virtual ~budgetedModel (void)

  Destructor, cleans up the memory.
- virtual bool saveToTextFile (const char *filename, vector< int > *yLabels, parameters *param)=0

  Saves the trained model to .txt file.
- virtual bool loadFromTextFile (const char *filename, vector< int > *yLabels, parameters *param)=0

  Loads the trained model from .txt file.

Static Public Member Functions

- static int getAlgorithm (const char *filename)

  Get algorithm code from the trained model stored in .txt file, according to enumeration explained at the top of this page.

5.3.1 Detailed Description

Interface which defines methods to load model from and save model to text file.

In order to ensure that all algorithms have the same interface when it comes to storing/loading of the trained model, this interface is to be implemented by each separate algorithm model.

5.3.2 Member Function Documentation

5.3.2.1 static int budgetedModel::getAlgorithm (const char * *filename*) [static]

Get algorithm code from the trained model stored in .txt file, according to enumeration explained at the top of this page.

Parameters

in	*filename*	Filename of the .txt file where the model is saved.

Returns

-1 if error, otherwise returns algorithm code from the model file.

5.3.2.2 bool budgetedModel::loadFromTextFile (const char * *filename,* vector< int > * *yLabels,* parameters * *param*) [pure virtual]

Loads the trained model from .txt file.

Parameters

in	*filename*	Filename of the .txt file where the model is saved.
out	*yLabels*	Vector of possible labels.
out	*param*	The parameters of the algorithm.

Returns

False if error encountered, otherwise true.

The text file has the following rows: [*ALGORITHM*, *DIMENSION*, *NUMBER_OF_CLASSES*, *LABELS*, *NUMBER_OF_WEIGHTS*, *BIAS_TERM*, *KERNEL_WIDTH*, *MODEL*]. In order to compress memory and to use the memory efficiently, we coded the model in the following way:

- For AMM batch, AMM online, Pegasos: The model is stored so that each row of the text file corresponds to one weight. The first element of each weight is the class of the weight, followed by the degradation of the weight. The rest of the row corresponds to non-zero elements of the weight, given as *feature_index:feature_value*, in a standard LIBSVM format.

- For BSGD: The model is stored so that each row corresponds to one support vector (or weight). The first elements of each weight correspond to alpha parameters for each class, given in order specified by *LABELS* row. However, since alpha can be equal to 0, we use LIBSVM format to store alphas, as *-class_index:class-specific_alpha,* where we added '-' (minus sign) in front of the class index to differentiate between class indices and feature indices that follow. After the alphas, in the same row the elements of the weights (or support vectors) for each feature are given in LIBSVM format.

- For LLSVM: The model is stored so that each row corresponds to one landmark point. The first element of each row corresponds to element of linear SVM hyperplane for that particular landmark point. This is followed by features of the landmark point in the original feature space of the data set in LIBSVM format.

Implemented in budgetedModelAMM, budgetedModelBSGD, and budgetedModelLLSVM.

5.3.2.3 bool budgetedModel::saveToTextFile (const char * *filename,* vector< int > * *yLabels,* parameters * *param*) [pure virtual]

Saves the trained model to .txt file.

Parameters

in	*filename*	Filename of the .txt file where the model is saved.
in	*yLabels*	Vector of possible labels.
in	*param*	The parameters of the algorithm.

Returns

False if error encountered, otherwise true.

The text file has the following rows: [*ALGORITHM*, *DIMENSION*, *NUMBER_OF_CLASSES*, *LABELS*, *NUMBER_OF_WEIGHTS*, *BIAS_TERM*, *KERNEL_WIDTH*, *MODEL*]. In order to compress memory and to use the memory efficiently, we coded the model in the following way:

- For AMM batch, AMM online, Pegasos: The model is stored so that each row of the text file corresponds to one weight. The first element of each weight is the class of the weight, followed by the degradation of the weight. The rest of the row corresponds to non-zero elements of the weight, given as *feature_index:feature_value*, in a standard LIBSVM format.

- For BSGD: The model is stored so that each row corresponds to one support vector (or weight). The first elements of each weight correspond to alpha parameters for each class, given in order by *LABELS* row. However, since alpha can be equal to 0, we use LIBSVM format to store alphas, as *-class_index:class-specific_alpha,* where we added '-' (minus sign) in front of the class index to differentiate between class indices and feature indices that follow. After the alphas, in the same row the elements of the weights (or support vectors) for each feature are given in LIBSVM format.

- For LLSVM: The model is stored so that each row corresponds to one landmark point. The first element of each row corresponds to element of linear SVM hyperplane for that particular landmark point. This is followed by features of the landmark point in the original feature space of the data set in LIBSVM format.

Implemented in budgetedModelAMM, budgetedModelBSGD, and budgetedModelLLSVM.

The documentation for this class was generated from the following files:

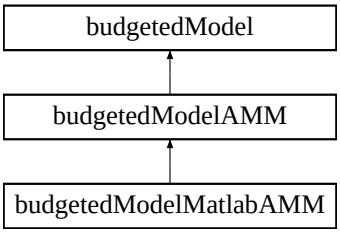
- budgetedSVM.h
- budgetedSVM.cpp

5.4 budgetedModelAMM Class Reference

Class which holds the AMM model, and implements methods to load AMM model from and save AMM model to text file.

`#include <mm_algs.h>`

Inheritance diagram for budgetedModelAMM:



Public Member Functions

- budgetedModelAMM (void)

  Constructor, initializes the MM model to zero weights.
- ~budgetedModelAMM (void)

  Destructor, cleans up memory taken by AMM.
- vector< vectorOfBudgetVectors > * getModel (void)

  Used to obtain a pointer to a current AMM model.
- bool saveToTextFile (const char *filename, vector< int > *yLabels, parameters *param)

  Saves the trained AMM model to .txt file.
- bool loadFromTextFile (const char *filename, vector< int > *yLabels, parameters *param)

  Loads the trained AMM model from .txt file.

Protected Attributes

- vector< vectorOfBudgetVectors > * modelMM

  Holds AMM batch, AMM online, or PEGASOS models.

Additional Inherited Members

5.4.1 Detailed Description

Class which holds the AMM model, and implements methods to load AMM model from and save AMM model to text file.

5.4.2 Member Function Documentation

5.4.2.1 vector< vectorOfBudgetVectors > * budgetedModelAMM::getModel (void) [inline]

Used to obtain a pointer to a current AMM model.

Returns

A pointer to a current AMM model.

5.4.2.2 bool budgetedModelAMM::loadFromTextFile (const char * *filename*, vector< int > * *yLabels*, parameters * *param*) [virtual]

Loads the trained AMM model from .txt file.

Parameters

in	*filename*	Filename of the .txt file where the model is saved.
out	*yLabels*	Vector of possible labels.
out	*param*	The parameters of the algorithm.

Returns

Returns false if error encountered, otherwise true.

The text file has the following rows: [*ALGORITHM*, *DIMENSION*, *NUMBER_OF_CLASSES*, *LABELS*, *NUMBER_OF_WEIGHTS*, *BIAS_TERM*, *KERNEL_WIDTH*, *MODEL*]. In order to compress memory and to use the memory efficiently, we coded the model in the following way:

Each row of the text file corresponds to one weight. The first element of each weight is the class of the weight, followed by the degradation of the weight. The rest of the row corresponds to non-zero elements of the weight, given as feature_index:feature_value, in a standard LIBSVM format.

Implements budgetedModel.

5.4.2.3 bool budgetedModelAMM::saveToTextFile (const char * *filename*, vector< int > * *yLabels*, parameters * *param*) [virtual]

Saves the trained AMM model to .txt file.

Parameters

in	*filename*	Filename of the .txt file where the model is saved.
in	*yLabels*	Vector of possible labels.
in	*param*	The parameters of the algorithm.

Returns

Returns false if error encountered, otherwise true.

The text file has the following rows: [*ALGORITHM*, *DIMENSION*, *NUMBER_OF_CLASSES*, *LABELS*, *NUMBER_OF_WEIGHTS*, *BIAS_TERM*, *KERNEL_WIDTH*, *MODEL*]. In order to compress memory and to use the memory efficiently, we coded the model in the following way:

Each row of the text file corresponds to one weight. The first element of each weight is the class of the weight, followed by the degradation of the weight. The rest of the row corresponds to non-zero elements of the weight, given as feature_index:feature_value, in a standard LIBSVM format.

Implements budgetedModel.

The documentation for this class was generated from the following files:

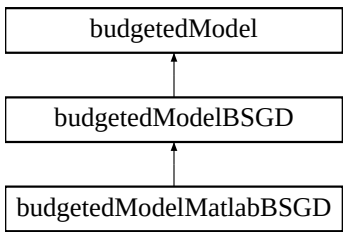
- mm_algs.h
- mm_algs.cpp

5.5 budgetedModelBSGD Class Reference

Class which holds the BSGD model (comprising the support vectors stored as budgetedVectorBSGD), and implements methods to load BSGD model from and save BSGD model to text file.

`#include <bsgd.h>`

Inheritance diagram for budgetedModelBSGD:



Public Member Functions

- budgetedModelBSGD (void)

  Constructor, initializes the BSGD model to zero-vectors.

- ~budgetedModelBSGD (void)

  Destructor, cleans up memory taken by BSGD.
- bool saveToTextFile (const char *filename, vector< int > *yLabels, parameters *param)

  Saves the trained BSGD model to .txt file.
- bool loadFromTextFile (const char *filename, vector< int > *yLabels, parameters *param)

  Loads the trained BSGD model from .txt file.

Public Attributes

- vector< budgetedVectorBSGD * > * modelBSGD

  Holds BSGD model.

Additional Inherited Members

5.5.1 Detailed Description

Class which holds the BSGD model (comprising the support vectors stored as budgetedVectorBSGD), and implements methods to load BSGD model from and save BSGD model to text file.

5.5.2 Member Function Documentation

5.5.2.1 bool budgetedModelBSGD::loadFromTextFile (const char * *filename*, vector< int > * *yLabels*, parameters * *param*) [virtual]

Loads the trained BSGD model from .txt file.

Parameters

in	*filename*	Filename of the .txt file where the model is saved.
out	*yLabels*	Vector of possible labels.
out	*param*	The parameters of the algorithm.

Returns

Returns false if error encountered, otherwise true.

The text file has the following rows: [*ALGORITHM*, *DIMENSION*, *NUMBER_OF_CLASSES*, *LABELS*, *NUMBER_OF_WEIGHTS*, *BIAS_TERM*, *KERNEL_WIDTH*, *MODEL*]. In order to compress memory and to use the memory efficiently, we coded the model in the following way:

Each row corresponds to one support vector (or weight). The first elements of each weight correspond to alpha parameters for each class, given in order of "labels" member of the Matlab structure. However, since alpha can be equal to 0, we use LIBSVM format to store alphas, as -class_index:class-specific_alpha, where we added '-' (minus sign) in front of the class index to differentiate between class indices and feature indices that follow. After the alphas, in the same row the elements of the weights (or support vectors) for each feature are given in LIBSVM format.

Implements budgetedModel.

5.5.2.2 bool budgetedModelBSGD::saveToTextFile (const char * *filename*, vector< int > * *yLabels*, parameters * *param*) [virtual]

Saves the trained BSGD model to .txt file.

Parameters

in	*filename*	Filename of the .txt file where the model is saved.
in	*yLabels*	Vector of possible labels.
in	*param*	The parameters of the algorithm.

Returns

Returns false if error encountered, otherwise true.

The text file has the following rows: [*ALGORITHM*, *DIMENSION*, *NUMBER_OF_CLASSES*, *LABELS*, *NUMBER_OF_WEIGHTS*, *BIAS_TERM*, *KERNEL_WIDTH*, *MODEL*]. In order to compress memory and to use the memory efficiently, we coded the model in the following way:

Each row corresponds to one support vector (or weight). The first elements of each weight correspond to alpha parameters for each class, given in order of "labels" member of the Matlab structure. However, since alpha can be equal to 0, we use LIBSVM format to store alphas, as -class_index:class-specific_alpha, where we added '-' (minus sign) in front of the class index to differentiate between class indices and feature indices that follow. After the alphas, in the same row the elements of the weights (or support vectors) for each feature are given in LIBSVM format.

Implements budgetedModel.

The documentation for this class was generated from the following files:

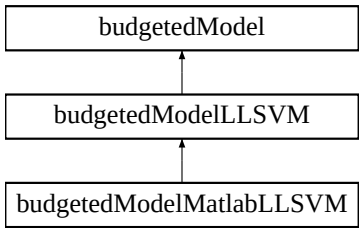
- bsgd.h
- bsgd.cpp

5.6 budgetedModelLLSVM Class Reference

Class which holds the LLSVM model, and implements methods to load LLSVM model from and save LLSVM model to text file.

`#include <llsvm.h>`

Inheritance diagram for budgetedModelLLSVM:



Public Member Functions

- budgetedModelLLSVM (void)

  Constructor, initializes the LLSVM model. Simply allocates memory for a vector of landmark points, where each is stored in budgetedVectorLLSVM.
- ~budgetedModelLLSVM (void)

  Destructor, cleans up memory taken by LLSVM.
- bool saveToTextFile (const char *filename, vector< int > *yLabels, parameters *param)

  Saves the trained LLSVM model to .txt file.
- bool loadFromTextFile (const char *filename, vector< int > *yLabels, parameters *param)

  Loads the trained LLSVM model from .txt file.

Public Attributes

- vector< budgetedVectorLLSVM * > * modelLLSVMlandmarks

  Holds landmark points, used to compute the transformation matrix modelLLSVMmatrixW.
- VectorXd modelLLSVMweightVector

  Holds weight vector, the solution of linear SVM on transformed points.
- MatrixXd modelLLSVMmatrixW

  Holds transformation matrix, used to compute the mapping from original feature space into low-D space.

Additional Inherited Members

5.6.1 Detailed Description

Class which holds the LLSVM model, and implements methods to load LLSVM model from and save LLSVM model to text file.

5.6.2 Member Function Documentation

5.6.2.1 bool budgetedModelLLSVM::loadFromTextFile (const char * *filename*, vector< int > * *yLabels*, parameters * *param*) [virtual]

Loads the trained LLSVM model from .txt file.

Parameters

in	*filename*	Filename of the .txt file where the model is saved.
out	*yLabels*	Vector of possible labels.
out	*param*	The parameters of the algorithm.

Returns

Returns false if error encountered, otherwise true.

The text file has the following rows: [*ALGORITHM*, *DIMENSION*, *NUMBER_OF_CLASSES*, *LABELS*, *NUMBER_OF_WEIGHTS*, *BIAS_TERM*, *KERNEL_WIDTH*, *MODEL*]. In order to compress memory and to use the memory efficiently, we coded the model in the following way:

Each row corresponds to one landmark point. The first element of each row corresponds to element of linear SVM hyperplane for that particular landmark point. This is followed by features of the landmark point in the original feature space of the data set, stored in LIBSVM format.

Implements budgetedModel.

5.6.2.2 bool budgetedModelLLSVM::saveToTextFile (const char * *filename*, vector< int > * *yLabels*, parameters * *param*) [virtual]

Saves the trained LLSVM model to .txt file.

Parameters

in	*filename*	Filename of the .txt file where the model is saved.
in	*yLabels*	Vector of possible labels.
in	*param*	The parameters of the algorithm.

Returns

Returns false if error encountered, otherwise true.

The text file has the following rows: [*ALGORITHM*, *DIMENSION*, *NUMBER_OF_CLASSES*, *LABELS*, *NUMBER_OF_WEIGHTS*, *BIAS_TERM*, *KERNEL_WIDTH*, *MODEL*]. In order to compress memory and to use the memory efficiently, we coded the model in the following way:

Each row corresponds to one landmark point. The first element of each row corresponds to element of linear SVM hyperplane for that particular landmark point. This is followed by features of the landmark point in the original feature space of the data set, stored in LIBSVM format.

Implements budgetedModel.

The documentation for this class was generated from the following files:

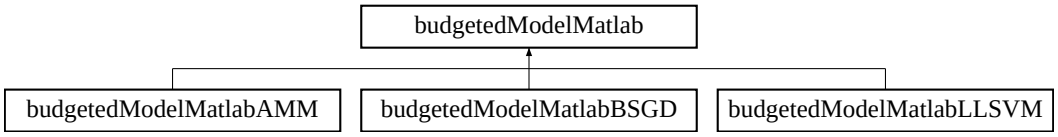
- llsvm.h
- llsvm.cpp

5.7 budgetedModelMatlab Class Reference

Interface which defines methods to load model from and save model to Matlab environment.

`#include <budgetedSVM_matlab.h>`

Inheritance diagram for budgetedModelMatlab:



Public Member Functions

- virtual void saveToMatlabStruct (mxArray ∗plhs[], vector< int > ∗yLabels, parameters ∗param)=0

  Save the trained model to Matlab, by creating Matlab structure.
- virtual bool loadFromMatlabStruct (const mxArray ∗matlabStruct, vector< int > ∗yLabels, parameters ∗param, const char ∗∗msg)=0

  Loads the trained model from Matlab structure.

Static Public Member Functions

- static int getAlgorithm (const mxArray ∗matlabStruct)

  Get algorithm from the trained model stored in Matlab structure.

5.7.1 Detailed Description

Interface which defines methods to load model from and save model to Matlab environment.

5.7.2 Member Function Documentation

5.7.2.1 static int budgetedModelMatlab::getAlgorithm (const mxArray ∗ *matlabStruct*) [static]

Get algorithm from the trained model stored in Matlab structure.

Parameters

in	*matlabStruct*	Pointer to Matlab structure.

Returns

-1 if error, otherwise returns algorithm code from the model file.

5.7.2.2 bool budgetedModelMatlab::loadFromMatlabStruct (const mxArray * *matlabStruct*, vector< int > * *yLabels*, parameters * *param*, const char ** *msg*) [pure virtual]

Loads the trained model from Matlab structure.

Parameters

in	*matlabStruct*	Pointer to Matlab structure.
in	*yLabels*	Vector of possible labels.
in	*param*	The parameters of the algorithm.
out	*msg*	Error message, if error encountered.

Returns

False if error encountered, otherwise true.

The Matlab structure is organized as [*algorithm*, *dimension*, *numClasses*, *labels*, *numWeights*, *paramBias*, *kernelWidth*, *model*]. In order to compress memory and to use the memory efficiently, we coded the model in the following way:

- AMM online, AMM batch, and Pegasos: The model is stored as ((*dimension* + 1) x *numWeights*) matrix. The first element of each weight is the degradation of the weight, followed by values of the weight for each feature of the data set. If bias term is non-zero, then the final element of each weight corresponds to bias term, and the matrix is of size ((*dimension* + 2) x *numWeights*). By looking at *labels* and *numWeights* members of Matlab structure we can find out which weights belong to which class. For example, first numWeights[0] weights belong to labels[0] class, next numWeights[1] weights belong to labels[1] class, and so on.

- BSGD: The model is stored as ((*numClasses* + *dimension*) x *numWeights*) matrix. The first *numClasses* elements of each weight correspond to alpha parameters for each class, given in order of "labels" member of the Matlab structure. This is followed by elements of the weights (or support vectors) for each feature of the data set.

- LLSVM: The model is stored as ((1 + *dimension*) x *numWeights*) matrix. Each row corresponds to one landmark point. The first element of each row corresponds to element of linear SVM hyperplane for that particular landmark point. This is followed by features of the landmark point in the original feature space.

Implemented in budgetedModelMatlabLLSVM, budgetedModelMatlabBSGD, and budgetedModelMatlabAMM.

5.7.2.3 void budgetedModelMatlab::saveToMatlabStruct (mxArray * *plhs[]*, vector< int > * *yLabels*, parameters * *param*) [pure virtual]

Save the trained model to Matlab, by creating Matlab structure.

Parameters

out	*plhs*	Pointer to Matlab output.
in	*yLabels*	Vector of possible labels.
in	*param*	The parameters of the algorithm.

The Matlab structure is organized as [*algorithm*, *dimension*, *numClasses*, *labels*, *numWeights*, *paramBias*, *kernelWidth*, *model*]. In order to compress memory and to use the memory efficiently, we coded the model in the following way:

- AMM online, AMM batch, and Pegasos: The model is stored as ((*dimension* + 1) x *numWeights*) matrix. The first element of each weight is the degradation of the weight, followed by values of the weight for each feature of the data set. If bias term is non-zero, then the final element of each weight corresponds to bias term, and the matrix is of size ((*dimension* + 2) x *numWeights*). By looking at *labels* and *numWeights* members of Matlab structure we can find out which weights belong to which class. For example, first numWeights[0] weights belong to labels[0] class, next numWeights[1] weights belong to labels[1] class, and so on.

- BSGD: The model is stored as ((*numClasses* + *dimension*) x *numWeights*) matrix. The first *numClasses* elements of each weight correspond to alpha parameters for each class, given in order of *labels* member of the Matlab structure. This is followed by elements of the weights (or support vectors) for each feature of the data set.

- LLSVM: The model is stored as ((1 + *dimension*) x *numWeights*) matrix. Each row corresponds to one landmark point. The first element of each row corresponds to element of linear SVM hyperplane for that particular landmark point. This is followed by features of the landmark point in the original feature space.

Implemented in budgetedModelMatlabLLSVM, budgetedModelMatlabBSGD, and budgetedModelMatlabAMM.

The documentation for this class was generated from the following files:

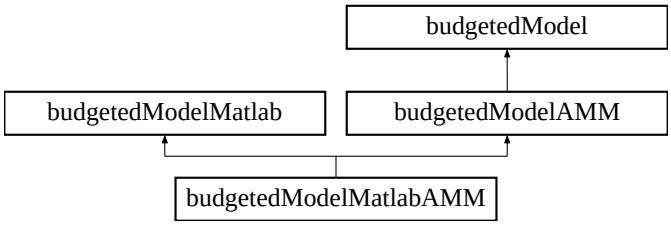
- budgetedSVM_matlab.h
- budgetedSVM_matlab.cpp

5.8 budgetedModelMatlabAMM Class Reference

Class which holds the AMM model, and implements methods to load AMM model from and save AMM model to Matlab environment.

`#include <budgetedSVM_matlab.h>`

Inheritance diagram for budgetedModelMatlabAMM:



Public Member Functions

- void saveToMatlabStruct (mxArray ∗plhs[], vector< int > ∗yLabels, parameters ∗param)

  Save the trained model to Matlab, by creating Matlab structure.

- bool loadFromMatlabStruct (const mxArray ∗matlabStruct, vector< int > ∗yLabels, parameters ∗param, const char ∗∗msg)

  Loads the trained model from Matlab structure.

Additional Inherited Members

5.8.1 Detailed Description

Class which holds the AMM model, and implements methods to load AMM model from and save AMM model to Matlab environment.

5.8.2 Member Function Documentation

5.8.2.1 bool budgetedModelMatlabAMM::loadFromMatlabStruct (const mxArray * *matlabStruct,* vector< int > * *yLabels,* parameters * *param,* const char ** *msg*) [virtual]

Loads the trained model from Matlab structure.

Parameters

in	*matlabStruct*	Pointer to Matlab structure.
in	*yLabels*	Vector of possible labels.
in	*param*	The parameters of the algorithm.
out	*msg*	Error message, if error encountered.

Returns

False if error encountered, otherwise true.

The Matlab structure is organized as ["algorithm", "dimension", "numClasses", "labels", "numWeights", "paramBias", "kernelWidth", "model"]. In order to compress memory and to use the memory efficiently, we coded the model in the following way:

The model is stored as (("dimension" + 1) by "numWeights") matrix. The first element of each weight is the degradation of the weight, followed by values of the weight for each feature of the data set. If bias term is non-zero, then the final element of each weight corresponds to bias term, and the matrix is of size (("dimension" + 2) by "numWeights"). By looking at "labels" and "numWeights" members of Matlab structure we can find out which weights belong to which class. For example, first numWeights[0] weights belong to labels[0] class, next numWeights[1] weights belong to labels[1] class, and so on.

Implements budgetedModelMatlab.

5.8.2.2 void budgetedModelMatlabAMM::saveToMatlabStruct (mxArray * *plhs[],* vector< int > * *yLabels,* parameters * *param*) [virtual]

Save the trained model to Matlab, by creating Matlab structure.

Parameters

out	*plhs*	Pointer to Matlab output.
in	*yLabels*	Vector of possible labels.
in	*param*	The parameters of the algorithm.

The Matlab structure is organized as ["algorithm", "dimension", "numClasses", "labels", "numWeights", "paramBias", "kernelWidth", "model"]. In order to compress memory and to use the memory efficiently, we coded the model in the following way:

The model is stored as (("dimension" + 1) by "numWeights") matrix. The first element of each weight is the degradation of the weight, followed by values of the weight for each feature of the data set. If bias term is non-zero, then the final element of each weight corresponds to bias term, and the matrix is of size (("dimension" + 2) by "numWeights"). By looking at "labels" and "numWeights" members of Matlab structure we can find out which weights belong to which class. For example, first numWeights[0] weights belong to labels[0] class, next numWeights[1] weights belong to

labels[1] class, and so on.

Implements budgetedModelMatlab.

The documentation for this class was generated from the following files:

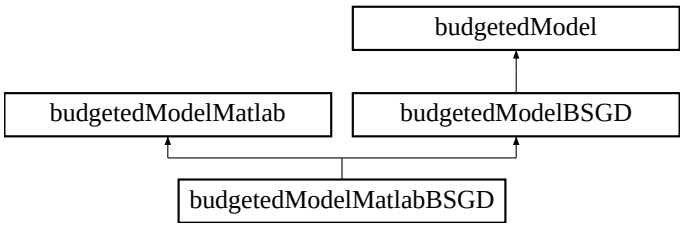
- budgetedSVM_matlab.h
- budgetedSVM_matlab.cpp

5.9 budgetedModelMatlabBSGD Class Reference

Class which holds the BSGD model, and implements methods to load BSGD model from and save BSGD model to Matlab environment.

`#include <budgetedSVM_matlab.h>`

Inheritance diagram for budgetedModelMatlabBSGD:



Public Member Functions

- void saveToMatlabStruct (mxArray ∗plhs[], vector< int > ∗yLabels, parameters ∗param)

  Save the trained model to Matlab, by creating Matlab structure.
- bool loadFromMatlabStruct (const mxArray ∗matlabStruct, vector< int > ∗yLabels, parameters ∗param, const char ∗∗msg)

  Loads the trained model from Matlab structure.

Additional Inherited Members

5.9.1 Detailed Description

Class which holds the BSGD model, and implements methods to load BSGD model from and save BSGD model to Matlab environment.

5.9.2 Member Function Documentation

5.9.2.1 bool budgetedModelMatlabBSGD::loadFromMatlabStruct (const mxArray ∗ *matlabStruct,* vector< int > ∗ *yLabels,* parameters ∗ *param,* const char ∗∗ *msg*) [virtual]

Loads the trained model from Matlab structure.

Parameters

in	*matlabStruct*	Pointer to Matlab structure.
in	*yLabels*	Vector of possible labels.
in	*param*	The parameters of the algorithm.
out	*msg*	Error message, if error encountered.

Returns

False if error encountered, otherwise true.

The Matlab structure is organized as ["algorithm", "dimension", "numClasses", "labels", "numWeights", "paramBias", "kernelWidth", "model"]. In order to compress memory and to use the memory efficiently, we coded the model in the following way:

The model is stored as (("numClasses" + "dimension") by "numWeights") matrix. The first "numClasses" elements of each weight correspond to alpha parameters for each class, given in order of "labels" member of the Matlab structure. This is followed by elements of the weights (or support vectors) for each feature of the data set.

Implements budgetedModelMatlab.

5.9.2.2 void budgetedModelMatlabBSGD::saveToMatlabStruct (mxArray * *plhs[]*, vector< int > * *yLabels*, parameters * *param*) [virtual]

Save the trained model to Matlab, by creating Matlab structure.

Parameters

out	*plhs*	Pointer to Matlab output.
in	*yLabels*	Vector of possible labels.
in	*param*	The parameters of the algorithm.

The Matlab structure is organized as ["algorithm", "dimension", "numClasses", "labels", "numWeights", "paramBias", "kernelWidth", "model"]. In order to compress memory and to use the memory efficiently, we coded the model in the following way:

The model is stored as (("numClasses" + "dimension") by "numWeights") matrix. The first "numClasses" elements of each weight correspond to alpha parameters for each class, given in order of "labels" member of the Matlab structure. This is followed by elements of the weights (or support vectors) for each feature of the data set.

Implements budgetedModelMatlab.

The documentation for this class was generated from the following files:

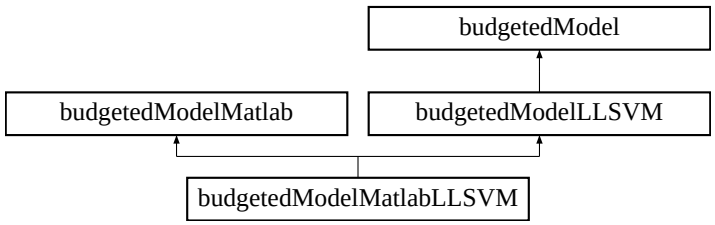
- budgetedSVM_matlab.h
- budgetedSVM_matlab.cpp

5.10 budgetedModelMatlabLLSVM Class Reference

Class which holds the LLSVM model, and implements methods to load LLSVM model from and save LLSVM model to Matlab environment.

`#include <budgetedSVM_matlab.h>`

Inheritance diagram for budgetedModelMatlabLLSVM:



Public Member Functions

- void saveToMatlabStruct (mxArray *plhs[], vector< int > *yLabels, parameters *param)

  Save the trained model to Matlab, by creating Matlab structure.
- bool loadFromMatlabStruct (const mxArray *matlabStruct, vector< int > *yLabels, parameters *param, const char **msg)

  Loads the trained model from Matlab structure.

Additional Inherited Members

5.10.1 Detailed Description

Class which holds the LLSVM model, and implements methods to load LLSVM model from and save LLSVM model to Matlab environment.

5.10.2 Member Function Documentation

5.10.2.1 bool budgetedModelMatlabLLSVM::loadFromMatlabStruct (const mxArray * *matlabStruct*, vector< int > * *yLabels*, parameters * *param*, const char ** *msg*) [virtual]

Loads the trained model from Matlab structure.

Parameters

in	*matlabStruct*	Pointer to Matlab structure.
in	*yLabels*	Vector of possible labels.
in	*param*	The parameters of the algorithm.
out	*msg*	Error message, if error encountered.

Returns

False if error encountered, otherwise true.

The Matlab structure is organized as ["algorithm", "dimension", "numClasses", "labels", "numWeights", "paramBias", "kernelWidth", "model"]. In order to compress memory and to use the memory efficiently, we coded the model in the following way:

The model is stored as ((1 + "dimension") by "numWeights") matrix. Each row corresponds to one landmark point. The first element of each row corresponds to element of linear SVM hyperplane for that particular landmark point. This is followed by features of the landmark point in the original feature space.

Implements budgetedModelMatlab.

5.10.2.2 void budgetedModelMatlabLLSVM::saveToMatlabStruct (mxArray * *plhs[]*, vector< int > * *yLabels*, parameters * *param*) [virtual]

Save the trained model to Matlab, by creating Matlab structure.

Parameters

out	*plhs*	Pointer to Matlab output.
in	*yLabels*	Vector of possible labels.
in	*param*	The parameters of the algorithm.

The Matlab structure is organized as ["algorithm", "dimension", "numClasses", "labels", "numWeights", "paramBias", "kernelWidth", "model"]. In order to compress memory and to use the memory efficiently, we coded the model in

the following way:

The model is stored as ((1 + "dimension") by "numWeights") matrix. Each row corresponds to one landmark point. The first element of each row corresponds to element of linear SVM hyperplane for that particular landmark point. This is followed by features of the landmark point in the original feature space.

Implements budgetedModelMatlab.

The documentation for this class was generated from the following files:

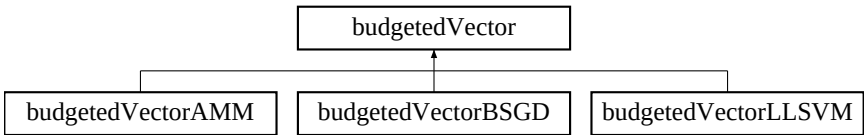
- budgetedSVM_matlab.h
- budgetedSVM_matlab.cpp

5.11 budgetedVector Class Reference

Class which handles high-dimensional vectors.

`#include <budgetedSVM.h>`

Inheritance diagram for budgetedVector:



Public Member Functions

- virtual long double getSqrL2norm (void)

  Returns sqrL2norm, a squared L2-norm of the vector.
- unsigned int getID (void)

  Returns weightID, a unique ID of a vector.
- const float operator[] (int idx) const

  Overloaded [] operator that returns a vector element stored in array.
- float & operator[] (int idx)

  Overloaded [] operator that assigns a value to vector element stored in array.
- budgetedVector (unsigned int dim=0, unsigned int chnkWght=0)

  Constructor, initializes the vector to all zeros.
- virtual ~budgetedVector ()

  Destructor, cleans up the memory.
- virtual void clear (void)

  Clears the vector of all non-zero elements, resulting in a zero-vector.
- virtual void createVectorUsingDataPoint (budgetedData *inputData, unsigned int t, parameters *param)

  Create new vector from training data point.
- virtual long double sqrNorm (void)

  Calculates a squared L2-norm of the vector.
- virtual long double gaussianKernel (budgetedVector *otherVector, parameters *param)

  Computes Gaussian kernel kernel between this budgetedVector vector and another vector stored in budgetedVector.
- virtual long double gaussianKernel (unsigned int t, budgetedData *inputData, parameters *param, long double inputVectorSqrNorm=0.0)

  Computes Gaussian kernel kernel between this budgetedVector vector and another vector from input data stored in budgetedData.
- virtual long double linearKernel (unsigned int t, budgetedData *inputData, parameters *param)

Computes linear kernel between this budgetedVector vector and another vector stored in budgetedData.

- virtual long double linearKernel (budgetedVector ∗otherVector)

  Computes linear kernel between this budgetedVector vector and another vector stored in budgetedVector.

Protected Member Functions

- virtual void setSqrL2norm (long double newSqrNorm)

  Returns sqrL2norm, a squared L2-norm of the vector.

Protected Attributes

- unsigned int weightID

  Unique ID of the vector, used in AMM batch to uniquely identify which vector is assigned to which data points. Assigned when the vector is created.
- vector< float ∗ > array

  Array of vector chunks, element of the array is NULL if all features within a chunk represented by the element are equal to 0.
- long double sqrL2norm

  Squared L2-norm of the vector.

Static Protected Attributes

- static unsigned int dimension = 0

  Dimensionality of the vector.
- static unsigned int id = 0

  ID of the vector.
- static unsigned int arrayLength = 0

  Number of vector chunks.
- static unsigned int chunkWeight = 0

  Length of the vector chunk (implemented as an array).

5.11.1 Detailed Description

Class which handles high-dimensional vectors.

In order to handle high-dimensional vectors (i.e., data points), we split the data vector into an array of smaller vectors (or chunks; implemented as a vector of arrays), and allocate memory for each chunk only if it contains at least one element that is non-zero. This is especially beneficial for very sparse data sets, where we can have considerable memory gains. Each chunk has a pointer to it stored in array, and a pointer is NULL if the chunk has all zero elements; non-NULL pointer points to a chunk that has allocated memory and which stores elements of the vector.

5.11.2 Constructor & Destructor Documentation

5.11.2.1 budgetedVector::budgetedVector (unsigned int *dim* = 0, unsigned int *chnkWght* = 0) [inline]

Constructor, initializes the vector to all zeros.

Parameters

in	*dim*	Dimensionality of the vector.
in	*chnkWght*	Size of each vector chunk.

5.11.3 Member Function Documentation

5.11.3.1 void budgetedVector::createVectorUsingDataPoint (budgetedData * *inputData*, unsigned int *t*, parameters * *param*) [inline],[virtual]

Create new vector from training data point.

Parameters

in	*inputData*	Input data from which t-th vector is considered.
in	*t*	Index of the input vector in the input data.
in	*param*	The parameters of the algorithm.

Initializes elements of a vector using a data point. Simply copies non-zero elements of the data point stored in budgetedData to the vector. If the vector already had non-zero elements, it is first cleared to become a zero-vector before copying the elements of a data point.

5.11.3.2 long double budgetedVector::gaussianKernel (budgetedVector * *otherVector*, parameters * *param*) [virtual]

Computes Gaussian kernel kernel between this budgetedVector vector and another vector stored in budgetedVector.

Parameters

in	*otherVector*	The second input vector to RBF kernel.
in	*param*	The parameters of the algorithm.

Returns

Value of RBF kernel between two vectors.

Function computes the value of Gaussian kernel between two vectors. The computation is very fast for sparse data, being only linear in a number of non-zero features. We use the fact that ||x - y||^2 = ||x||^2 - 2 * x^T * y + ||y||^2, where all right-hand side elements can be computed efficiently.

5.11.3.3 long double budgetedVector::gaussianKernel (unsigned int *t*, budgetedData * *inputData*, parameters * *param*, long double *inputVectorSqrNorm* = 0.0) [virtual]

Computes Gaussian kernel kernel between this budgetedVector vector and another vector from input data stored in budgetedData.

Parameters

in	*t*	Index of the input vector in the input data.
in	*inputData*	Input data from which t-th vector is considered.
in	*inputVectorSqrNorm*	If zero or not provided, the norm of t-th vector from inputData is computed on-the-fly.
in	*param*	The parameters of the algorithm.

Returns

Value of RBF kernel between two vectors.

Function computes the value of Gaussian kernel between two vectors. The computation is very fast for sparse data, being only linear in a number of non-zero features. We use the fact that ||x - y||^2 = ||x||^2 - 2 * x^T * y + ||y||^2, where all right-hand side elements can be computed efficiently.

5.11.3.4 unsigned int budgetedVector::getID (void) [inline]

Returns weightID, a unique ID of a vector.

Returns

Unique ID of a vector.

5.11.3.5 long double budgetedVector::getSqrL2norm (void) [inline], [virtual]

Returns sqrL2norm, a squared L2-norm of the vector.

Returns

Squared L2-norm of the vector.

Reimplemented in budgetedVectorAMM.

5.11.3.6 long double budgetedVector::linearKernel (unsigned int *t*, budgetedData * *inputData*, parameters * *param*) [virtual]

Computes linear kernel between this budgetedVector vector and another vector stored in budgetedData.

Parameters

in	*t*	Index of the input vector in the input data.
in	*inputData*	Input data from which t-th vector is considered.
in	*param*	The parameters of the algorithm.

Returns

Value of linear kernel between two input vectors.

Function computes the dot product of budgetedVector vector, and the input data point stored in budgetedData.

Reimplemented in budgetedVectorAMM.

5.11.3.7 long double budgetedVector::linearKernel (budgetedVector * *otherVector*) [virtual]

Computes linear kernel between this budgetedVector vector and another vector stored in budgetedVector.

Parameters

in	*otherVector*	The second input vector to linear kernel.

Returns

Value of linear kernel between two input vectors.

Function computes the value of linear kernel between two vectors.

5.11.3.8 const float budgetedVector::operator[] (int *idx*) const

Overloaded [] operator that returns a vector element stored in array.

Parameters

in	*idx*	Index of vector element that is retrieved.

Returns

Value of the element of the vector.

5.11.3.9 float & budgetedVector::operator[] (int *idx*)

Overloaded [] operator that assigns a value to vector element stored in array.

Parameters

in	*idx*	Index of vector element that is modified.

Returns

Value of the modified element of the vector.

5.11.3.10 void budgetedVector::setSqrL2norm (long double *newSqrNorm*) [inline], [protected], [virtual]

Returns sqrL2norm, a squared L2-norm of the vector.

Returns

Squared L2-norm of the vector.

5.11.3.11 long double budgetedVector::sqrNorm (void) [virtual]

Calculates a squared L2-norm of the vector.

Returns

Squared L2-norm of the vector.

Reimplemented in budgetedVectorAMM.

5.11.4 Member Data Documentation

5.11.4.1 vector< float * > budgetedVector::array [protected]

Array of vector chunks, element of the array is NULL if all features within a chunk represented by the element are equal to 0.

When the data is sparse, then we do not have to explicitly store every feature as most of them are equal to 0. One option is simply to follow LIBSVM format, and store in two linked lists feature index and the corresponing feature value. However, we found that updating this data structure can become prohibitively slow, as for high-dimensional data the weights can become much less sparse than the original data due to the weight update process, and the insertion of new elements into vector and vector traversal becomes very slow. We address this by storing a vector into structure that is a vector of dynamic arrays, where original, large vector is split into parts (or chunks), and each part is stored in an array within the vector structure. If all elements of the large vector within a chunk are zero, we do not allocate memory for that chunk and array element for this chunk will be NULL. In our experience, this significantly improves the training and testing time on very high-dimensional sparse data, such as on URL data set with more than 3.2 million features and only 0.004% non-zero values. If parameters::CHUNK_WEIGHT is set to 1, we obtain the LIBSVM-type representation where each chunk stores only one feature.

See Also

parameters::CHUNK_WEIGHT

5.11.4.2 static unsigned int budgetedVector::arrayLength = 0 [static], [protected]

Number of vector chunks.

In order to deal with high-dimensional data, each vector is split into several chunks, and the memory for the chunk is not allocated if all elements of a vector are equal to 0. The static variable chunkWeight specifies how many of these chunks are used to represent each vector.

See Also

parameters::CHUNK_WEIGHT

5.11.4.3 static unsigned int budgetedVector::chunkWeight = 0 [static], [protected]

Length of the vector chunk (implemented as an array).

See Also

parameters::CHUNK_WEIGHT

5.11.4.4 static unsigned int budgetedVector::id = 0 [static], [protected]

ID of the vector.

Each vector is uniquely identifiable using its ID. This is used in AMM batch algorithm, where weights and data points are matched, and we need to know which weight (represented as budgetedVector), is assigned to which data point during stochastic gradient descent training.

5.11.4.5 long double budgetedVector::sqrL2norm [protected]

Squared L2-norm of the vector.

After every modification to a budgetedVector object (e.g., due to an update in Stochastic Gradient Descent (SGD) learning step of AMM or BSGD algorithms), this property is updated to reflect the current squared norm of the vector. This is done to speed up computations of kernel functions, as Gaussian kernel used in BSGD and LLSVM is computed much faster when we know squared norms of two vectors that are inputs to a kernel function. Also, in AMM it is used in pruning phase to find the weights that need to be deleted, as we will prune only weights that have small L2-norm.

5.11.4.6 unsigned int budgetedVector::weightID [protected]

Unique ID of the vector, used in AMM batch to uniquely identify which vector is assigned to which data points. Assigned when the vector is created.

See Also

id

The documentation for this class was generated from the following files:

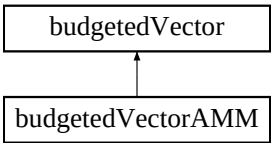
- budgetedSVM.h
- budgetedSVM.cpp

5.12 budgetedVectorAMM Class Reference

Class which holds sparse vector, which is split into a number of arrays to trade-off between speed of access and memory usage of sparse data, with added methods for AMM algorithms.

`#include <mm_algs.h>`

Inheritance diagram for budgetedVectorAMM:



Public Member Functions

- budgetedVectorAMM (unsigned int dim=0, unsigned int chnkWght=0)

  Constructor, initializes the vector to all zeros, and also initializes degradation parameter.
- long double getSqrL2norm (void)

  Returns sqrL2norm, a squared L2-norm of the vector, which accounts for the vector degradation.
- void downgrade (long oto)

  Downgrade the existing weight-vector.
- long double sqrNorm (void)

  Calculates a squared norm of the vector, but takes into consideration current degradation of a vector.
- long double getDegradation (void)

  Returns degradation of a vector.
- void setDegradation (long double deg)

  Sets degradation of a vector.
- void updateDegradation (unsigned int iteration, parameters *param)

  Computes degradation of a vector.
- void updateUsingDataPoint (budgetedData *inputData, unsigned int oto, unsigned int t, int sign, parameters *param)

  Updates a weight-vector when misclassification happens.
- void updateUsingVector (budgetedVectorAMM *otherVector, unsigned int oto, int sign, parameters *param)

  Updates a weight-vector when misclassification happens.
- void createVectorUsingDataPoint (budgetedData *inputData, unsigned int oto, unsigned int t, parameters *param)

  Create new weight from one of the zero-weights.
- long double linearKernel (unsigned int t, budgetedData *inputData, parameters *param)

  Computes linear kernel between vector and given input data point, but also accounts for degradation.
- long double linearKernel (budgetedVectorAMM *otherVector)

  Computes linear kernel between this budgetedVectorAMM vector and another vector stored in budgetedVectorAMM, but also accounts for degradation.

Protected Attributes

- long double degradation

  Degradation of the vector.

Friends

- class **budgetedModelAMM**
- class **budgetedModelMatlabAMM**

Additional Inherited Members

5.12.1 Detailed Description

Class which holds sparse vector, which is split into a number of arrays to trade-off between speed of access and memory usage of sparse data, with added methods for AMM algorithms.

5.12.2 Constructor & Destructor Documentation

5.12.2.1 budgetedVectorAMM::budgetedVectorAMM (unsigned int *dim* = 0, unsigned int *chnkWght* = 0) [inline]

Constructor, initializes the vector to all zeros, and also initializes degradation parameter.

Parameters

in	*dim*	Dimensionality of the vector.
in	*chnkWght*	Size of each vector chunk.

5.12.3 Member Function Documentation

5.12.3.1 void budgetedVectorAMM::createVectorUsingDataPoint (budgetedData * *inputData*, unsigned int *oto*, unsigned int *t*, parameters * *param*) [inline]

Create new weight from one of the zero-weights.

Parameters

in	*inputData*	Input data from which t-th vector is considered.
in	*oto*	Total number of iterations so far.
in	*t*	Index of the input vector in the input data.
in	*param*	The parameters of the algorithm.

The function simply copies the t-th data point in the input data to the vector vij, while also updating the degradation variable.

5.12.3.2 void budgetedVectorAMM::downgrade (long *oto*) [inline]

Downgrade the existing weight-vector.

Parameters

in	*oto*	Total number of AMM training iterations so far.

Using this function, each training iteration all non-zero weights are pushed closer to 0, to ensure the convergence of the algorithm to the optimal solution.

5.12.3.3 long double budgetedVectorAMM::getDegradation (void) [inline]

Returns degradation of a vector.

Returns

Degradation of a vector.

5.12.3.4 double budgetedVectorAMM::getSqrL2norm (void) [inline], [virtual]

Returns sqrL2norm, a squared L2-norm of the vector, which accounts for the vector degradation.

Returns

Squared L2-norm of the vector.

Reimplemented from budgetedVector.

5.12.3.5 long double budgetedVectorAMM::linearKernel (unsigned int *t*, **budgetedData** * *inputData*, **parameters** * *param*) [inline], [virtual]

Computes linear kernel between vector and given input data point, but also accounts for degradation.

Parameters

in	*t*	Index of the input vector in the input data.
in	*inputData*	Input data from which t-th vector is considered.
in	*param*	The parameters of the algorithm.

Returns

Value of linear kernel between two input vectors.

Function computes the dot product (i.e., linear kernel) between budgetedVector vector and the input data point from budgetedData.

Reimplemented from budgetedVector.

5.12.3.6 long double budgetedVectorAMM::linearKernel (**budgetedVectorAMM** * *otherVector*) [inline]

Computes linear kernel between this budgetedVectorAMM vector and another vector stored in budgetedVectorAMM, but also accounts for degradation.

Parameters

in	*otherVector*	The second input vector to linear kernel.

Returns

Value of linear kernel between two input vectors.

Function computes the dot product (or linear kernel) between two vectors.

5.12.3.7 long double budgetedVectorAMM::sqrNorm (void) [inline], [virtual]

Calculates a squared norm of the vector, but takes into consideration current degradation of a vector.

Returns

Squared norm of the vector.

Reimplemented from budgetedVector.

5.12.3.8 void budgetedVectorAMM::updateDegradation (unsigned int *iteration,* parameters * *param*) [inline]

Computes degradation of a vector.

Parameters

in	*iteration*	Training iteration at which the degradation is set, used to compute the degradation value.
in	*param*	The parameters of the algorithm.

5.12.3.9 void budgetedVectorAMM::updateUsingDataPoint (budgetedData * *inputData,* unsigned int *oto,* unsigned int *t,* int *sign,* parameters * *param*)

Updates a weight-vector when misclassification happens.

Parameters

in	*inputData*	Input data from which t-th vector is considered.
in	*oto*	Total number of iterations so far.
in	*t*	Index of the input vector in the input data.
in	*sign*	+1 if the input vector is of the true class, -1 otherwise, specifies how the weights will be updated.
in	*param*	The parameters of the algorithm.

When we misclassify a data point during training, this function is used to update the existing weight-vector. It brings the true-class weight closer to the misclassified data point, and to push the winning other-class weight away from the misclassified point according to AMM weight-update equations. The missclassified example used to update an existing weight is located in the input data set loaded to budgetedData.

5.12.3.10 void budgetedVectorAMM::updateUsingVector (budgetedVectorAMM * *otherVector,* unsigned int *oto,* int *sign,* parameters * *param*)

Updates a weight-vector when misclassification happens.

Parameters

in	*otherVector*	Misclassified example used to update the existing weight.
in	*oto*	Total number of iterations so far.
in	*sign*	+1 if the input vector is of the true class, -1 otherwise, specifies how the weights will be updated.
in	*param*	The parameters of the algorithm.

When we misclassify a data point during training, this function is used to update the existing weight-vector. It brings the true-class weight closer to the misclassified data point, and to push the winning other-class weight away from the misclassified point according to AMM weight-update equations. The missclassified example used to update an existing weight is located in the budgetedVectorAMM object.

5.12.4 Member Data Documentation

5.12.4.1 long double budgetedVectorAMM::degradation [protected]

Degradation of the vector.

At each iteration during the training procedure of AMM algorithms and Pegasos all weights are degraded, meaning that their elements are pushed slightly towards 0. This can, in addition to numerical issues, also be a problem when the dimensionality of the data set is large, as in naive implementation each feature needs to be degraded independently. However, instead of degrading each element separately, we can keep degradation level as a single number which is the same for all features, thus avoiding round-off problems and also speeding up the degradation step, which now amounts to a single multiplication operation.

Consequently, the actual feature value of a vector is equal to the value stored in array, multiplied by degradation.

The documentation for this class was generated from the following files:

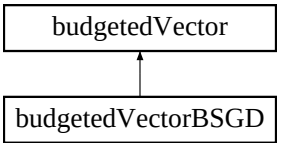
- mm_algs.h
- mm_algs.cpp

5.13 budgetedVectorBSGD Class Reference

Class which holds sparse vector, which is split into a number of arrays to trade-off between speed of access and memory usage of sparse data, with added methods for BSGD algorithm.

`#include <bsgd.h>`

Inheritance diagram for budgetedVectorBSGD:



Public Member Functions

- void updateSV (budgetedVectorBSGD *v, long double kMax)

  Updates the vector to obtain a merged vector, used during merging budget maintenance.
- budgetedVectorBSGD (unsigned int dim=0, unsigned int chnkWght=0, unsigned int numCls=0)

  Constructor, initializes the vector to all zeros, and also initializes class-specific alpha parameters.
- long double alphaNorm (void)

  Computes the norm of alpha vector.
- void downgrade (unsigned long oto)

  Downgrade the alpha-parameters.

Static Public Member Functions

- static unsigned int getNumClasses (void)

  Get the number of classes in the classification problem.

Public Attributes

- vector< long double > alphas

  Array of class-specific alpha parameters, used in BSGD algorithm.

Static Protected Attributes

- static unsigned int numClasses = 0

  Number of classes of the classification problem, specifies the size of alphas vector.

Friends

- class **budgetedModelBSGD**
- class **budgetedModelMatlabBSGD**

Additional Inherited Members

5.13.1 Detailed Description

Class which holds sparse vector, which is split into a number of arrays to trade-off between speed of access and memory usage of sparse data, with added methods for BSGD algorithm.

5.13.2 Constructor & Destructor Documentation

5.13.2.1 budgetedVectorBSGD::budgetedVectorBSGD (unsigned int *dim* = 0, unsigned int *chnkWght* = 0, unsigned int *numCls* = 0) [inline]

Constructor, initializes the vector to all zeros, and also initializes class-specific alpha parameters.

Parameters

in	*dim*	Dimensionality of the vector.
in	*chnkWght*	Size of each vector chunk.
in	*numCls*	Number of classes in the classification problem, specifies the size of alphas vector.

5.13.3 Member Function Documentation

5.13.3.1 long double budgetedVectorBSGD::alphaNorm (void)

Computes the norm of alpha vector.

Returns

Norm of the alpha vector.

Computes the l2-norm of the alpha vector.

See Also

budgetedVector::alphas

5.13.3.2 void budgetedVectorBSGD::downgrade (unsigned long *oto*) [inline]

Downgrade the alpha-parameters.

Parameters

in	*oto*	Total number of iterations so far.

Each training iteration the alpha parameters are pushed towards 0 to ensure the convergence of the algorithm to the optimal solution.

5.13.3.3 unsigned int budgetedVectorBSGD::getNumClasses (void) [inline],[static]

Get the number of classes in the classification problem.

Returns

Number of classes that are covered by this vector, also the length of alphas.

5.13.3.4 void budgetedVectorBSGD::updateSV (budgetedVectorBSGD * *v*, long double *kMax*)

Updates the vector to obtain a merged vector, used during merging budget maintenance.

Parameters

in	*v*	Vector that is merged with this vector.
in	*kMax*	Parameter that specifies how to combine them (currentVector <- kMax * currentVector + (1 - kMax) * v).

When we find which two support vectors to merge, together with the value of the merging parameter kMax, this function updates one of the two vectors to obtain the merged support vector. After the merging, the other vector is no longer needed and can be deleted.

See Also

computeKmax

5.13.4 Member Data Documentation

5.13.4.1 vector< double > budgetedVectorBSGD::alphas

Array of class-specific alpha parameters, used in BSGD algorithm.

This vector is of the size that equals number of classes in the data set. Each element specifies the influence a budgetedVector has on a specific class.

The documentation for this class was generated from the following files:

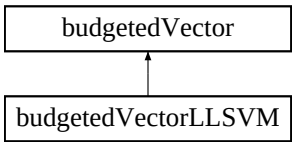
- bsgd.h
- bsgd.cpp

5.14 budgetedVectorLLSVM Class Reference

Class which holds sparse vector, which is split into a number of arrays to trade-off between speed of access and memory usage of sparse data, with added methods for LLSVM algorithm.

`#include <llsvm.h>`

Inheritance diagram for budgetedVectorLLSVM:



Public Member Functions

- void createVectorUsingDataPointMatrix (VectorXd &dataVector)

  Initialize the vector using a data point represented as a (1 x DIMENSION) matrix.
- budgetedVectorLLSVM (unsigned int dim=0, unsigned int chnkWght=0)

  Constructor, initializes the LLSVM vector to zero weights.

Friends

- class **budgetedModelLLSVM**
- class **budgetedModelMatlabLLSVM**

Additional Inherited Members

5.14.1 Detailed Description

Class which holds sparse vector, which is split into a number of arrays to trade-off between speed of access and memory usage of sparse data, with added methods for LLSVM algorithm.

5.14.2 Member Function Documentation

5.14.2.1 void budgetedVectorLLSVM::createVectorUsingDataPointMatrix (VectorXd & *dataVector*) [inline]

Initialize the vector using a data point represented as a (1 x DIMENSION) matrix.

Parameters

in	*dataVector*	Row vector holding a data point.

Used during the initialization stage of the LLSVM algorithm to store the found landmark point in an instance of budgetedVectorLLSVM class.

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

- llsvm.h

5.15 parameters Struct Reference

Structure holds the parameters of the implemented algorithms.

`#include <budgetedSVM.h>`

Public Member Functions

- parameters (void)

  Constructor of the structure. The default values of the parameters can be modified here manually.

- void updateVerySparseDataParameter (double dataSparsity)

  If VERY_SPARSE_DATA parameter was not set by a user, this function sets this parameter according to the sparsity of the loaded data.

Public Attributes

- unsigned int ALGORITHM

  Algorithm that is used, 0 - Pegasos; 1 - AMM batch; 2 - AMM online; 3 - LLSVM; 4 - BSGD (default: 2)
- unsigned int NUM_SUBEPOCHS

  Number of training subepochs of AMM batch algorithm (default: 1)
- unsigned int NUM_EPOCHS

  Number of training epochs (default: 5)
- unsigned int K_PARAM

  Frequency of weight pruning of AMM algorithms (default: 10,000 iterations)
- unsigned int DIMENSION

  Dimensionality of the classification problem, MUST be set by a user (default: 0)
- unsigned int CHUNK_SIZE

  Size of the chunk of the data loaded at once (default: 50,000 data points)
- unsigned int CHUNK_WEIGHT

  Size of chunk of budgetedVector weight (whole vector is split into smaller parts) (default: 1,000)
- unsigned int LIMIT_NUM_WEIGHTS_PER_CLASS

  Maximum number of weight per class of AMM algorithms (default: 20)
- unsigned int BUDGET_SIZE

  Size of the budget of BSGD algorithm, OR number of landmark points in LLSVM algorithm (default: 100)
- unsigned int K_MEANS_ITERS

  Number of k-means iterations in initialization of LLSVM algorithm (default: 10)
- unsigned int MAINTENANCE_SAMPLING_STRATEGY

  Budget maintenance strategy of BSGD algorithm, 0 - random removal; 1 - merging, OR type of landmark points sampling in LLSVM algorithm, 0 - random; 1 - k-means; 2 - k-medoids (default: 0)
- unsigned int VERY_SPARSE_DATA

  User set parameter, if a user believes the data is very sparse this parameters can be set to 0/1, where 1 - very sparse data; 0 - not very sparse data (default: see long description)
- double C_PARAM

  Weight pruning parameter of AMM algorithms, OR linear-SVM regularization parameter used in LLSVM (default: 10.0)
- double BIAS_TERM

  Bias term of AMM batch, AMM online, and PEGASOS algorithms (default: 1.0)
- double GAMMA_PARAM

  *Kernel width parameter in Gaussian kernel exp(-0.5 * GAMMA_PARAM * ||x - y||^2) (default: 1/DIMENSIONALITY)*
- double LAMBDA_PARAM

  Lamdba parameter of AMM algorithms (default: 0.0001)
- bool VERBOSE

  Print verbose output during algorithm execution, 1 - verbose output; 0 - quiet (default: 0)

5.15.1 Detailed Description

Structure holds the parameters of the implemented algorithms.

Structure holds the parameters of the implemented algorithms. If needed, the default parameters for each algorithm can be manually modified here.

5.15.2 Member Function Documentation

5.15.2.1 void parameters::updateVerySparseDataParameter (double *dataSparsity*) [inline]

If VERY_SPARSE_DATA parameter was not set by a user, this function sets this parameter according to the sparsity of the loaded data.

Parameters

in	*dataSparsity*	The sparsity of the loaded data set.

When computing the kernels between support vectors/hyperplanes kept in the available budget in budgetedVector objects on one side, and the incoming data points on the other, we have two options: (1) we can either do the computations directly between the support vectors and data points that are stored in budgetedData; or (2) we can do the computations between the support vectors and data points that are in the intermediate step stored in the budgetedVector object. When the data is very sparse option (1) is faster, as there is very small number of non-zero features that affects the speed of the computations, and the overhead of creating the budgetedVector instance might prove too costly. On the other hand, when the data is not too sparse, then it might prove faster to first create budgetedVector that will hold the incoming data point, and only then do the kernel computations. The reason is partly in a slow modulus operation that is used in the case (1) (please refer to the implementation of linear and Gaussian kernels to see how it was coded.

See Also

VERY_SPARSE_DATA, budgetedVector::linearKernel(unsigned int, budgetedData∗, parameters∗), budgetedVector::linearKernel(budgetedVector∗), budgetedVector::gaussianKernel(unsigned int, budgetedData∗, parameters∗, long double), budgetedVector::gaussianKernel(budgetedVector∗, parameters∗)

5.15.3 Member Data Documentation

5.15.3.1 double parameters::BIAS_TERM

Bias term of AMM batch, AMM online, and PEGASOS algorithms (default: 1.0)

If the parameter is non-zero, a bias, or intercept term, is added to the data set as an additional feature. The value of this additional feature is equal to BIAS_TERM.

5.15.3.2 unsigned int parameters::BUDGET_SIZE

Size of the budget of BSGD algorithm, OR number of landmark points in LLSVM algorithm (default: 100)

- BSGD: Maximum number of support vectors that can be stored. After the budget is exceeded, MAINTENANCE_SAMPLING_STRATEGY specifies how the number of support vectors is kept limited.
- LLSVM: In addition, it also specifies the number of landmark points in LLSVM algorithm, that are used to represent the data set in lower-dimensional space using the Nystrom method.

5.15.3.3 double parameters::C_PARAM

Weight pruning parameter of AMM algorithms, OR linear-SVM regularization parameter used in LLSVM (default: 10.0)

- AMM: In order to reduce the complexity of the learned model, which directly improves generalization of the model as shown in the AMM paper, pruning of small non-zero weights is performed. C_PARAM specifies the aggressiveness of weight pruning, where larger value results in pruning of more weights. More specifically, we sort the weights by their L2-norms, and then prune from the smallest toward larger weight until

the cummulative weight norm exceeds value of C_PARAM. Frequency of pruning is controlled by K_PARAM parameter.

- LLSVM: Regularization parameter used in the objective function of SVM, which is used in the LLSVM algorithm to find the best hyperplane separating the classes after mapping from the input feature space to the new, linearized feature space. Larger values result in more complex model.

5.15.3.4 unsigned int parameters::CHUNK_SIZE

Size of the chunk of the data loaded at once (default: 50,000 data points)

While CHUNK_WEIGHT helps when one is working with high-dimensional data, this parameter helps when working with large data with many instances. If the data set is very large and can not fit into memory, we can then load only a small part of it (called *data chunk*), that is processed before being discarded to make room for the next chunk. Therefore, we load only a smaller part of the large data set, with size of this chunk specified by this parameter.

5.15.3.5 unsigned int parameters::CHUNK_WEIGHT

Size of chunk of budgetedVector weight (whole vector is split into smaller parts) (default: 1,000)

While CHUNK_SIZE helps when one is working with large data with many data points, this parameter helps when working with high-dimensional data. When the data is sparse, then we do not have to explicitly store every feature as most of them are equal to 0. One option is simply to follow LIBSVM format, and store a vector in two linked lists, one holding feature index and the other holding the corresponding feature value. However, we found that accessing this data structure can become prohibitively slow, as for high-dimensional data weights can become less sparse than the original data due to the weight update process. For example, when we want to update a specific feature during gradient descent training we would like to do it very quickly, most preferably we would like to have random access to the element of the weight vector that will be updated. We address this by storing a vector into linked list, where each element of the linked list, called *weight chunk*, holding a subset of features. For example, the first chunk would hold features indexed from 1 to CHUNK_SIZE, the second would hold features indexed from CHUNK_SIZE+1 to 2∗CHUNK_SIZE, and so on. If all elements of a weight chunk are zero, we do not allocate memory for that array. In our experience, this significantly improved the training and testing time on truly high-dimensional data, such as on URL data set with more than 3.2 million features. If CHUNK_WEIGHT is equal to 1, we obtain the LIBSVM-type representation.

5.15.3.6 unsigned int parameters::DIMENSION

Dimensionality of the classification problem, MUST be set by a user (default: 0)

Although the dimensionality of the data set can be found from the training data set during loading, we ask a user to specify it beforehand, as it is usually a known parameter. The reason why we require this as an input is to speed up processing of the data, since the emphasis of the software is on speeding up the training of classification algorithm on large data, and this little piece of information can help avoid unnecessary bookkeeping tasks. More specifically, the parameter is important for memory management of budgetedVector, where it is used to find how many weight chunks of size CHUNK_WEIGHT are needed to represent the data.

However, in the case of Matlab interface, it is not required to manually set this parameter as it is easily found by reading the dimensions of the Matlab structure holding the data set.

5.15.3.7 unsigned int parameters::K_MEANS_ITERS

Number of k-means iterations in initialization of LLSVM algorithm (default: 10)

In order to find better lower-dimensional representation of the data set using Nystrom method, k-means can be used to improve the choice of landmark points. Unlike in random sampling of landmark points from the data set, cluster centers of k-means will represent BUDGET_SIZE points used for the Nystrom method.

5.15.3.8 unsigned int parameters::K_PARAM

Frequency of weight pruning of AMM algorithms (default: 10,000 iterations)

In order to reduce the complexity of the learned model, which directly improves generalization of the model as shown in the AMM paper, pruning of small non-zero weights is performed. K_PARAM specifies the frequency of weight pruning, i.e., after how many iterations we perform the pruning step. Aggressiveness of pruning is controlled by C_PARAM parameter.

5.15.3.9 double parameters::LAMBDA_PARAM

Lamdba parameter of AMM algorithms (default: 0.0001)

The parameter defines the level of model regularization of AMM models, where larger values result in less complex model. The parameter is similar to C_PARAM parameter used in LLSVM when solving linear-SVM, as both parameters are used to set the level of regularization, only with opposite effect. Namely, while larger values of LAMBDA_PARAM result in less complex AMM model, larger values of C_PARAM result in more complex LLSVM model.

5.15.3.10 unsigned int parameters::LIMIT_NUM_WEIGHTS_PER_CLASS

Maximum number of weight per class of AMM algorithms (default: 20)

As the number of weights in AMM algorithms is infinite, we can set the limit on the number of non-zero weights that can be stored in memory. This can be done in order to avoid memory-related problems. Once the limit is reached, we do not allow creation of new non-zero weights until some get pruned.

5.15.3.11 unsigned int parameters::MAINTENANCE_SAMPLING_STRATEGY

Budget maintenance strategy of BSGD algorithm, 0 - random removal; 1 - merging, OR type of landmark points sampling in LLSVM algorithm, 0 - random; 1 - k-means; 2 - k-medoids (default: 0)

- BSGD: Whenever a number of support vectors in BSGD algorithm exceeds BUDGET_SIZE, one of the following budget maintenance steps is performed, depending on the value of the MAINTENANCE_SAMPLING-_STRATEGY parameter
  - 0 - deleting random support vector to maintain the budget
  - 1 - take two support vectors and merging them into one. The new, merged support vector is located on the straight line connecting the two existing support vectors; where exactly on the line is explained in *computeKmax()* function from *bsdg.cpp* file. Then, the two existing support vectors are deleted and the merged vector is inserted in the budget. (default setting)

- LLSVM: Specifies how the landmark points, used to represent the data set in lower-dimensional space using the Nystrom method, are chosen.
  - 0 - landmark points are randomly sampled from the the first loaded data chunk
  - 1 - landmark points will be cluster centers after running k-means on the first loaded data chunk (default setting)
  - 2 - landmark points will be cluster medoids after running k-medoids on the first loaded data chunk

5.15.3.12 unsigned int parameters::NUM_EPOCHS

Number of training epochs (default: 5)

Number of times the data set is seen by the training procedure, each time randomly reshuffled.

5.15.3.13 unsigned int parameters::NUM_SUBEPOCHS

Number of training subepochs of AMM batch algorithm (default: 1)

AMM batch has an option to reassign data points to weights several times during one epoch. In the most extreme case, if NUM_SUBEPOCHS is equal to the size of the data set, we obtain AMM online algorithm. This parameter specifies how many times we reassign points to weights within a single epoch.

5.15.3.14 unsigned int parameters::VERY_SPARSE_DATA

User set parameter, if a user believes the data is very sparse this parameters can be set to 0/1, where 1 - very sparse data; 0 - not very sparse data (default: see long description)

When computing the kernels between support vectors/hyperplanes kept in the available budget in budgetedVector objects on one side, and the incoming data points on the other, we have two options: (1) we can either do the computations directly between the support vectors and data points that are stored in budgetedData; or (2) we can do the computations between the support vectors and data points that are in the intermediate step stored in the budgetedVector object. When the data is very sparse option (1) is faster, as there is very small number of non-zero features that affects the speed of the computations, and the overhead of creating the budgetedVector instance might prove too costly. On the other hand, when the data is not too sparse, then it might prove faster to first create budgetedVector that will hold the incoming data point, and only then do the kernel computations. The reason is partly in a slow modulus operation that is used in the case (1) (please refer to the implementation of linear and Gaussian kernels to see how it was coded).

If a user does not manually set this parameter to 0 (i.e., instructs the toolbox to compute kernels as in case (1)) or 1 (i.e., compute kernels as in case (2)), the default setting will be 0 if the sparsity of the loaded data is less than 5% (i.e., less than 5% of the features are non-zero on average), otherwise it will default to 1. For this default behavior that is adaptive to the found data sparsity a developer can set this parameter to anything other than 0 or 1. For more details, please see the train and test functions of the implemented algorithms, and look for code parts where VERY_SPARSE_DATA appears.

See Also

updateVerySparseDataParameter(), budgetedVector::linearKernel(unsigned int, budgetedData∗, parameters∗), budgetedVector::linearKernel(budgetedVector∗), budgetedVector::gaussianKernel(unsigned int, budgetedData∗, parameters∗, long double), budgetedVector::gaussianKernel(budgetedVector∗, parameters∗)

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

- budgetedSVM.h

Chapter 6

File Documentation

6.1 bsgd.h File Reference

Defines classes and functions used for training and testing of BSGD (Budgeted Stochastic Gradient Descent) algorithm.

Classes

- class budgetedVectorBSGD

  Class which holds sparse vector, which is split into a number of arrays to trade-off between speed of access and memory usage of sparse data, with added methods for BSGD algorithm.
- class budgetedModelBSGD

  Class which holds the BSGD model (comprising the support vectors stored as budgetedVectorBSGD), and implements methods to load BSGD model from and save BSGD model to text file.

Functions

- void trainBSGD (budgetedData ∗trainData, parameters ∗param, budgetedModelBSGD ∗model)

  Train BSGD.
- float predictBSGD (budgetedData ∗testData, parameters ∗param, budgetedModelBSGD ∗model, vector< char > ∗labels)

  Given a BSGD model, predict the labels of testing data.

6.1.1 Detailed Description

Defines classes and functions used for training and testing of BSGD (Budgeted Stochastic Gradient Descent) algorithm.

6.1.2 Function Documentation

6.1.2.1 float predictBSGD (budgetedData ∗ *testData,* parameters ∗ *param,* budgetedModelBSGD ∗ *model,* vector< char > ∗ *labels*)

Given a BSGD model, predict the labels of testing data.

Parameters

in	*testData*	Input test data.
in	*param*	The parameters of the algorithm.
in	*model*	Trained BSGD model.
out	*labels*	Vector of predicted labels.

Returns

Testing set error rate.

Given the learned BSGD model, the function computes the predictions on the testing data, outputing the predicted labels and the error rate.

6.1.2.2 void trainBSGD (budgetedData * *trainData*, parameters * *param*, budgetedModelBSGD * *model*)

Train BSGD.

Parameters

in	*trainData*	Input training data.
in	*param*	The parameters of the algorithm.
in,out	*model*	Initial BSGD model.

The function trains BSGD model, given input data, the initial model (most often zero-weight model), and the parameters of the model.

6.2 budgetedSVM.h File Reference

Header file defining classes and functions used throughout the budgetedSVM toolbox.

Classes

- struct parameters

  Structure holds the parameters of the implemented algorithms.
- class budgetedData

  Class which handles manipulation of large data sets that cannot be fully loaded to memory (using a data structure similar to Matlab's sparse matrix structure).
- class budgetedVector

  Class which handles high-dimensional vectors.
- class budgetedModel

  Interface which defines methods to load model from and save model to text file.

Macros

- #define INF HUGE_VAL

  Large (infinite) value, similar to Matlab's Inf.

Typedefs

- typedef void(* funcPtr)(const char *text)

  Defines pointer to a function that prints information for a user, defined for more clear code.

Enumerations

- enum {
  PEGASOS, **AMM_BATCH**, **AMM_ONLINE**, **LLSVM**,
  BSGD }

  Available large-scale, non-linear algorithms (note: unlike other algorithms, PEGASOS is a linear SVM solver).

Functions

- void svmPrintString (const char ∗text)

  Prints string to the output.
- void setPrintStringFunction (funcPtr printFunc)

  Modifies a callback that prints a string.
- void svmPrintErrorString (const char ∗text)

  Prints error string to the output.
- void setPrintErrorStringFunction (funcPtr printFunc)

  Modifies a callback that prints an error string.
- bool fgetWord (FILE ∗fHandle, char ∗str)

  Reads one word string from an input file.
- bool readableFileExists (const char fileName[])

  Checks if the file, identified by the input parameter, exists and is available for reading.
- void parseInputPrompt (int argc, char ∗∗argv, bool trainingPhase, char ∗inputFile, char ∗modelFile, char ∗outputFile, parameters ∗param)

  Parses the user input from command prompt and modifies parameter settings as necessary, taken from LIBLINEAR implementation.
- void printUsagePrompt (bool trainingPhase, parameters ∗param)

  Prints the instructions on how to use the software to standard output.

6.2.1 Detailed Description

Header file defining classes and functions used throughout the budgetedSVM toolbox.

6.2.2 Function Documentation

6.2.2.1 bool fgetWord (FILE ∗ *fHandle*, char ∗ *str*)

Reads one word string from an input file.

Parameters

in	*fHandle*	Handle to an open file from which one word is read.
out	*str*	A character string that will hold the read word.

Returns

True if end-of-line or end-of-file encountered after reading a word string, otherwise false.

The function is similar to C++ functions fgetc() and getline(), only that it reads a single word from a text file. For the purposes of this project, a word is defined as a sequence of characters that does not contain a white-space character or new-line character ’

’. As a model in BudgetedSVM is stored in a text file where each line may corresponds to a single support vector, it is also useful to know if we reached the end of the line or the end of the file, which is indicated by the return value of the function.

6.2.2.2 void parseInputPrompt (int *argc,* char ** *argv,* bool *trainingPhase,* char * *inputFile,* char * *modelFile,* char * *outputFile,* **parameters** * *param*)

Parses the user input from command prompt and modifies parameter settings as necessary, taken from LIBLINEAR implementation.

Parameters

in	*argc*	Argument count.
in	*argv*	Argument vector.
in	*trainingPhase*	True for training phase parsing, false for testing phase.
out	*inputFile*	Filename of input data file.
out	*modelFile*	Filename of model file.
out	*outputFile*	Filename of output file (only used during testing phase).
out	*param*	Parameter object modified by user input.

6.2.2.3 void printUsagePrompt (bool *trainingPhase,* **parameters** * *param*)

Prints the instructions on how to use the software to standard output.

Parameters

in	*trainingPhase*	Indicator if training or testing phase instructions.
in	*param*	Parameter object modified by user input.

6.2.2.4 bool readableFileExists (const char *fileName[]*)

Checks if the file, identified by the input parameter, exists and is available for reading.

Parameters

in	*fileName*	Handle to an open file from which one word is read.

Returns

True if the file exists and is available for reading, otherwise false.

6.2.2.5 void setPrintErrorStringFunction (**funcPtr** *printFunc*)

Modifies a callback that prints an error string.

Parameters

in	*printFunc*	New text-printing function.

This function is used to modify the function that is used to print to error output. After calling this function, which modifies the callback function for printing error string, the text is printed simply by invoking svmPrintErrorString.

See Also

funcPtr

6.2.2.6 void setPrintStringFunction (funcPtr *printFunc*)

Modifies a callback that prints a string.

Parameters

in	*printFunc*	New text-printing function.

This function is used to modify the function that is used to print to standard output. After calling this function, which modifies the callback function for printing, the text is printed simply by invoking svmPrintString.

See Also

funcPtr

6.2.2.7 void svmPrintErrorString (const char * *text*)

Prints error string to the output.

Parameters

in	*text*	Text to be printed.

Prints error string to the output. Exactly to which output should be specified by setPrintErrorStringFunction, which modifies the callback that is invoked for printing. This is convinient when an error is detected and, prior to printing appropriate message to a user, we want to exit the program. For example on how to set the printing function in Matlab environment, see the implementation of parseInputMatlab.

6.2.2.8 void svmPrintString (const char * *text*)

Prints string to the output.

Parameters

in	*text*	Text to be printed.

Prints string to the output. Exactly to which output should be specified by setPrintStringFunction, which modifies the callback that is invoked for printing. This is convinient when simple printf() can not be used, for example if we want to print to Matlab prompt. For example on how to set the printing function in Matlab environment, see the implementation of parseInputMatlab.

6.3 budgetedSVM_matlab.h File Reference

Implements classes and functions used for training and testing of budgetedSVM algorithms in Matlab.

Classes

- class budgetedDataMatlab

  Class which manipulates sparse array of vectors (similarly to Matlab sparse matrix structure), with added functionality to load data directly from Matlab.
- class budgetedModelMatlab

  Interface which defines methods to load model from and save model to Matlab environment.
- class budgetedModelMatlabAMM

Class which holds the AMM model, and implements methods to load AMM model from and save AMM model to Matlab environment.

- class budgetedModelMatlabBSGD

  Class which holds the BSGD model, and implements methods to load BSGD model from and save BSGD model to Matlab environment.

- class budgetedModelMatlabLLSVM

  Class which holds the LLSVM model, and implements methods to load LLSVM model from and save LLSVM model to Matlab environment.

Functions

- void printStringMatlab (const char ∗s)

  Prints string to Matlab, used to modify callback in budgetedSVM.cpp.

- void printErrorStringMatlab (const char ∗s)

  Prints error string to Matlab, used to modify callback found in budgetedSVM.cpp.

- void fakeAnswer (mxArray ∗plhs[])

  Returns empty matrix to Matlab.

- void printUsageMatlab (bool trainingPhase, parameters ∗param)

  Prints to standard output the instructions on how to use the software.

- void parseInputMatlab (parameters ∗param, const char ∗paramString, bool trainingPhase, const char ∗inputFileName=NULL, const char ∗modelFileName=NULL)

  Parses the user input and modifies parameter settings as necessary.

6.3.1 Detailed Description

Implements classes and functions used for training and testing of budgetedSVM algorithms in Matlab.

6.3.2 Function Documentation

6.3.2.1 void fakeAnswer (mxArray ∗ *plhs[]*)

Returns empty matrix to Matlab.

Parameters

out	*plhs*	Pointer to Matlab output.

6.3.2.2 void parseInputMatlab (parameters ∗ *param*, const char ∗ *paramString*, bool *trainingPhase*, const char ∗ *inputFileName*, const char ∗ *modelFileName*)

Parses the user input and modifies parameter settings as necessary.

Parameters

out	*param*	Parameter object modified by user input.
in	*paramString*	User-provided parameter string, can be NULL in which case default parameters are used..
in	*trainingPhase*	Indicator if training or testing phase.
in	*inputFileName*	User-provided filename with input data (if NULL no check of filename validity).
in	*modelFileName*	User-provided filename with learned model (if NULL no check of filename validity).

6.3.2.3 void printErrorStringMatlab (const char * s)

Prints error string to Matlab, used to modify callback found in budgetedSVM.cpp.

Parameters

in	*s*	Text to be printed.

6.3.2.4 void printStringMatlab (const char * s)

Prints string to Matlab, used to modify callback in budgetedSVM.cpp.

Parameters

in	*s*	Text to be printed.

6.3.2.5 void printUsageMatlab (bool *trainingPhase,* parameters * *param*)

Prints to standard output the instructions on how to use the software.

Parameters

in	*trainingPhase*	Indicator if training or testing phase.
in	*param*	Parameter object modified by user input.

6.4 llsvm.h File Reference

Defines classes and functions used for training and testing of LLSVM algorithm.

Classes

- class budgetedVectorLLSVM

  Class which holds sparse vector, which is split into a number of arrays to trade-off between speed of access and memory usage of sparse data, with added methods for LLSVM algorithm.
- class budgetedModelLLSVM

  Class which holds the LLSVM model, and implements methods to load LLSVM model from and save LLSVM model to text file.

Functions

- void trainLLSVM (budgetedData *trainData, parameters *param, budgetedModelLLSVM *model)

  Train LLSVM online.
- float predictLLSVM (budgetedData *testData, parameters *param, budgetedModelLLSVM *model, vector< char > *labels)

  Given an LLSVM model, predict the labels of testing data.

6.4.1 Detailed Description

Defines classes and functions used for training and testing of LLSVM algorithm.

6.4.2 Function Documentation

6.4.2.1 float predictLLSVM (budgetedData * *testData*, parameters * *param*, budgetedModelLLSVM * *model*, vector< char > * *labels*)

Given an LLSVM model, predict the labels of testing data.

Parameters

in	*testData*	Input test data.
in	*param*	The parameters of the algorithm.
in	*model*	Trained LLSVM model.
out	*labels*	Vector of predicted labels.

Returns

Testing set error rate.

Given the learned BSGD model, the function computes the predictions on the testing data, outputing the predicted labels and the error rate.

6.4.2.2 void trainLLSVM (budgetedData * *trainData*, parameters * *param*, budgetedModelLLSVM * *model*)

Train LLSVM online.

Parameters

in	*trainData*	Input training data.
in	*param*	The parameters of the algorithm.
in,out	*model*	Initial LLSVM model.

The function trains LLSVM model, given input data, the initial model (most often zero-weight model), and the parameters of the model.

6.5 mm_algs.h File Reference

Defines classes and functions used for training and testing of large-scale multi-hyperplane algorithms (AMM batch, AMM online, and Pegasos).

Classes

- class budgetedVectorAMM

  Class which holds sparse vector, which is split into a number of arrays to trade-off between speed of access and memory usage of sparse data, with added methods for AMM algorithms.
- class budgetedModelAMM

  Class which holds the AMM model, and implements methods to load AMM model from and save AMM model to text file.

Typedefs

- typedef vector < budgetedVectorAMM * > vectorOfBudgetVectors

  A vector of vectors, implements the weight matrix of AMM algorithms as jagged array.

Functions

- void trainPegasos (budgetedData *trainData, parameters *param, budgetedModelAMM *model)

  Train Pegasos.
- void trainAMMonline (budgetedData *trainData, parameters *param, budgetedModelAMM *model)

  Train AMM online.
- void trainAMMbatch (budgetedData *trainData, parameters *param, budgetedModelAMM *model)

  Train AMM batch.
- float predictAMM (budgetedData *testData, parameters *param, budgetedModelAMM *model, vector< char > *labels)

  Given a multi-hyperplane machine (MM) model, predict the labels of testing data.

6.5.1 Detailed Description

Defines classes and functions used for training and testing of large-scale multi-hyperplane algorithms (AMM batch, AMM online, and Pegasos).

6.5.2 Function Documentation

6.5.2.1 float predictAMM (budgetedData * *testData,* parameters * *param,* budgetedModelAMM * *model,* vector< char > * *labels*)

Given a multi-hyperplane machine (MM) model, predict the labels of testing data.

Parameters

in	*testData*	Input test data.
in	*param*	The parameters of the algorithm.
in	*model*	Trained MM model.
out	*labels*	Vector of predicted labels.

Returns

Testing set error rate.

Given the learned multi-hyperplane machine, the function computes the predictions on the testing data, outputing the predicted labels and the error rate.

6.5.2.2 void trainAMMbatch (budgetedData * *trainData,* parameters * *param,* budgetedModelAMM * *model*)

Train AMM batch.

Parameters

in	*trainData*	Input training data.
in	*param*	The parameters of the algorithm.
in,out	*model*	Initial AMM model.

The function trains multi-hyperplane machine using AMM batch algotihm, given input data, the initial model (most often zero-weight model), and the parameters of the model.

6.5.2.3 void trainAMMonline (budgetedData * *trainData,* parameters * *param,* budgetedModelAMM * *model*)

Train AMM online.

Parameters

in	*trainData*	Input training data.
in	*param*	The parameters of the algorithm.
in,out	*model*	Initial AMM model.

The function trains multi-hyperplane machine using AMM online algotihm, given input data, the initial model (most often zero-weight model), and the parameters of the model.

6.5.2.4 void trainPegasos (**budgetedData** * *trainData,* **parameters** * *param,* **budgetedModelAMM** * *model*)

Train Pegasos.

Parameters

in	*trainData*	Input training data.
in	*param*	The parameters of the algorithm.
in,out	*model*	Initial Pegasos model.

The function trains Pegasos model, given input data, initial model (most often zero-weight model), and the parameters of the model.

Index