SLEP: Sparse Learning with Efficient Projections

Version 4.1

Jun Liu, Shuiwang Ji, and Jieping Ye

Computer Science and Engineering
Center for Evolutionary Medicine and Informatics
The Biodesign Institute
Arizona State University
Tempe, AZ 85287

{j.liu, shuiwang.ji, jieping.ye}@asu.edu
http://www.public.asu.edu/~jye02/Software/SLEP

December 28, 2011

Contents

List of Tables

1 Introduction

The underlying representations of many real-world processes are often sparse. For example, in disease diagnosis, even though humans have a large number of genes, only a small number of them contribute to certain disease [16, 17]. In neuroscience, the neural representation of sounds in the auditory cortex of unanesthetized animals is sparse, since the fraction of neurons that are active at a given instant is typically small [20]. In signal processing, many natural signals are sparse in that they have concise representations when expressed under a proper basis [7]. Therefore, finding sparse representations is fundamentally important in many fields of science. The last decade has witnessed a growing interest in the search for sparse representations of data.

ℓ_1-Norm Regularization Most existing work on sparse learning are based on a variant of the ℓ_1 norm regularization due to its sparsity-inducing property, convenient convexity, strong theoretical guarantees, and great empirical success in various applications. The use of the ℓ_1 norm regularization has led to sparse models for linear regression [69], principle component analysis [83], linear discriminant analysis [76], canonical correlation analysis [65, 75], partial least squares [28], support vector machines [82], and logistic regression [27, 35, 64, 68].

ℓ_1/ℓ_q-Norm Regularization Recent studies in areas such as machine learning and statistics have also witnessed growing interests in extending the ℓ_1-regularization to the ℓ_1/ℓ_q-regularization [2, 3, 6, 11, 31, 32, 33, 34, 36, 44, 48, 55, 56, 59, 80, 81]. The ℓ_1/ℓ_q-regularization belongs to the composite absolute penalties (CAP) [81] family. When $q > 1$, the ℓ_1/ℓ_q-regularization facilitates group sparsity in the resulting model, which is desirable in many applications of regression and classification.

Fused Lasso Regularization The fused Lasso penalty introduced in [70] yields a solution that has sparsity in both the coefficients and their successive differences. It has found applications in comparative genomic hybridization [60, 71], prostate cancer analysis [70], image denoising [13], and time-varying networks [1], where features can be ordered in some meaningful way. Some properties of the fused Lasso have been established in [63].

Sparse Group Lasso As an extension of Lasso and group Lasso, the sparse group Lasso penalty [15, 57] yields a solution that achieves the within- and between- group sparsity simultaneously. That is, many feature groups are exactly zero (thus not selected) and within the non-zero (thus selected) feature groups, some features are also exactly zero. The simultaneous within- and between- group sparsity makes sparse group Lasso ideal for applications where we are interested in identifying important groups as well as important features within the selected groups. Sparse group Lasso is a special case of the tree structured group Lasso to be discussed next.

Tree Structured Group Lasso The past few years have witnessed increasing interests in structured sparsity [22, 23, 24, 26, 42, 81]. In the tree structured group Lasso, the structure over the features is represented as a tree with leaf nodes as features and internal nodes as clusters of the features. The structured regularization with a pre-defined tree structure is based on a group-Lasso penalty, where one group is defined for each node in the tree. The tree structures can be found in many applications including multi-task learning [26] and image classification [24, 42].

Overlapping Group Lasso Group Lasso, sparse group Lasso, and tree structured group Lasso can be employed to achieve the group sparsity. However, group Lasso and sparse group Lasso are only restricted to the (predefined) non-overlapping groups of features; and the tree structured group Lasso is restricted to the tree structured groups. However, in some applications, a more flexible (overlapping) group structure is desired. For example, in the study of the breast cancer using the gene expression data [74], researchers have come

up with different approaches for organizing the genes into a set of overlapping genes, e.g., pathways [66] and edges [10]. There have been several recent attempts to study a more general formulation, where groups of features are given, potentially with overlaps between the groups [22, 23, 41, 81].

Ordered Tree–Nonnegative Max-Heap In many regression/classification problems, the features exhibit certain hierarchical or structural relationships, the usage of which can yield an interpretable model with improved regression/classification performance [81]. In the ordered tree structure, the following assumption is imposed: a given feature is selected for the given regression/classification task only if its parent node is selected. To incorporate such ordered tree structure, it is assumed that the model parameter follows the non-negative max-heap structure (see the discussion in Section 10). Such non-negative max-heap can induce the so-called "heredity principle" [9, 18, 49, 79], which has been proven effective for high-dimensional variable selection. In a recent study [30], Li et al. conducted a meta-analysis of 113 data sets from published factorial experiments and concluded that an overwhelming majority of these real studies conform with the heredity principles. The ordered tree structure is a special case of the non-negative garrote discussed in [79] when the hierarchical relationship is depicted by a tree. Therefore, the asymptotic properties established in [79] are applicable to the ordered tree structrue. Several related approaches that can incorporate the ordered tree structure include the Wedge approach [46] and the hierarchical group Lasso [81].

Trace-Norm Regularization The problem of minimizing the rank of a matrix variable subject to certain constraints arises in many fields including machine learning, automatic control, and image compression. For example, in collaborative filtering we are given a partially filled rating matrix and the task is to predict the missing entries. Since it is commonly believed that only a few factors contribute to an individual's tastes, it is natural to approximate the given rating matrix by a low-rank matrix. However, the matrix rank minimization problem is NP-hard in general due to the combinatorial nature of the rank function. A commonly-used convex relaxation of the rank function is the trace norm (nuclear norm) [12], defined as the sum of the singular values of the matrix, since it is the convex envelope of the rank function over the unit ball of spectral norm. A number of recent work has shown that the low rank solution can be recovered exactly via minimizing the trace norm under certain conditions [61, 62, 8]. Trace norm regularized problems can also be considered as a form of sparse learning, since the trace norm equals to the ℓ_1 norm of the vector consisting of singular values.

The optimization problems resulting from the above sparse learning formulations are challenging to solve, since the ℓ_1-norm, the ℓ_1/ℓ_q-norm, the fused Lasso penalty, the sparse group Lasso penalty, the tree structured group Lasso penalty, the overlapping group Lasso penalty, and the trace norm penalty are all nonsmooth. It is known that, the lower complexity bound for nonsmooth convex optimization by first-order black-box methods is $O(\frac{1}{\varepsilon^2})$ [50, 51, 53], i.e., it consumes $O(\frac{1}{\varepsilon^2})$ iterations for achieving an accuracy of ε. Subgradient descent [50, 53] has been proven to achieve the convergence rate of $O(\frac{1}{\sqrt{k}})$, and thus touches the aforementioned lower complexity bound. It is also known that, the lower complexity bound for smooth convex optimization by first-order black-box methods is $O(\frac{1}{\sqrt{\varepsilon}})$ [50, 52, 53], i.e., it consumes at least $O(\frac{1}{\sqrt{\varepsilon}})$ iterations for achieving an accuracy of ε. Nesterov's method [50, 53] (illustrated in Figure 1) has been proven to achieve the convergence rate of $O(\frac{1}{k^2})$, and thus touches the aforementioned lower complexity bound. Therefore, Nesterov's method is one of the optimal first-order black-box methods for smooth convex optimization. It is clear that, nonsmooth convex optimization is much more challenging than smooth convex optimization; and the convergence rate of smooth convex optimization can be significantly better than that of nonsmooth convex optimization.

Based on our previous work [21, 25, 35, 36, 37, 38, 39, 40, 41, 42, 43, 58, 67, 78], we have developed the SLEP (**S**parse **L**earning with **E**fficient **P**rojections) package, which provides functions for solving a family of sparse learning algorithms. The functions implemented in the SLEP package enjoy the convergence rate of $O(\frac{1}{k^2})$, although the objective function is non-smooth. The underlying reason is that, we

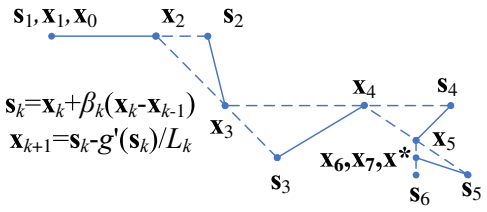


Figure 1: Illustration of the Nesterov's method. We set $\mathbf{x}_1 = \mathbf{x}_0$, and thus $\mathbf{s}_1 = \mathbf{x}_1$. The search point $\mathbf{s}_k$ is the affine combination of $\mathbf{x}_{k-1}$ and $\mathbf{x}_k$ (the dashed lines), and the next approximate solution is obtained by a gradient step of $\mathbf{s}_k$ (solid lines). We assume that $\mathbf{x}_6 = \mathbf{x}_7 = \mathbf{x}^*$, and $\mathbf{x}^*$ is an optimal solution.

utilize the "structures" of the aforementioned nonsmooth penalties via the so-called Euclidean projection (such Euclidean projection is closely related the Moreau-Yosida regularization [19, 47, 77] and the so-called proximal operator). Specifically, to achieve the optimal convergence rate of $O(\frac{1}{k^2})$, we make use of the following three techniques:

- We solve the equivalent smooth reformulation (subject to certain constraints) [35, 36] by the Nesterov's method [50, 53]. One key building block is the so-called Euclidean projection. The related Euclidean projections [36, 40] can be computed either analytically or in linear time.
- We solve the regularized non-smooth optimization problem[1] via the accelerated gradient method [5, 54]. One key building block is the so-called regularized Euclidean projection [39] (also called proximity operator [47]). The regularized Euclidean projection can be efficiently solved by single variable root finding problems [39].
- For the trace norm regularized problems, the proximal regularized linear problem at each step can be solved by applying a soft-thresholding operation on the singular values, and the overall algorithm converges at the same rate of $O(\frac{1}{k^2})$ as the smooth problems.

Sparse Inverse Covariance Estimation The pattern of zero entries in the inverse covariance matrix of a multivariate normal distribution corresponds to conditional independence restrictions between variables [45]. To obtain the sparse inverse covariance matrix, a sparse graphical model has been proposed based on the ℓ_1 regularization (of the inverse covariance matrix). The model has been applied for quite a few applications [4, 21, 67]. One of the state-of-the-art optimization algorithms is based on the block coordinate [73] (the readers are referred to [4, 14] for details). We implemented the Sparse Inverse Covariance Estimation based on the algorithm outlined in [14].

In the following subsections, we introduce the main features of the implemented functions, the usage of the package, and the applicability of the package.

[1] We assume that the loss function is smooth convex.

1.1 Main Features

- **First-order black-box method**

  At each iteration, we only need to evaluate the function value and gradient; and thus the algorithms can handle large-scale sparse data.

- **Optimal convergence rate**

  The convergence rate $O(1/k^2)$ (k denotes the iteration number) is optimal for smooth convex optimization via the first-order black-box methods [50, 53].

- **Efficient Euclidean projection**

  The Euclidean projection problem can be solved efficiently. For example, the Euclidean projection onto the ℓ_1 ball [40] of size 10^7 can be solved within 0.3 seconds on a PC.

- **Pathwise Solutions**

  The SLEP package provides functions that can efficiently compute the pathwise solutions corresponding to a series of regularization parameters using the "warm-start" technique.

1.2 How to Use the SLEP Package

The main functions were written in Matlab[2], while the codes for key subroutines–the related Euclidean projections—were written in C (located at /SLEP/CFiles/). To call the C file in Matlab, you need to mex[3] the corresponding C file. **For successfully using the functions provided in this package, you are suggested to first run the Matlab file "mexC"**[4] located at the root folder of the package.

When computing the solution corresponding to one single regularization parameter, you can call the functions as

$$[\mathbf{x}, \text{funVal}]=\text{functionName}(A, \mathbf{y}, \lambda, \text{opts})$$

$$[\mathbf{x}, c, \text{funVal}]=\text{functionName}(A, \mathbf{y}, \lambda, \text{opts})$$

for least squares loss and logistic loss, respectively. You can simply set opts=[][5]. For advanced usages (e.g., starting point, termination, normalization and regularization), you can specify opts with details provided in Section 11. The details on the input and output parameters can be found in Tables 1 & 2.

To efficiently compute the solutions corresponding to a series of regularization parameters $\boldsymbol{\lambda}$, you can use

$$\mathbf{X}=\text{pathSolutionLeast(functionName, } A, \mathbf{y}, \boldsymbol{\lambda}, \text{opts)}$$

$$[\mathbf{X}, \text{C}]=\text{pathSolutionLogistic(functionName, } A, \mathbf{y}, \boldsymbol{\lambda}, \text{opts)}$$

for computing the pathwise solutions for the least squares loss and the logistic loss, respectively. For details, please refer to Section 12.

[2] We are currently developing the C version, which shall be distributed later.

[3] `http://www.mathworks.com/support/tech-notes/1600/1605.html`

[4] For achieving better efficiency, we suggest using the up-to-date C compiler for usage in Matlab.

[5] For some functions discussed in Sections 3, 4, 5, 7, 8, 9, some fields such as opts.ind, opts.G, opts.fusedPenalty need to be specified for providing the additional information.

Table 1: Input parameters

Parameter	Description
A	The data matrix (dense, sparse, or partial DCT). Each row corresponds to a sample. $[m, n] = \text{size}(A)$.
$\mathbf{y}$	The response (column vector or matrix). The length of $\mathbf{y}$ equals to m. In the multi-class classification scenario with $k > 2$ classes, $\mathbf{y}$ is a matrix of size $m \times k$.
λ	The regularization parameter or the ball radius.
opts	The optional inputs. By default, use opts=[]. For advanced usage, please refer to Section 11.

Table 2: Output parameters

Parameter	Description
$\mathbf{x}$	The returned weight vector (or matrix). In most functions, $\mathbf{x}$ is an n-dimensional column vector. In the multi-class classification scenario with $k > 2$ classes, $\mathbf{x}$ is an $n \times k$ matrix.
c	The intercept used in the logistic loss.
funVal	The function values during the iterations.

1.3 Applicability of the SLEP Package

In the current version, we implement the algorithms using two types of loss functions: the least squares loss and the logistic loss, and for trace norm only the least squares loss is implemented except in the matrix classification case in which only the logistic loss is implemented.

We briefly describe the meaning of some abbreviations as follows:

- "Least": *least* squares loss.
- "Logistic" : *logistic* loss.
- "R": the function solves the *regularized* problem.
- "C": the function solves the *constrained* problem.
- "nn": it has an additional *non-negative* constraint.
- "gl": the features of a sample are grouped into several groups, as in *group lasso* [44].
- "mt": the function is for *multi-task* learning, where different tasks may have different samples.
- "mc": the function is for *multi-class* learning or multi-task learning where the different tasks share the same samples.
- "pathSolution": the function computes the *pathwise solutions* corresponding to a series of parameters, using the "warm-start" technique.

Table 3: Applicability of the SLEP package

Penalty	Problem	Function	Description	Section
Lasso	$\min_{\mathbf{x}:\|\mathbf{x}\|_1\leq z} \frac{1}{2}\|\mathbf{x}-\mathbf{v}\|_2^2$	eplb	Euclidean projection onto the ℓ_1 ball	2.1
	$\min_{\mathbf{x}} f(\mathbf{x})+\lambda\|\mathbf{x}\|_1$	LeastR	Least squares loss	2.2
		LogisticR	Logistic loss	2.3
	$\min_{\mathbf{x}:\|\mathbf{x}\|_1\leq z} f(\mathbf{x})$	LeastC	Least squares loss	2.4
		LogisticC	Logistic loss	2.5
	$\min_{\mathbf{x}\geq\mathbf{0}} f(\mathbf{x})+\lambda\|\mathbf{x}\|_1$	nnLeastR	Least squares loss	2.6
		nnLogisticR	Logistic loss	2.7
	$\min_{\mathbf{x}:\|\mathbf{x}\|_1\leq z,\mathbf{x}\geq\mathbf{0}} f(\mathbf{x})$	nnLeastC	Least squares loss	2.8
		nnLogisticC	Logistic loss	2.9
group Lasso[1]	$\min_{\mathbf{x}} \frac{1}{2}\|\mathbf{x}-\mathbf{v}\|_2^2+\lambda\|\mathbf{x}\|_q$	epp	ℓ_q-regularized Euclidean projection	3.1
	$\min_{\mathbf{x}} f(\mathbf{x})+\lambda\|\mathbf{x}\|_{q,1}$	glLeastR	Group Lasso	3.2
		glLogisticR		3.3
		mtLeastR	Multi-task learning	3.4
		mtLogisticR		3.5
		mcLeastR	Multi-class/task learning	3.6
		mcLogisticR		3.7
	$\min_{\mathbf{x}:\|\mathbf{x}\|_{2,1}\leq z} f(\mathbf{x})$	mtLeastC	Multi-task learning	4.1
		mtLogisticC		4.2
		mcLeastC	Multi-class/task learning	4.3
		mcLogisticC		4.4
fused Lasso	$\min_{\mathbf{x}} \frac{1}{2}\|\mathbf{x}-\mathbf{v}\|_2^2+\lambda_1\|\mathbf{x}\|_1+\lambda_2\sum_{i=1}^{p-1}\|x_i-x_{i+1}\|$	flsa	fused Lasso signal approximator	5.1
	$\min_{\mathbf{x}} f(\mathbf{x})+\lambda_1\|\mathbf{x}\|_1+\lambda_2\sum_{i=1}^{p-1}\|x_i-x_{i+1}\|$	fusedLeastR	Least Squares Loss	5.2
		fusedLogisticR	Logistic Loss	5.3
sparse inverse covariance	$\max_{\Theta\succ 0}\log\|\Theta\|-\langle S,\Theta\rangle-\lambda\|\Theta\|_1$	spaInvCov	sparse inverse covariance estimation	6
sparse group Lasso[2]	$\min_{\mathbf{x}} \frac{1}{2}\|\mathbf{x}-\mathbf{v}\|_2^2+\lambda_1\|\mathbf{x}\|_1+\lambda_2\sum_{i=1}^{g} w_i\|\mathbf{x}_{G_i}\|_2$	altra	Moreau-Yosida Regularization	7.1
	$\min_{\mathbf{x}} f(\mathbf{x})+\lambda_1\|\mathbf{x}\|_1+\lambda_2\sum_{i=1}^{g} w_i\|\mathbf{x}_{G_i}\|_2$	sgLeastR	Least Squares Loss	7.2
		sgLogisticR	Logistic Loss	7.3
		mc_sgLeastR	Logistic Loss	7.4
tree structured group Lasso[3]	$\min_{\mathbf{x}} \frac{1}{2}\|\mathbf{x}-\mathbf{v}\|_2^2+\lambda\sum_{i=1}^{g} w_i\|\mathbf{x}_{G_i}\|_2$	altra	Moreau-Yosida Regularization	8.1
		general_altra		
	$\min_{\mathbf{x}} f(\mathbf{x})+\lambda\sum_{i=1}^{g} w_i\|\mathbf{x}_{G_i}\|_2$	tree_LeastR	Least Squares Loss	8.2
		tree_LogisticR	Logistic Loss	8.3
		tree_mcLeastR	Least Squares Loss	8.4
		tree_mcLogisticR	Logistic Loss	8.5
		tree_mcLeastR	Least Squares Loss	8.6
		tree_mcLogisticR	Logistic Loss	8.7
overlapping group Lasso[4]	$\min_{\mathbf{x}} \frac{1}{2}\|\mathbf{x}-\mathbf{v}\|_2^2+\lambda\sum_{i=1}^{g} w_i\|\mathbf{x}_{G_i}\|_2$	overlapping	Moreau-Yosida Regularization	9.1
	$\min_{\mathbf{x}} f(\mathbf{x})+\lambda\sum_{i=1}^{g} w_i\|\mathbf{x}_{G_i}\|_2$	overlapping_LeastR	Least Squares Loss	9.2
		overlapping_LogisticR	Logistic Loss	9.3
Ordered Tree	$\min_{\mathbf{x}\in P} \frac{1}{2}\|\mathbf{x}-\mathbf{v}\|_2^2$	orderTree	Euclidean Projection onto P	10.1
	$\min_{\mathbf{x}\in P} f(\mathbf{x})+\lambda\|\mathbf{x}\|_1$	orderLeastC	Least Squares Loss	10.2
Least Squares Loss		pathSolutionLeast	Pathwise solutions	12
Logistic Loss		pathSolutionLogistic		
trace norm	$\min_W \frac{1}{2}\|\|XW-Y\|\|_F^2+\lambda\|\|W\|\|_*$	accel_grad_mlr	Linear regression	13.1
		mat_primal	Linear regression	13.4
		mat_dual	Linear regression	13.5
	$\min_W \frac{1}{2}\sum_{i=1}^{k}\|\|X_iw_i-Y_i\|\|_2^2+\lambda\|\|W\|\|_*$	accel_grad_mtl	Multi-task learning	13.2
	$\min_W \sum_{i=1}^{n}\ell(y_i,\mathrm{Tr}(W^TX_i))+\lambda\|\|W\|\|_*$	accel_grad_mc	Matrix classification	13.3

[1]: The ℓ_1/ℓ_q-norm is defined as the summation of ℓ_q-norm of the non-overlapping groups.
[2]: In the sparse group Lasso, the indices G_i do not overlap, i.e. $G_i\cap G_j=\emptyset,\forall i\neq j$.
[3]: In the tree structured group Lasso, the indices G_i overlap. However, note that, G_i's follow the tree structure, as depicted in Figure 11.
[4]: In the overlapping group Lasso, the indices G_i may overlap, without the restriction in the tree structured group Lasso.

1.4 Folders & Examples

The functions listed in Table 3 are located at the folder "/SLEP/functions". To use these functions, you can use the command "addpath(genpath([root '/SLEP']));", where "root" is the path to the folder "SLEP". In the folder "SLEP" , the subfolder 'CFiles" contains all the C files for Euclidean projections. Recall that, to call the C files in Matlab, you need to use "mex**C**".

For each function listed in Table 3, we provide an example named "example_functionName" for illustration (such examples can be found at the folder "/SLEP/Examples"). Each example includes:

- Generating a synthetic data matrix A, the response $\mathbf{y}$, and others.
- Setting values for 'opts'.
- Running the function and plotting the objective function values during iterations.
- Computing the pathwise solutions corresponding to a series of parameters.

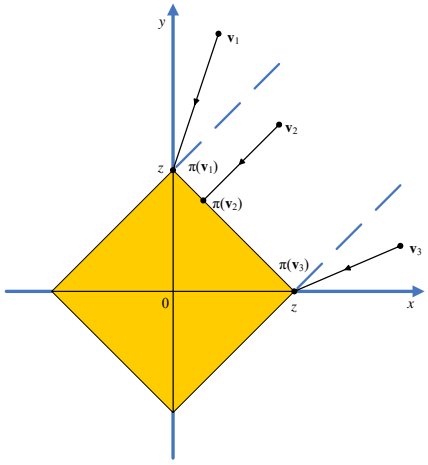


Figure 2: Illustration of the Euclidean projection onto the ℓ_1 ball.

2 ℓ_1-Norm Regularized/Constrained Problems

In this section, we provide the details of the Euclidean projection onto the ℓ_1 ball [40] and the eight functions (located at /SLEP/functions/L1) for sparse learning via the ℓ_1-norm regularization (or the constrained version). The functions described in the following subsections are based on our work in [35, 40].

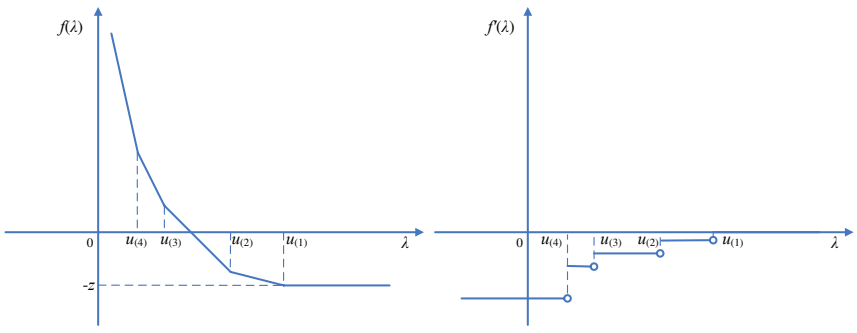
2.1 Euclidean Projection onto the ℓ_1 Ball via Improved Bisection

The problem of Euclidean projections onto the ℓ_1 ball $G = \{\mathbf{x} : \|\mathbf{x}\|_1 \leq z\}$ can be formally defined as:

$$\pi_G(\mathbf{v}) = \arg \min_{\mathbf{x}:\|\mathbf{x}\|_1 \leq z} \frac{1}{2}\|\mathbf{x} - \mathbf{v}\|^2. \tag{1}$$

Such a projection is illustrated in Figure 2. We show in [40] that, this problem can be converted to a zero finding problem for the function $f(\cdot)$ depicted in Figure 3.

We proposed an improved bisection for the efficient computation of (1), and developed the function "eplb" (Euclidean Projection onto the ℓ_1 Ball). The function "eplb" plays a key building block role in the functions: "LeastC", "LogisticC", "nnLeastC", and "nnLogisticC". The function "eplb" is written in the standard C language, and can be found in the head file /SELP/CFiles/q1/epph.h. After running "mex eplb.c", you can call eplb in Matlab as



(a) (b)

Figure 3: Illustration of the auxiliary function $f(\lambda)$ (a) and its subgradient $f'(\lambda)$ (b).

Table 4: Illustration of the improved bisection algorithm.

$\lvert U\rvert$	λ_1	λ_T	λ	λ_S	λ_2
10^5	0	0.79907	2.49512	4.19116	4.19641
1242	2.49512	2.72716	3.24086	3.75455	4.19116
502	2.72716	2.85927	2.93296	3.00665	3.24086
88	2.85927	2.89934	2.90139	2.90343	2.93296
1	2.89934	**2.90105**	2.90105	2.90105	2.90139

$$[\mathbf{x},\, \lambda,\, s]\text{=eplb}(\mathbf{v},\, n,\, z,\, \lambda_0),$$

where n is the size of $\mathbf{v}$, λ_0 is an initial guess of the root λ, s is the number of iterations consumed by the improved bisection algorithm.

Table 4 illustrates the improved bisection algorithm: each row corresponds to an iteration; $[\lambda_1, \lambda_2]$ denotes the current interval, and $|U|$ denotes its size; λ_T is computed from the two models $T_1(.)$ and $T_2(.)$; λ_S is computed from the model $S(.)$; λ is the middle point of λ_T and λ_S; and the found root is in bold.

2.2 LeastR

The function

$$[\mathbf{x},\, \text{funVal}]\text{=LeastR}(A,\, \mathbf{y},\, \lambda,\, \text{opts})$$

solves the ℓ_1-norm (and the squared ℓ_2 norm) regularized least squares problem:

$$\min_{\mathbf{x}} \frac{1}{2}\|A\mathbf{x} - \mathbf{y}\|_2^2 + \frac{\rho}{2}\|\mathbf{x}\|_2^2 + \lambda\|\mathbf{x}\|_1, \tag{2}$$

where $A \in \mathbb{R}^{m\times n}$, $\mathbf{y} \in \mathbb{R}^{m\times 1}$, and $\mathbf{x} \in \mathbb{R}^{n\times 1}$. For illustration of A, $\mathbf{x}$, and $\mathbf{y}$, please refer to Figure 4.

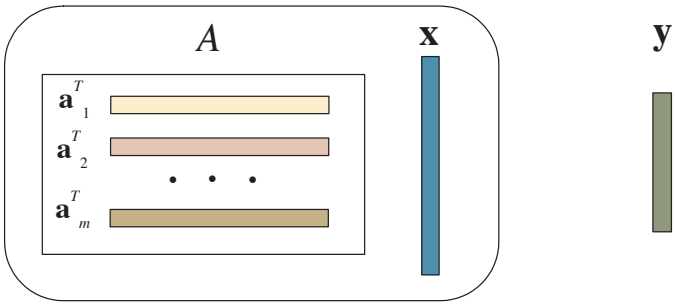


Figure 4: Illustration of the data matrix A, the response $\mathbf{y}$, and the solution $\mathbf{x}$. Such a data structure is employed in the functions "LeastR", "LeastC", "LogisticR", "LogisticC", "nnLeastR", "nnLeastC", "nnLogisticR", and "nnLogisticC". A is a matrix of size $m \times n$ ($\mathbf{a}_i^T$ is the i-th sample and corresponds to the i-th row of A), $\mathbf{x}$ is of size $n \times 1$, and $\mathbf{y}$ is of size $m \times 1$.

In (2), λ is the ℓ_1-norm regularization parameter, and ρ (specified by 'opts.rsL2'; $\rho = 0$ by default) is the regularization parameter for the squared ℓ_2 norm. By setting 'opts.rFlag=0', the program uses the input values for λ and ρ. By setting 'opts.rFlag=1', the program automatically computes $\lambda_{\max}$, the maximal value of λ, above which (2) shall obtain the zero solution. In the latter case, the input λ should be specified as a ratio whose value lies in the interval $[0, 1]$, and the resulting (ℓ_1) regularization used in the program is $\lambda \times \lambda_{\max}$; similarly, the input ρ is a ratio larger than 0, and the actual (ℓ_2) regularization used in the program is $\rho \times \lambda_{\max}$.

If A is a sparse matrix, you can store A in the sparse format in Matlab. The use of the sparse matrix format can significantly improve the efficiency. If you want to normalize the sparse matrix A, you can perform this implicitly through 'opts.nFlag', 'opts.mu', and 'opts.nu'.

If A is a partial DCT matrix, you need to register this variable as a partialDCT class. The use of the partial DCT matrix can significantly improve the efficiency.

By default, you can set opts=[] to use the default settings. For the more advanced usage, you can specify 'opts' (See Section 11). Currently, this function supports the following fields:

- Starting point: .x0, .init
- Termination: .maxIter, .tol, .tFlag
- Normalization: .nFlag, .mu, .nu
- Regularization: .rFlag, .rsL2
- Method: .mFlag, .lFlag
- Group & Others: .fName

2.3 LogisticR

The function

$$[\mathbf{x}, c, \text{funVal}]=\text{LogisticR}(A, \mathbf{y}, \lambda, \text{opts})$$

solves the ℓ_1-norm (and the squared ℓ_2 norm) regularized logistic regression problem:

$$\min_{\mathbf{x}} \sum_{i=1}^{m} w_i \log(1+\exp(-y_i(\mathbf{x}^T\mathbf{a}_i + c))) + \frac{\rho}{2}\|\mathbf{x}\|_2^2 + \lambda\|\mathbf{x}\|_1, \tag{3}$$

where $\mathbf{a}_i^T$ denotes the i-th row of $A \in \mathbb{R}^{m\times n}$, w_i is the weight for the i-th sample, $\mathbf{y} \in \mathbb{R}^{m\times 1}$, and $\mathbf{x} \in \mathbb{R}^{n\times 1}$, and c is the intercept (scalar). For illustration of A, $\mathbf{x}$, and $\mathbf{y}$, please refer to Figure 4.

In (3), λ is the ℓ_1-norm regularization parameter, and ρ (specified by 'opts.rsL2'; $\rho = 0$ by default) is the regularization parameter for the squared ℓ_2 norm. By setting 'opts.rFlag=0', the program uses the input values for λ and ρ. By setting 'opts.rFlag=1', the program automatically computes $\lambda_{\max}$, the maximal value of λ, above which (3) shall obtain the zero solution. In the latter case, the input λ should be specified as a ratio whose value lies in the interval $[0, 1]$, and the resulting (ℓ_1) regularization used in the program is $\lambda \times \lambda_{\max}$; similarly, the input ρ is a ratio larger than 0, and the actual (ℓ_2) regularization used in the program is $\rho \times \lambda_{\max}$.

If A is a sparse matrix, you can store A in the sparse format in Matlab. The use of the sparse matrix format can significantly improve the efficiency. If you want to normalize the sparse matrix A, you can perform this implicitly through 'opts.nFlag', 'opts.mu', and 'opts.nu'.

By default, you can set opts=[] to use the default settings. For the more advanced usage, you can specify 'opts' (See Section 11). Currently, this function supports the following fields:

- Starting point: .x0, .c0, .init
- Termination: .maxIter, .tol, .tFlag
- Normalization: .nFlag, .mu, .nu
- Regularization: .rFlag, .rsL2

- Method: .mFlag, .lFlag
- Group & Others: .sWeight, .fName

In this function, the elements in $\mathbf{y}$ are required to be either 1 or -1, and w_i's (specified by 'opts.sWeight') satisfy: $w_i = w_j$ if $y_i = y_j$. We normalize w_i so that $\sum_i w_i = 1$.

2.4 LeastC

The function

[$\mathbf{x}$, funVal]=LeastC(A, $\mathbf{y}$, z, opts)

solves the ℓ_1-ball constrained least squares problem:

$$\begin{aligned} \min_{\mathbf{x}} \quad & \frac{1}{2}\|A\mathbf{x}-\mathbf{y}\|_2^2 + \frac{\rho}{2}\|\mathbf{x}\|_2^2 \\ \text{subject to} \quad & \|\mathbf{x}\|_1 \le z, \end{aligned} \tag{4}$$

where $A \in \mathbb{R}^{m\times n}$, $\mathbf{y} \in \mathbb{R}^{m\times 1}$, and $\mathbf{x} \in \mathbb{R}^{n\times 1}$. For illustration of A, $\mathbf{x}$, and $\mathbf{y}$, please refer to Figure 4.

In (4), z is the radius of the ℓ_1 ball, and ρ (specified by 'opts.rsL2'; $\rho = 0$ by default) is the regularization parameter for the squared ℓ_2 norm. There is a one-to-one correspondence between the radius z in (4) and the regularization parameter λ in (2), although the explicit formula is usually unknown.

If A is a sparse matrix, you can store A in the sparse format in Matlab. The use of the sparse matrix format can significantly improve the efficiency. If you want to normalize the sparse matrix A, you can perform this implicitly through 'opts.nFlag', 'opts.mu', and 'opts.nu'.

If A is a partial DCT matrix, you need to register this variable as a partialDCT class. The use of the partial DCT matrix can significantly improve the efficiency.

By default, you can set opts=[] to use the default settings. For the more advanced usage, you can specify 'opts' (See Section 11). Currently, this function supports the following fields:

- Starting point: .x0, .init
- Termination: .maxIter, .tol, .tFlag
- Normalization: .nFlag, .mu, .nu
- Regularization: .rsL2
- Method: .mFlag, .lFlag
- Group & Others: .fName

2.5 LogisticC

The function

[$\mathbf{x}$, c, funVal]=LogisticC(A, $\mathbf{y}$, z, opts)

solves the ℓ_1-ball constrained logistic regression problem:

$$\begin{aligned} \min_{\mathbf{x}} \quad & \sum_{i=1}^{m} w_i \log(1+\exp(-y_i(\mathbf{x}^T\mathbf{a}_i + c))) + \frac{\rho}{2}\|\mathbf{x}\|_2^2 \\ \text{subject to} \quad & \|\mathbf{x}\|_1 \le z, \end{aligned} \tag{5}$$

where $\mathbf{a}_i^T$ denotes the i-th row of $A \in \mathbb{R}^{m\times n}$, w_i is the weight for the i-th sample, $\mathbf{y} \in \mathbb{R}^{m\times 1}$, and $\mathbf{x} \in \mathbb{R}^{n\times 1}$, and c is the intercept (scalar). For illustration of A, $\mathbf{x}$, and $\mathbf{y}$, please refer to Figure 4.

In (5), z is the radius of the ℓ_1 ball, and ρ (specified by 'opts.rsL2'; $\rho = 0$ by default) is the regularization parameter for the squared ℓ_2 norm. There is a one-to-one correspondence between the radius z in (5) and the regularization parameter λ in (3), although the explicit formula is usually unknown.

If A is a sparse matrix, you can store A in the sparse format in Matlab. The use of the sparse matrix format can significantly improve the efficiency. If you want to normalize the sparse matrix A, you can perform this implicitly through 'opts.nFlag', 'opts.mu', and 'opts.nu'.

In this function, the elements in $\mathbf{y}$ are required to be either 1 or -1, and w_i's (specified by opts.sWeight) satisfy: $w_i = w_j$ if $y_i = y_j$. In the function, we normalize w_i so that $\sum_i w_i = 1$.

By default, you can set opts=[] to use the default settings. For the more advanced usage, you can specify 'opts' (See Section 11). Currently, this function supports the following fields:

- Starting point: .x0, .c0, .init
- Termination: .maxIter, .tol, .tFlag
- Normalization: .nFlag, .mu, .nu
- Regularization: .rsL2
- Method: .mFlag, .lFlag
- Group & Others: .sWeight, .fName

2.6 nnLeastR

The function

[$\mathbf{x}$, funVal]=nnLeastR(A, $\mathbf{y}$, λ, opts)

solves the ℓ_1-norm regularized least squares problem (subject to an additional non-negative constraint):

$$\min_{\mathbf{x}\geq 0} \frac{1}{2}\|A\mathbf{x} - \mathbf{y}\|_2^2 + \frac{\rho}{2}\|\mathbf{x}\|_2^2 + \lambda\|\mathbf{x}\|_1, \tag{6}$$

where $A \in \mathbb{R}^{m\times n}$, $\mathbf{y} \in \mathbb{R}^{m\times 1}$, and $\mathbf{x} \in \mathbb{R}^{n\times 1}$. For illustration of A, $\mathbf{x}$, and $\mathbf{y}$, please refer to Figure 4.

In (6), λ is the ℓ_1-norm regularization parameter, and ρ (specified by 'opts.rsL2'; $\rho = 0$ by default) is the regularization parameter for the squared ℓ_2 norm. By setting 'opts.rFlag=0', the program uses the input values for λ and ρ. By setting 'opts.rFlag=1', the program automatically computes $\lambda_{\max}$, the maximal value of λ, above which (6) shall obtain the zero solution. In the latter case, the input λ should be specified as a ratio whose value lies in the interval $[0, 1]$, and the resulting (ℓ_1) regularization used in the program is $\lambda \times \lambda_{\max}$; similarly, the input ρ is a ratio larger than 0, and the actual (ℓ_2) regularization used in the program is $\rho \times \lambda_{\max}$.

If A is a sparse matrix, you can store A in the sparse format in Matlab. The use of the sparse matrix format can significantly improve the efficiency. If you want to normalize the sparse matrix A, you can perform this implicitly through 'opts.nFlag', 'opts.mu', and 'opts.nu'.

If A is a partial DCT matrix, you need to register this variable as a partialDCT class. The use of the partial DCT matrix can significantly improve the efficiency.

By default, you can set opts=[] to use the default settings. For the more advanced usage, you can specify 'opts' (See Section 11). Currently, this function supports the following fields:

- Starting point: .x0, .init
- Termination: .maxIter, .tol, .tFlag
- Normalization: .nFlag, .mu, .nu
- Regularization: .rFlag, .rsL2
- Group & Others: .fName

2.7 nnLogisticR

The function

$$[\mathbf{x}, c, \text{funVal}]=\text{nnLogisticR}(A, \mathbf{y}, \lambda, \text{opts})$$

solves the ℓ_1-norm regularized logistic regression problem (subject to an additional non-negative constraint):

$$\min_{\mathbf{x}\geq 0} \sum_{i=1}^{m} w_i \log(1+\exp(-y_i(\mathbf{x}^T\mathbf{a}_i+c))) + \frac{\rho}{2}\|\mathbf{x}\|_2^2 + \lambda\|\mathbf{x}\|_1, \tag{7}$$

where $\mathbf{a}_i^T$ denotes the i-th row of $A \in \mathbb{R}^{m\times n}$, w_i is the weight for the i-th sample, $\mathbf{y} \in \mathbb{R}^{m\times 1}$, and $\mathbf{x} \in \mathbb{R}^{n\times 1}$, and c is the intercept (scalar). For illustration of A, $\mathbf{x}$, and $\mathbf{y}$, please refer to Figure 4.

In (7), λ is the ℓ_1-norm regularization parameter, and ρ (specified by 'opts.rsL2'; $\rho = 0$ by default) is the regularization parameter for the squared ℓ_2 norm. By setting 'opts.rFlag=0', the program uses the input values for λ and ρ. By setting 'opts.rFlag=1', the program automatically computes $\lambda_{\max}$, the maximal value of λ, above which (7) shall obtain the zero solution. In the latter case, the input λ should be specified as a ratio whose value lies in the interval $[0, 1]$, and the resulting (ℓ_1) regularization used in the program is $\lambda \times \lambda_{\max}$; similarly, the input ρ is a ratio larger than 0, and the actual (ℓ_2) regularization used in the program is $\rho \times \lambda_{\max}$.

If A is a sparse matrix, you can store A in the sparse format in Matlab. The use of the sparse matrix format can significantly improve the efficiency. If you want to normalize the sparse matrix A, you can perform this implicitly through 'opts.nFlag', 'opts.mu', and 'opts.nu'.

By default, you can set opts=[] to use the default settings. For the more advanced usage, you can specify 'opts' (See Section 11). Currently, this function supports the following fields:

- Starting point: .x0, .c0, .init
- Termination: .maxIter, .tol, .tFlag
- Normalization: .nFlag, .mu, .nu
- Regularization: .rFlag, .rsL2
- Group & Others: .sWeight, .fName

In this function, the elements in $\mathbf{y}$ are required to be either 1 or -1, and w_i's (specified by opts.sWeight) satisfy: $w_i = w_j$ if $y_i = y_j$. In this function, we normalize w_i so that $\sum_i w_i = 1$.

2.8 nnLeastC

The function

$$[\mathbf{x}, \text{funVal}]=\text{LeastC}(A, \mathbf{y}, z, \text{opts})$$

solves the ℓ_1-ball constrained least squares problem (subject to an additional non-negative constraint):

$$\begin{aligned} \min_{\mathbf{x}} \quad & \frac{1}{2}\|A\mathbf{x}-\mathbf{y}\|_2^2+\frac{\rho}{2}\|\mathbf{x}\|_2^2 \\ \text{subject to} \quad & \|\mathbf{x}\|_1 \le z, \mathbf{x} \ge 0 \end{aligned} \tag{8}$$

where $A \in \mathbb{R}^{m\times n}$, $\mathbf{y} \in \mathbb{R}^{m\times 1}$, and $\mathbf{x} \in \mathbb{R}^{n\times 1}$. For illustration of A, $\mathbf{x}$, and $\mathbf{y}$, please refer to Figure 4.

In (8), z is the radius of the ℓ_1 ball, and ρ (specified by opts.rsL2; $\rho = 0$ by default) is the regularization parameter for the squared ℓ_2 norm.

If A is a sparse matrix, you can store A in the sparse format in Matlab. The use of the sparse matrix format can significantly improve the efficiency. If you want to normalize the sparse matrix A, you can perform this implicitly through 'opts.nFlag', 'opts.mu', and 'opts.nu'.

If A is a partial DCT matrix, you need to register this variable as a partialDCT class. The use of the partial DCT matrix can significantly improve the efficiency.

By default, you can set opts=[] to use the default settings. For the more advanced usage, you can specify 'opts' (See Section 11). Currently, this function supports the following fields:

- Starting point: .x0, .init
- Termination: .maxIter, .tol, .tFlag
- Regularization: .rsL2
- Normalization: .nFlag, .mu, .nu
- Group & Others: .fName

2.9 nnLogisticC

The function

$$[\mathbf{x}, c, \text{funVal}]=\text{LogisticC}(A, \mathbf{y}, z, \text{opts})$$

solves the ℓ_1-ball constrained logistic regression problem (subject to an additional non-negative constraint):

$$\begin{aligned} \min_{\mathbf{x}} \quad & \sum_{i=1}^{m} w_i \log(1+\exp(-y_i(\mathbf{x}^T\mathbf{a}_i+c)))+\frac{\rho}{2}\|\mathbf{x}\|_2^2 \\ \text{subject to} \quad & \|\mathbf{x}\|_1 \le z, \mathbf{x} \ge 0 \end{aligned} \tag{9}$$

where $\mathbf{a}_i^T$ denotes the i-th row of $A \in \mathbb{R}^{m\times n}$, w_i is the weight for the i-th sample, $\mathbf{y} \in \mathbb{R}^{m\times 1}$, and $\mathbf{x} \in \mathbb{R}^{n\times 1}$, and c is the intercept (scalar). For illustration of A, $\mathbf{x}$, and $\mathbf{y}$, please refer to Figure 4.

In (9), z is the radius of the ℓ_1 ball, and ρ (specified by opts.rsL2; $\rho = 0$ by default) is the regularization parameter for the squared ℓ_2 norm.

If A is a sparse matrix, you can store A in the sparse format in Matlab. The use of the sparse matrix format can significantly improve the efficiency. If you want to normalize the sparse matrix A, you can perform this implicitly through 'opts.nFlag', 'opts.mu', and 'opts.nu'.

By default, you can set opts=[] to use the default settings. For the more advanced usage, you can specify 'opts' (See Section 11). Currently, this function supports the following fields:

- Starting point: .x0, .c0, .init
- Termination: .maxIter, .tol, .tFlag
- Normalization: .nFlag, .mu, .nu
- Regularization: .rsL2
- Group & Others: .sWeight, .fName

In this function, the elements in $\mathbf{y}$ are required to be either 1 or -1, and w_i's (specified by opts.sWeight) satisfy: $w_i = w_j$ if $y_i = y_j$. In the function, we normalize w_i so that $\sum_i w_i = 1$.

3 ℓ_1/ℓ_q-Norm Regularized Problems

In this section, we provide the details of the ℓ_1/ℓ_q-regularized projection and the six functions (located at /SLEP/functions/L1Lq/Lq1R) for sparse learning via ℓ_1/ℓ_q-norm ($q \geq 1$). The functions described in the following subsections are based on our work in [36, 39].

3.1 The ℓ_q-Regularized Projection Via Bisection

In solving the ℓ_1/ℓ_q-norm regularized problem, one key subroutine is the ℓ_1/ℓ_q-regularized Euclidean projection (EP_{1q}) problem:

$$\pi_{1q}(\mathbf{V}, \lambda) = \arg\min_{\mathbf{X}\in\mathbb{R}^p} \frac{1}{2}\|\mathbf{X} - \mathbf{V}\|_2^2 + \lambda \sum_{i=1}^{s} \|\mathbf{x}_i\|_q, \tag{10}$$

which can be decoupled into the following ℓ_q-regularized projection problem:

$$\pi_q(\mathbf{v}, \lambda) = \min_{\mathbf{x}} \frac{1}{2}\|\mathbf{x} - \mathbf{v}\|_2^2 + \lambda\|\mathbf{x}\|_q. \tag{11}$$

In [39], we showed the key properties of the problem (11), proposed to formulate 11 as two simple zero finding problems, and developed the function "epp" for solving (11). After running "mex epp.c" (located at /SLEP/CFiles/epp), we can call "epp" in matlab as

[$\mathbf{x}$, c, s]=epp($\mathbf{v}$, n, λ, q c_0),

where n is the size of $\mathbf{v}$, $q \geq 1$, c_0 is an initial guess of the root c, s is the number of iterations consumed by the bisection algorithm.

3.2 glLeastR

The function

[$\mathbf{x}$, funVal]=glLeastR(A, $\mathbf{y}$, λ, opts)

solves the ℓ_1/ℓ_q-norm regularized least squares problem:

$$\min_{\mathbf{x}} \frac{1}{2}\|A\mathbf{x} - \mathbf{y}\|_2^2 + \lambda \sum_{i=1}^{k} w_i^g \|\mathbf{x}_{G_i}\|_q, \tag{12}$$

where $A \in \mathbb{R}^{m\times n}$, $\mathbf{y} \in \mathbb{R}^{m\times 1}$, $\mathbf{x} \in \mathbb{R}^{n\times 1}$ is divided into k non-overlapping groups $\mathbf{x}_{G_1}, \mathbf{x}_{G_2}, \ldots, \mathbf{x}_{G_k}$, and w_i^g denotes the weight for the i-th group. The group information is provided in 'opts.ind'. For illustration of A and $\mathbf{x}$, please refer to Figure 5.

In (12), λ is the ℓ_1/ℓ_q-norm regularization parameter. By setting 'opts.rFlag=0', the program uses the input value for λ. By setting 'opts.rFlag=1', the program automatically computes $\lambda_{\max}$, the maximal value of λ, above which (12) shall obtain the zero solution. In the latter case, the input λ should be specified as a ratio whose value lies in the interval $[0, 1]$, and the resulting regularization used in the program is $\lambda \times \lambda_{\max}$.

The parameter w_i^g denotes the weight for the i-th group; you can specify w_i^g via 'opts.gWeight'. The program requires $w_i^g > 0$. Please note that 'opts.gWeight' should be specified as a $k \times 1$ column vector (See Section 11).

If A is a sparse matrix, you can store A in the sparse format in Matlab. The use of the sparse matrix format can significantly improve the efficiency. If you want to normalize the sparse matrix A, you can perform this implicitly through 'opts.nFlag', 'opts.mu', and 'opts.nu'.

By default, you can set empty the fields of 'opts' except 'opts.ind'. For the more advanced usage, you can specify 'opts' (See Section 11). Currently, this function supports the following fields:

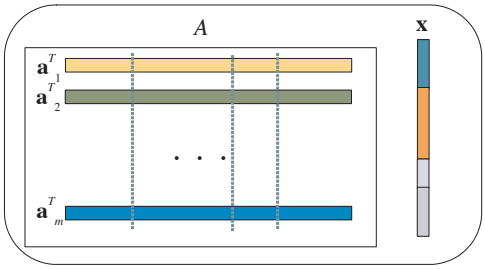


Figure 5: Illustration of the data matrix A, and the solution $\mathbf{x}$ for the functions "glLeastR" and "glLogisticR". A is a matrix of size $m \times n$, $\mathbf{x}$ is of size $n \times 1$ and is divided into k non-overlapping groups $\mathbf{x}_{G_1}, \mathbf{x}_{G_2}, \ldots, \mathbf{x}_{G_k}$. In this example, the features are grouped into four non-overlapping groups (by the dashed lines).

- Starting point: .x0, .init
- Termination: .maxIter, .tol, .tFlag
- Normalization: .nFlag, .mu, .nu
- Regularization: .rFlag
- Method: .mFlag, .lFlag
- Group & Others: .ind, .q, .gWeight, .fName

3.3 glLogisticR

The function

[$\mathbf{x}$, c, funVal]=glLogisticR(A, $\mathbf{y}$, λ, opts)

solves the ℓ_1/ℓ_q-norm regularized logistic regression problem:

$$\min_{\mathbf{x}} \sum_{i=1}^{m} w_i \log(1 + \exp(-y_i(\mathbf{x}^T \mathbf{a}_i + c))) + \lambda \sum_{i=1}^{k} w_i^g \|\mathbf{x}_{G_i}\|_q, \tag{13}$$

where $\mathbf{a}_i^T$ denotes the i-th row of $A \in \mathbb{R}^{m \times n}$, w_i is the weight for the i-th sample, $\mathbf{y} \in \mathbb{R}^{m \times 1}$, $\mathbf{x} \in \mathbb{R}^{n \times 1}$ is divided into k non-overlapping groups $\mathbf{x}_{G_1}, \mathbf{x}_{G_2}, \ldots, \mathbf{x}_{G_k}$, w_i^g denotes the weight for the i-th group, and c is the intercept (scalar). The group information is provided in 'opts.ind'. For illustration of A and $\mathbf{x}$, please refer to Figure 5.

In (13), λ is the ℓ_1/ℓ_q-norm regularization parameter. By setting 'opts.rFlag=0', the program uses the input value for λ. By setting 'opts.rFlag=1', the program automatically computes $\lambda_{\max}$, the maximal value of λ, above which (13) shall obtain the zero solution. In the latter case, the input λ should be specified as a ratio whose value lies in the interval $[0, 1]$, and the resulting regularization used in the program is $\lambda \times \lambda_{\max}$.

The parameter w_i^g denotes the weight for the i-th group; you can specify w_i^g via 'opts.gWeight'. The program requires $w_i^g > 0$. The program requires $w_i^g > 0$. Please note that 'opts.gWeight' should be specified as a $k \times 1$ column vector (See Section 11).

If A is a sparse matrix, you can store A in the sparse format in Matlab. The use of the sparse matrix format can significantly improve the efficiency. If you want to normalize the sparse matrix A, you can perform this implicitly through 'opts.nFlag', 'opts.mu', and 'opts.nu'.

In this function, the elements in $\mathbf{y}$ are required to be either 1 or -1, and w_i's (specified by 'opts.sWeight') satisfy: $w_i = w_j$ if $y_i = y_j$. We also normalize w_i's so that $\sum_i w_i = 1$ holds.

By default, you can set empty the fields of 'opts' except 'opts.ind'. For the more advanced usage, you can specify 'opts' (See Section 11). Currently, you can specify the following fields:

- Starting point: .x0, .c0, .init
- Termination: .maxIter, .tol, .tFlag
- Normalization: .nFlag, .mu, .nu
- Regularization: .rFlag
- Method: .mFlag, .lFlag
- Group & Others: .ind, .q, .sWeight, .fName

3.4 mtLeastR

The function

$$[\mathbf{x}, \text{funVal}]=\text{mtLeastR}(A, \mathbf{y}, \lambda, \text{opts})$$

solves the ℓ_1/ℓ_q-norm regularized multi-task least squares problem:

$$\min_{\mathbf{x}} \frac{1}{2}\|A\mathbf{x} - \mathbf{y}\|_2^2 + \lambda\|\mathbf{x}\|_{\ell_1/\ell_q}, \tag{14}$$

where $A = [A_1; A_2; \ldots; A_k] \in \mathbb{R}^{m\times n}$, $A_i \in \mathbb{R}^{m_i\times n}$ is the data matrix for the i-th task ($m = \sum_i m_i$), $\mathbf{y} = [\mathbf{y}_1; \mathbf{y}_2; \ldots; \mathbf{y}_k] \in \mathbb{R}^{m\times 1}$, $\mathbf{y}_i \in \mathbb{R}^{m_i}$ is the response for the i-th task, $\mathbf{x} = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_k] \in \mathbb{R}^{n\times k}$, and $\mathbf{x}_i \in \mathbb{R}^n$ is the weight vector for the i-th task. The group information is provided in 'opts.ind'. For illustration of A, $\mathbf{y}$, and $\mathbf{x}$, please refer to Figure 6.

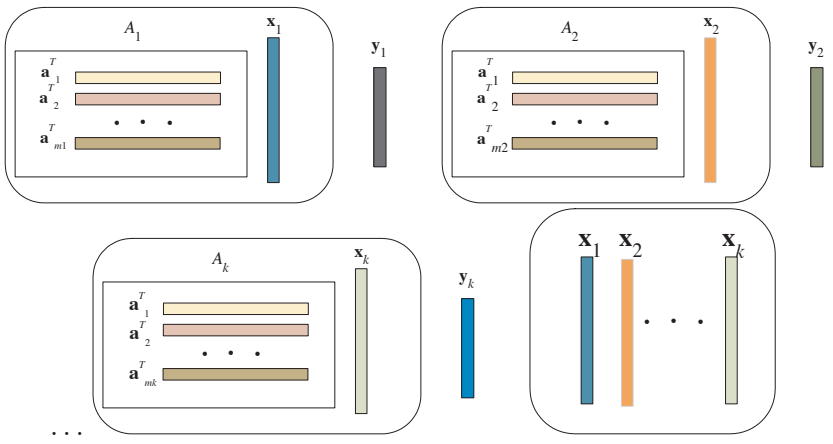


Figure 6: Illustration of the data matrix A, $\mathbf{y}$, and the solution $\mathbf{x}$ for the functions "mtLeastR" and "mtLogisticR" for multi-task learning. $A = [A_1; A_2; \ldots; A_k]$ is a matrix with size $m \times n$ ($m = \sum_i m_i$), $\mathbf{y} = [\mathbf{y}_1; \mathbf{y}_2; \ldots; \mathbf{y}_k]$, and $\mathbf{x} = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_k]$ is of size $n \times k$. Each row of $\mathbf{x}$ forms a group (via the ℓ_q norm). For A and $\mathbf{y}$, (ind(i)+1): ind(i+1) are the indices for the i-th task, namely, $A_i = A($ (ind(i)+1): ind(i+1),:) and $\mathbf{y}_i = \mathbf{y}$ ((ind(i)+1): ind(i+1), :).

In (14), λ is the ℓ_1/ℓ_q-norm regularization parameter. By setting 'opts.rFlag=0', the program uses the input value for λ. By setting 'opts.rFlag=1', the program automatically computes $\lambda_{\max}$, the maximal value of λ, above which (14) shall obtain the zero solution. In the latter case, the input λ should be specified as a ratio whose value lies in the interval $[0, 1]$, and the resulting regularization used in the program is $\lambda \times \lambda_{\max}$.

If A is a sparse matrix, you can store A in the sparse format in Matlab. The use of the sparse matrix format can significantly improve the efficiency. If you want to normalize the sparse matrix A, you can perform this implicitly through 'opts.nFlag', 'opts.mu', and 'opts.nu'.

By default, you can set empty the fields of 'opts' except 'opts.ind'. For the more advanced usage, you can specify 'opts' (See Section 11). Currently, you can specify the following fields:

- Starting point: .x0, .init
- Termination: .maxIter, .tol, .tFlag
- Normalization: .nFlag, .mu, .nu
- Regularization: .rFlag
- Method: .mFlag, .lFlag
- Group & Others: .ind, .q, .fName

3.5 mtLogisticR

The function

$$[\mathbf{x}, \mathbf{c}, \text{funVal}]=\text{mtLogisticR}(A, \mathbf{y}, \lambda, \text{opts})$$

solves the ℓ_1/ℓ_q-norm regularized multi-task logistic regression problem:

$$\min_{\mathbf{x}} \sum_{l=1}^{k} \sum_{i=1}^{m_l} w_{il} \log(1 + \exp(-y_{il}(\mathbf{x}_l^T \mathbf{a}_{il} + c_l))) + \lambda \|\mathbf{x}\|_{\ell_1/\ell_q}, \tag{15}$$

where $\mathbf{a}_{il}^T$ denotes the i-th sample for the l-th task, w_{il} is the weight for $\mathbf{a}_{il}^T$, y_{il} is the response of $\mathbf{a}_{il}$, and c_l is the intercept (scalar) for the l-th task. We have $A \in \mathbb{R}^{m \times n}$, $\mathbf{x} \in \mathbb{R}^{n \times k}$, $\mathbf{c} \in \mathbb{R}^{1 \times k}$, $\mathbf{y} \in \mathbb{R}^{m \times 1}$. The parameter 'opts.ind' provides the indices for the tasks. For illustration of A, $\mathbf{y}$, and $\mathbf{x}$, please refer to Figure 6.

In (15), λ is the ℓ_1/ℓ_q-norm regularization parameter. By setting 'opts.rFlag=0', the program uses the input value for λ. By setting 'opts.rFlag=1', the program automatically computes $\lambda_{\max}$, the maximal value of λ, above which (15) shall obtain the zero solution. In the latter case, the input λ should be specified as a ratio whose value lies in the interval $[0, 1]$, and the resulting regularization used in the program is $\lambda \times \lambda_{\max}$.

If A is a sparse matrix, you can store A in the sparse format in Matlab. The use of the sparse matrix format can significantly improve the efficiency. If you want to normalize the sparse matrix A, you can perform this implicitly through 'opts.nFlag', 'opts.mu', and 'opts.nu'.

By default, you can set the fields of 'opts' to empty except 'opts.ind'. For the more advanced usage, you can specify 'opts' (See Section 11). Currently, you can specify the following fields:

- Starting point: .x0, .c0, .init
- Termination: .maxIter, .tol, .tFlag
- Normalization: .nFlag, .mu, .nu

- Regularization: .rFlag
- Method: .mFlag, .lFlag
- Group & Others: .ind, .q, .fName

In this function, the elements in $\mathbf{y}$ are required to be either 1 or -1. Currently, we simply set $w_{il} = \frac{1}{m}$.

3.6 mcLeastR

The function

[$\mathbf{x}$, funVal]=mcLeastR(A, $\mathbf{y}$, λ, opts)

solves the ℓ_1/ℓ_q-norm regularized multi-class least squares problem:

$$\min_{\mathbf{x}} \frac{1}{2}\|A\mathbf{x} - \mathbf{y}\|_2^2 + \lambda\|\mathbf{x}\|_{\ell_1/\ell_q}, \tag{16}$$

where $A \in \mathbb{R}^{m\times n}$, $\mathbf{y} \in \mathbb{R}^{m\times k}$, and $\mathbf{x} \in \mathbb{R}^{n\times k}$. For illustration of A, $\mathbf{y}$, and $\mathbf{x}$, please refer to Figure 7.

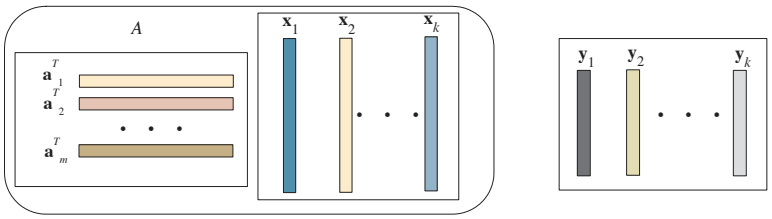


Figure 7: Illustration of the data matrix A, $\mathbf{y}$, and the solution $\mathbf{x}$ for the functions "mcLeastR" and "mcLogisticR" for multi-class classification. A is of size $m \times n$, $\mathbf{y} = [\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_k]$ is of size $m \times n$, and $\mathbf{x} = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_k]$ is of size $n \times k$. Each row of $\mathbf{x}$ forms a group (via the ℓ_q norm).

In (16), λ is the ℓ_1/ℓ_q-norm regularization parameter. By setting 'opts.rFlag=0', the program uses the input value for λ. By setting 'opts.rFlag=1', the program automatically computes $\lambda_{\max}$, the maximal value of λ, above which (16) shall obtain the zero solution. In the latter case, the input λ should be specified as a ratio whose value lies in the interval $[0, 1]$, and the resulting regularization used in the program is $\lambda \times \lambda_{\max}$.

If A is a sparse matrix, you can store A in the sparse format in Matlab. The use of the sparse matrix format can significantly improve the efficiency. If you want to normalize the sparse matrix A, you can perform this implicitly through 'opts.nFlag', 'opts.mu', and 'opts.nu'.

By default, you can set 'opts=[]' to use the default settings. For the more advanced usage, you can specify 'opts' (See Section 11). Currently, this function supports the following fields:

- Starting point: .x0, .init
- Termination: .maxIter, .tol, .tFlag
- Normalization: .nFlag, .mu, .nu
- Regularization: .rFlag
- Method: .mFlag, .lFlag
- Group & Others: .q, .fName

3.7 mcLogisticR

The function

$$[\mathbf{x}, \mathbf{c}, \text{funVal}]=\text{mcLogisticR}(A, \mathbf{y}, \lambda, \text{opts})$$

solves the ℓ_1/ℓ_q-norm regularized multi-class logistic regression problem:

$$\min_{\mathbf{x}} \sum_{l=1}^{k} \sum_{i=1}^{m} w_{il} \log(1 + \exp(-y_{il}(\mathbf{x}_l^T \mathbf{a}_{il} + c_l))) + \lambda \|\mathbf{x}\|_{\ell_1/\ell_q}, \tag{17}$$

where $\mathbf{a}_{il}^T$ denotes the i-th sample for the l-th class, w_{il} is the weight for $\mathbf{a}_{il}^T$, y_{il} is the response of $\mathbf{a}_{il}$, and c_l is the intercept (scalar) for the l-th class. In multi-class classification, we have $\mathbf{a}_{il} = \mathbf{a}_i, \forall l$, $A \in \mathbb{R}^{m \times n}$, $\mathbf{x} \in \mathbb{R}^{n \times k}$, $\mathbf{c} \in \mathbb{R}^{1 \times k}$, $\mathbf{y} \in \mathbb{R}^{m \times k}$. For illustration of A, $\mathbf{y}$, and $\mathbf{x}$, please refer to Figure 7.

In (17), λ is the ℓ_1/ℓ_q-norm regularization parameter. By setting 'opts.rFlag=0', the program uses the input value for λ. By setting 'opts.rFlag=1', the program automatically computes $\lambda_{\max}$, the maximal value of λ, above which (17) shall obtain the zero solution. In the latter case, the input λ should be specified as a ratio whose value lies in the interval $[0, 1]$, and the resulting regularization used in the program is $\lambda \times \lambda_{\max}$.

If A is a sparse matrix, you can store A in the sparse format in Matlab. The use of the sparse matrix format can significantly improve the efficiency. If you want to normalize the sparse matrix A, you can perform this implicitly through 'opts.nFlag', 'opts.mu', and 'opts.nu'.

By default, you can set 'opts=[]' to use the default settings. For the more advanced usage, you can specify 'opts' (See Section 11). Currently, you can specify the following fields:

- Starting point: .x0, .c0, .init
- Termination: .maxIter, .tol, .tFlag
- Normalization: .nFlag, .mu, .nu
- Regularization: .rFlag
- Method: .mFlag, .lFlag
- Group & Others: .q, .fName

In this function, the elements in $\mathbf{y}$ are required to be either 1 or -1. Currently, we set $w_{il} = \frac{1}{mk}$.

4 ℓ_1/ℓ_2-Norm Constrained Problems

In this section, we provide the details of the four functions (located at /SLEP/functions/L1Lq/L21C) for sparse learning via the ℓ_1/ℓ_2-norm constraint. The functions described in the following subsections are based on our work in [36].

4.1 mtLeastC

The function

[$\mathbf{x}$, funVal]=mtLeastC(A, $\mathbf{y}$, z, opts)

solves the ℓ_1/ℓ_2-ball constrained least squares problem:

$$\begin{aligned} \min_{\mathbf{x}} \quad & \frac{1}{2}\|A\mathbf{x}-\mathbf{y}\|_2^2 \\ \text{s.t.} \quad & \|\mathbf{x}\|_{\ell_1/\ell_2} \leq z, \end{aligned} \tag{18}$$

where $A \in \mathbb{R}^{m\times n}$, $\mathbf{y} \in \mathbb{R}^{m\times 1}$, and $\mathbf{x} \in \mathbb{R}^{n\times k}$. The group information is provided in 'opts.ind'. For illustration of A, $\mathbf{y}$, and $\mathbf{x}$, please refer to Figure 6.

If A is a sparse matrix, you can store A in the sparse format in Matlab. The use of the sparse matrix format can significantly improve the efficiency. If you want to normalize the sparse matrix A, you can perform this implicitly through 'opts.nFlag', 'opts.mu', and 'opts.nu'.

By default, you can set the fields of 'opts' to empty except 'opts.ind'. For the more advanced usage, you can specify 'opts' (See Section 11). Currently, you can specify the following fields:

- Starting point: .x0, .init
- Termination: .maxIter, .tol, .tFlag
- Normalization: .nFlag, .mu, .nu
- Regularization: .rFlag
- Method: .lFlag
- Group & Others: .ind, .fName

4.2 mtLogisticC

The function

[$\mathbf{x}$, $\mathbf{c}$, funVal]=mtLogisticC(A, $\mathbf{y}$, z, opts)

solves the ℓ_1/ℓ_2-ball constrained multi-task logistic regression problem:

$$\begin{aligned} \min_{\mathbf{x}} \quad & \sum_{l=1}^{k}\sum_{i=1}^{m_l} w_{il}\log(1+\exp(-y_{il}(\mathbf{x}_l^T\mathbf{a}_{il}+c_l))) \\ \text{s.t.} \quad & \|\mathbf{x}\|_{\ell_1/\ell_2} \leq z, \end{aligned} \tag{19}$$

where $\mathbf{a}_{il}^T$ denotes the i-th sample for the l-th task, w_{il} is the weight for $\mathbf{a}_{il}^T$, y_{il} is the response of $\mathbf{a}_{il}$, and c_l is the intercept (scalar) for the l-th task. We have $A \in \mathbb{R}^{m\times n}$, $\mathbf{x} \in \mathbb{R}^{n\times k}$, $\mathbf{c} \in \mathbb{R}^{1\times k}$, $\mathbf{y} \in \mathbb{R}^{m\times 1}$.

The parameter 'opts.ind' provides the indices for the tasks. For illustration of A, $\mathbf{y}$, and $\mathbf{x}$, please refer to Figure 6.

If A is a sparse matrix, you can store A in the sparse format in Matlab. The use of the sparse matrix format can significantly improve the efficiency. If you want to normalize the sparse matrix A, you can perform this implicitly through 'opts.nFlag', 'opts.mu', and 'opts.nu'.

By default, you can set the fields of 'opts' to empty except 'opts.ind'. For the more advanced usage, you can specify 'opts' (See Section 11). Currently, you can specify the following fields:

- Starting point: .x0, .c0, .init
- Termination: .maxIter, .tol, .tFlag
- Normalization: .nFlag, .mu, .nu
- Regularization: .rFlag
- Method: .lFlag
- Group & Others: .ind, .fName

In this function, the elements in $\mathbf{y}$ are required to be either 1 or -1. Currently, we simply set $w_{il} = \frac{1}{m}$.

4.3 mcLeastC

The function

[$\mathbf{x}$, funVal]=mcLeastC(A, $\mathbf{y}$, z, opts)

solves the ℓ_1/ℓ_2-ball constrained multi-class least squares problem:

$$\begin{array}{ll} \min\limits_{\mathbf{x}} & \frac{1}{2}\|A\mathbf{x} - \mathbf{y}\|_2^2 \\ \text{s.t.} & \|\mathbf{x}\|_{\ell_1/\ell_2} \leq z, \end{array} \tag{20}$$

where $A \in \mathbb{R}^{m\times n}$, $\mathbf{y} \in \mathbb{R}^{m\times k}$, and $\mathbf{x} \in \mathbb{R}^{n\times k}$. For illustration of A, $\mathbf{y}$, and $\mathbf{x}$, please refer to Figure 7.

If A is a sparse matrix, you can store A in the sparse format in Matlab. The use of the sparse matrix format can significantly improve the efficiency. If you want to normalize the sparse matrix A, you can perform this implicitly through 'opts.nFlag', 'opts.mu', and 'opts.nu'.

By default, you can set 'opts=[]' to use the default settings. For the more advanced usage, you can specify 'opts' (See Section 11). Currently, this function supports the following fields:

- Starting point: .x0, .init
- Termination: .maxIter, .tol, .tFlag
- Normalization: .nFlag, .mu, .nu
- Regularization: .rFlag
- Method: .lFlag
- Group & Others: .fName

4.4 mcLogisticC

The function

$$[\mathbf{x}, \mathbf{c}, \text{funVal}]=\text{mcLogisticC}(A, \mathbf{y}, z, \text{opts})$$

solves the ℓ_1/ℓ_2-ball constrained multi-class logistic regression problem:

$$\begin{aligned} \min_{\mathbf{x}} \quad & \sum_{l=1}^{k}\sum_{i=1}^{m} w_{il}\log(1+\exp(-y_{il}(\mathbf{x}_l^T\mathbf{a}_{il}+c_l))) \\ \text{s.t.} \quad & \|\mathbf{x}\|_{\ell_1/\ell_2} \le z, \end{aligned} \tag{21}$$

where $\mathbf{a}_{il}^T$ denotes the i-th sample for the l-th class, w_{il} is the weight for $\mathbf{a}_{il}^T$, y_{il} is the response of $\mathbf{a}_{il}$, and c_l is the intercept (scalar) for the l-th class. In multi-class classification, we have $\mathbf{a}_{il} = \mathbf{a}_i, \forall l$, $A \in \mathbb{R}^{m\times n}$, $\mathbf{x} \in \mathbb{R}^{n\times k}$, $\mathbf{c} \in \mathbb{R}^{1\times k}$, $\mathbf{y} \in \mathbb{R}^{m\times k}$. For illustration of A, $\mathbf{y}$, and $\mathbf{x}$, please refer to Figure 7.

If A is a sparse matrix, you can store A in the sparse format in Matlab. The use of the sparse matrix format can significantly improve the efficiency. If you want to normalize the sparse matrix A, you can perform this implicitly through 'opts.nFlag', 'opts.mu', and 'opts.nu'.

By default, you can set 'opts=[]' to use the default settings. For the more advanced usage, you can specify 'opts' (See Section 11). Currently, you can specify the following fields:

- Starting point: .x0, .init
- Termination: .maxIter, .tol, .tFlag
- Normalization: .nFlag, .mu, .nu
- Regularization: .rFlag
- Method: .lFlag
- Group & Others: .fName

In this function, the elements in $\mathbf{y}$ are required to be either 1 or -1. Currently, we set $w_{il} = \frac{1}{mk}$.

Table 5: A running example for solving FLSA via the SFA_R proposed in [43].

	variable	1	2	3	4	5	6	7	8	9	10	duality gap
input	$\mathbf{v}$	0.50	0.60	1.30	-2.10	-2.60	-2.50	-0.10	-0.30	-0.20	2.60	
	$\mathbf{z}_0$	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00		2.06
iteration 1	$\mathbf{z}$	0.02	0.18	-0.20	-0.13	0.03	0.20	-0.05	0.02	0.20		0.16
	$S(\mathbf{z}) - \{0, 10\}$	3	6	9								
	$\mathbf{x}_1$	0.73	0.73	0.73	-2.27	-2.27	-2.27	-0.20	-0.20	-0.20	2.40	
	$\mathbf{z}_1$	0.20	0.20	-0.20	-0.20	-0.03	0.20	0.10	0.20	0.20		0.08
iteration 2	$\mathbf{z}$	0.17	0.20	-0.20	-0.20	0.01	0.20	0.10	0.20	0.20		0.03
	$S(\mathbf{z}) - \{0, 10\}$	2	3	4	6	8	9					
	$\mathbf{x}_2$	0.65	0.65	0.90	-2.10	-2.35	-2.35	-0.20	-0.20	-0.20	2.40	
	$\mathbf{z}_2$	0.15	0.20	-0.20	-0.20	0.05	0.20	0.10	0.20	0.20		0
solution	$S(\mathbf{z}^*) - \{0, 10\}$	2	3	4	6	8	9					
	$\mathbf{x}^*$	0.65	0.65	0.90	-2.10	-2.35	-2.35	-0.20	-0.20	-0.20	2.40	
	$\mathbf{z}^*$	0.15	0.20	-0.20	-0.20	0.05	0.20	0.10	0.20	0.20		0

5 The Fused Lasso Penalized Problems

In this section, we provide the details of the fused Lasso signal approximator and the two functions (located at /SLEP/functions/fusedLasso) for sparse learning via the fused Lasso penalty. The functions described in the following subsections are based on our work in [43].

5.1 Fused Lasso Signal Approximator

The function

$$[\mathbf{x}, \mathbf{z}, \text{infor}]=\text{flsa}(\mathbf{v}, \mathbf{z}_0, \lambda_1, \lambda_2, p, 1000, 10^{-10}, 1, 6)$$

solves the Fused Lasso Signal Approximator (flsa):

$$\min_{\mathbf{x}} \quad \frac{1}{2}\|\mathbf{x} - \mathbf{v}\|_2^2 + \lambda_1\|\mathbf{x}\|_1 + \lambda_2 \sum_{i=2}^{p} |x_i - x_{i-1}|. \tag{22}$$

The problem (22) is a special case of (23), and it plays a building block role in solving (23) by the methods such as the Nesterov's method. In the function flsa, $\mathbf{z}_0$ is an initial guess of the subgradient of the fused penalty at the minimizer, and $\mathbf{z}$ is the obtained subgradient; 1000 is the maximal number of iterations (we have observed in our experiments that, the program converges in dozens of iterations with the "cold" start, e.g., $\mathbf{z}_0 = \mathbf{0}$, and within ten iterations with the "warm" start technique); and the algorithm terminates when the duality gap of the solution is less than 10^{-10}. The output parameter "infor" contains the information for solving (22) including the number of the iterations.

Table 5 gives a running example for solving (22). The input $\mathbf{v}$ has $n = 10$ variables, the regularization parameter $\lambda_2 = 0.2$, and the starting point $\mathbf{z}_0$ is initialized with zero. SFA_R converges in 2 iterations. In each iteration i, 1) $\mathbf{z}$ corresponds to a gradient descent step based on $\mathbf{z}_{i-1}$, 2) $S(\mathbf{z})$ is the support set computed at $\mathbf{z}$, 3) $\mathbf{x}_i = \omega(\mathbf{z})$ is the approximate primal variable computed using the support set, and 4) $\mathbf{z}_i$ is the restarting point obtained from $\mathbf{x}_i$. The duality gap is shown in the last column for each dual variable. $\mathbf{x}^*$ and $\mathbf{z}^*$ denote the optimal primal and dual solutions, respectively.

The function "flsa" is written in the standard C, and the source code is available at /SLEP/CFiles/flsa.h. The function "flsa" plays a building block role for the functions "fusedLeastR" and "fusedLogisticR".

5.2 fusedLeastR

The function

$$[\mathbf{x}, \text{funVal}]=\text{fusedLeastR}(A, \mathbf{y}, \lambda_1, \text{opts})$$

solves the fused Lasso problem:

$$\min_{\mathbf{x}} \quad \frac{1}{2}\|A\mathbf{x}-\mathbf{y}\|_2^2+\lambda_1\|\mathbf{x}\|_1+\lambda_2\sum_{i=1}^{p-1}|x_i-x_{i+1}|, \tag{23}$$

where λ_2 =opts.fusedPenalty.

If A is a sparse matrix, you can store A in the sparse format in Matlab, which can significantly improve the efficiency. If you want to normalize the sparse matrix A, you can perform this implicitly through 'opts.nFlag', 'opts.mu', and 'opts.nu'.

By default, you can set 'opts=[]' to use the default settings. For the more advanced usage, you can specify 'opts' (See Section 11). Currently, you can specify the following fields:

- Starting point: .x0, .init
- Termination: .maxIter, .tol, .tFlag
- Normalization: .nFlag, .mu, .nu
- Regularization: .rFlag
- Method: .lFlag

A note on "opts.rFlag": When opts.rFlag=0, the actual values of λ_1 and λ_2 are used. When opts.rFlag=1, we first compute $\lambda_1^{\max}$ for λ_1, which shall enforce (23) to yield a zero solution, and then the program will use the following regularization values: $\lambda_1 \times \lambda_1^{\max}$ and $\lambda_2 \times \lambda_1^{\max}$.

With properly chosen λ_1 and λ_2, the function fusedLassoLeastR yields an output $\mathbf{x}$ that has sparsity in both the coefficients and their successive differences, as illustrated in Figure 8.

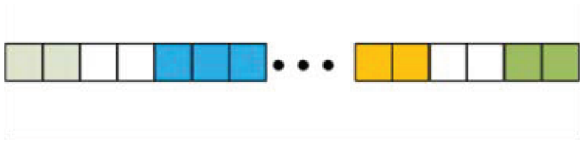


Figure 8: Illustration of the solution of the fused Lasso. The solution is blockwise constant.

5.3 fusedLogisticR

The function

[$\mathbf{x}$, c, funVal]=fusedLogisticR(A, $\mathbf{y}$, λ, opts)

solves the fused Lasso penalized logistic regression problem:

$$\min_{\mathbf{x}} \sum_{i=1}^{m} w_i \log(1+\exp(-y_i(\mathbf{x}^T\mathbf{a}_i+c)))+\lambda_1\|\mathbf{x}\|_1+\lambda_2\sum_{i=1}^{p-1}|x_i-x_{i+1}|, \tag{24}$$

where $\mathbf{a}_i^T$ denotes the i-th row of $A \in \mathbb{R}^{m\times n}$, w_i is the weight for the i-th sample, $\mathbf{y} \in \mathbb{R}^{m\times 1}$, and $\mathbf{x} \in \mathbb{R}^{n\times 1}$, and c is the intercept (scalar). For illustration of A, $\mathbf{x}$, and $\mathbf{y}$, please refer to Figure 4.

If A is a sparse matrix, you can store A in the sparse format in Matlab, which can significantly improve the efficiency. If you want to normalize the sparse matrix A, you can perform this implicitly through 'opts.nFlag', 'opts.mu', and 'opts.nu'.

By default, you can set 'opts=[]' to use the default settings. For the more advanced usage, you can specify 'opts' (See Section 11). Currently, you can specify the following fields:

- Starting point: .x0, .c0, .init
- Termination: .maxIter, .tol, .tFlag
- Normalization: .nFlag, .mu, .nu
- Regularization: .rFlag
- Method: .lFlag

A note on "opts.rFlag": When opts.rFlag=0, the actual values of λ_1 and λ_2 are used. When opts.rFlag=1, we first compute $\lambda_1^{\max}$ for λ_1, which shall enforce (23) to yield a zero solution, and then the program will use the following regularization values: $\lambda_1 \times \lambda_1^{\max}$ and $\lambda_2 \times \lambda_1^{\max}$.

With properly chosen λ_1 and λ_2, the function fusedLassoLeastR yields an output $\mathbf{x}$ that has sparsity in both the coefficients and their successive differences, as illustrated in Figure 8.

6 Sparse Inverse Covariance Estimation

The function

$$\Theta\text{=sparseInverseCovariance}(S,\, \lambda,\, \text{opts})$$

solves the following sparse inverse covariance estimation problem:

$$\max_{\Theta \succ 0} \log |\Theta| - \langle S, \Theta \rangle - \lambda \|\Theta\|_1, \tag{25}$$

where $S \in \mathbb{R}^{n \times n}$ is the sample covariance matrix estimated from the data, $\Theta \in \mathbb{R}^{n \times n}$ is the (sparse) inverse covariance matrix to be estimated, and $\lambda > 0$ is the regularization parameter.

Our implementation is based on the block coordinate descent described in [14]. The sparse inverse covariance estimation has been used to study the brain connectivity for Alzheimer's Disease [21, 67]. With a properly chosen λ, the function sparseInverseCovariance yields an output Θ that is sparse. Figure 9 illustrates the solution of this function.

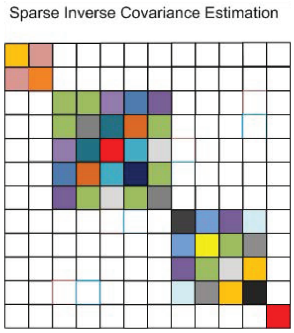


Figure 9: Illustration of the solution of the sparse inverse covariance.

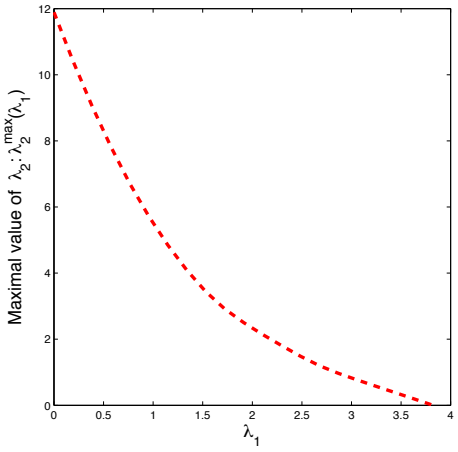


Figure 10: Illustration of $\lambda_1^{\max}$ and $\lambda_2^{\max}(\lambda_1)$. There are $p = 10000$ features, and the features are divided into $k = 100$ non-overlapping groups of size 100.

7 Sparse Group Lasso

In this section, we provide the details of the Moreau-Yosida regularization associated with the sparse group Lasso penalty and the four functions (located at /SLEP/functions/sgLasso) for sparse learning via the sparse group Lasso penalty. The functions described in the following are based on our work in [42].

7.1 The Moreau-Yosida Regularization Associated with the Sparse Group Lasso

The Moreau-Yosida regularization associated with the sparse group Lasso penalty for a given $\mathbf{v} \in \mathbb{R}^p$ is given by:

$$\phi_\lambda(\mathbf{v}) = \min_{\mathbf{x}} \left\{ f(\mathbf{x}) = \frac{1}{2}\|\mathbf{x}-\mathbf{v}\|^2 + \lambda_1\|\mathbf{x}\|_1 + \lambda_2 \sum_{i=1}^{g} w_i^g \|\mathbf{x}_{G_i}\|_2 \right\}. \tag{26}$$

The problem (26) is a special case of the problem (30), and thus can be solved by the function "altra" (located at /SLEP/CFiles/tree/altra.h) to be discussed in Section 8.1. In addition, for the problem 26, we can specify the meaningful interval for parameters λ_1 and λ_2, illustrated in Figure 10. If (λ_1, λ_2) falls above the curve, the solution to (26) shall be exactly zero.

7.2 sgLeastR

The function

$$[\mathbf{x}, \text{funVal}]=\text{sgLeastR}(A, \mathbf{y}, \lambda, \text{opts})$$

solves the sparse group Lasso penalized least squares problem:

$$\min_{\mathbf{x}} \frac{1}{2}\|A\mathbf{x}-\mathbf{y}\|_2^2 + \lambda_1\|\mathbf{x}\|_1 + \lambda_2 \sum_{i=1}^{g} w_i^g \|\mathbf{x}_{G_i}\|_2, \tag{27}$$

where $A \in \mathbb{R}^{m\times n}$, $\mathbf{y} \in \mathbb{R}^{m\times 1}$, $\mathbf{x} \in \mathbb{R}^{n\times 1}$ is divided into k non-overlapping groups $\mathbf{x}_{G_1}, \mathbf{x}_{G_2}, \ldots, \mathbf{x}_{G_g}$. The group and weight ($w_i^g$) information is provided in 'opts.ind'. For illustration of A and $\mathbf{x}$, please refer to Figure 4.

通过设定opts.rFlag=0,程序使用输入的l1和l2，如果设定opts.rFlag=1，程序会自动计算l1max，l1的最大值，(27)的解会包含0值，我们设定l1=l1*l1max。同时，给定一个l1
我们计算l2max(l1)，也应该包含0解。注意如果设定opts.rFlag=1，那么的l1和l2是一个比率值在[0-1]，参考图10，
如果A是一个稀疏矩阵。系数矩阵的

By setting 'opts.rFlag=0', the program uses the input values for λ_1 and λ_2. By setting 'opts.rFlag=1', the program automatically computes $\lambda_1^{\max}$, the maximal value of λ_1, above which (27) shall obtain the zero solution. We set $\lambda_1 \leftarrow \lambda_1 \times \lambda_1^{\max}$. Meanwhile, for a given λ_1, we compute $\lambda_2^{\max}(\lambda_1)$, above which (27) shall also obtain the zero solution. We set $\lambda_2 \leftarrow \lambda_2 \times \lambda_2^{\max}(\lambda_1)$. Note that when 'opts.rFlag=1' the inputs λ_1 and λ_2 should be specified as a ratio lying in the interval $[0, 1]$. Please refer to Figure 10 for an illustration of $\lambda_1^{\max}$ and $\lambda_2^{\max}(\lambda_1)$.

If A is a sparse matrix, you can store A in the sparse format in Matlab. The use of the sparse matrix format can significantly improve the efficiency. If you want to normalize the sparse matrix A, you can perform this implicitly through 'opts.nFlag', 'opts.mu', and 'opts.nu'.

By default, you can set the fields of 'opts' to empty except 'opts.ind'. For the more advanced usage, you can specify 'opts' (See Section 11). Currently, this function supports the following fields:

- Starting point: .x0, .init
- Termination: .maxIter, .tol, .tFlag
- Normalization: .nFlag, .mu, .nu
- Regularization: .rFlag
- Method: .mFlag, .lFlag
- Group & Others: .ind

7.3 sgLogisticR

The function

$$[\mathbf{x}, c, \text{funVal}]=\text{sgLogisticR}(A, \mathbf{y}, \lambda, \text{opts})$$

solves the sparse group Lasso penalized logistic regression problem:

$$\min_{\mathbf{x}} \sum_{i=1}^{m} w_i \log(1 + \exp(-y_i(\mathbf{x}^T \mathbf{a}_i + c))) + \lambda_1 \|\mathbf{x}\|_1 + \lambda_2 \sum_{i=1}^{g} w_i^g \|\mathbf{x}_{G_i}\|_2, \tag{28}$$

where $\mathbf{a}_i^T$ denotes the i-th row of $A \in \mathbb{R}^{m \times n}$, w_i is the weight for the i-th sample, $\mathbf{y} \in \mathbb{R}^{m \times 1}$, $\mathbf{x} \in \mathbb{R}^{n \times 1}$ is divided into k non-overlapping groups $\mathbf{x}_{G_1}, \mathbf{x}_{G_2}, \ldots, \mathbf{x}_{G_k}$, w_i^g denotes the weight for the i-th group, and c is the intercept (scalar). 'opts.ind' provides the group information and the corresponding weights w_i^g. For an illustration of A and $\mathbf{x}$, please refer to Figure 4.

By setting 'opts.rFlag=0', the program uses the input values for λ_1 and λ_2. By setting 'opts.rFlag=1', the program automatically computes $\lambda_1^{\max}$, the maximal value of λ_1, above which (28) shall obtain the zero solution. We set $\lambda_1 \leftarrow \lambda_1 \times \lambda_1^{\max}$. Meanwhile, for a given λ_1, we compute $\lambda_2^{\max}(\lambda_1)$, above which (28) shall also obtain the zero solution. We set $\lambda_2 \leftarrow \lambda_2 \times \lambda_2^{\max}(\lambda_1)$. Note that when 'opts.rFlag=1' the inputs λ_1 and λ_2 should be specified as a ratio lying in the interval $[0, 1]$. Please refer to Figure 10 for an illustration of $\lambda_1^{\max}$ and $\lambda_2^{\max}(\lambda_1)$.

If A is a sparse matrix, you can store A in the sparse format in Matlab. The use of the sparse matrix format can significantly improve the efficiency. If you want to normalize the sparse matrix A, you can perform this implicitly through 'opts.nFlag', 'opts.mu', and 'opts.nu'.

In this function, the elements in $\mathbf{y}$ are required to be either 1 or -1, and w_i's (specified by 'opts.sWeight') satisfy: $w_i = w_j$ if $y_i = y_j$. We also normalize w_i's so that $\sum_i w_i = 1$ holds.

By default, you can set the fields of 'opts' to empty except 'opts.ind'. For the more advanced usage, you can specify 'opts' (See Section 11). Currently, you can specify the following fields:

- Starting point: .x0, .c0, .init
- Termination: .maxIter, .tol, .tFlag
- Normalization: .nFlag, .mu, .nu
- Regularization: .rFlag
- Method: .mFlag, .lFlag
- Group & Others: .ind

7.4 mc_sgLeastR

The function

$$[\mathbf{x}, \text{funVal}]=\text{mc\_sgLeastR}(A, \mathbf{y}, \lambda, \text{opts})$$

solves the sparse group Lasso penalized multi-class least squares problem:

$$\min_{\mathbf{x}} \frac{1}{2}\|A\mathbf{x}-\mathbf{y}\|_2^2 + \lambda_1\|\mathbf{x}\|_1 + \lambda_2\|\mathbf{x}\|_{\ell_1/\ell_2}, \tag{29}$$

where $A \in \mathbb{R}^{m\times n}$, $\mathbf{y} \in \mathbb{R}^{m\times k}$ is the class label matrix, and $\mathbf{x} \in \mathbb{R}^{n\times k}$ is the weight matrix for all k classes. For an illustration of A, $\mathbf{y}$, and $\mathbf{x}$, please refer to Figure 7.

By setting 'opts.rFlag=0', the program uses the input values for λ_1 and λ_2. By setting 'opts.rFlag=1', the program automatically computes $\lambda_1^{\max}$, the maximal value of λ_1, above which (29) shall obtain the zero solution. We set $\lambda_1 \leftarrow \lambda_1 \times \lambda_1^{\max}$. Meanwhile, for a given λ_1, we compute $\lambda_2^{\max}(\lambda_1)$, above which (29) shall also obtain the zero solution. We set $\lambda_2 \leftarrow \lambda_2 \times \lambda_2^{\max}(\lambda_1)$. Note that when 'opts.rFlag=1' the inputs λ_1 and λ_2 should be specified as a ratio lying in the interval $[0, 1]$. Please refer to Figure 10 for an illustration of $\lambda_1^{\max}$ and $\lambda_2^{\max}(\lambda_1)$.

If A is a sparse matrix, you can store A in the sparse format in Matlab. The use of the sparse matrix format can significantly improve the efficiency. If you want to normalize the sparse matrix A, you can perform this implicitly through 'opts.nFlag', 'opts.mu', and 'opts.nu'.

By default, you can set 'opts=[]' to use the default settings. For the more advanced usage, you can specify 'opts' (See Section 11). Currently, this function supports the following fields:

- Starting point: .x0, .init
- Termination: .maxIter, .tol, .tFlag
- Normalization: .nFlag, .mu, .nu
- Regularization: .rFlag
- Method: .mFlag, .lFlag

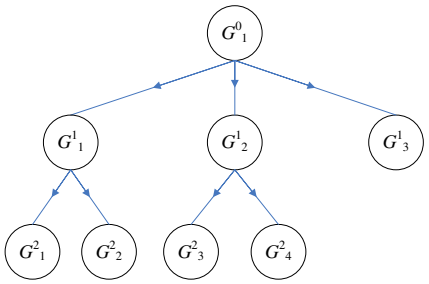


Figure 11: A sample index tree for illustration. Root: $G^0_1 = \{1, 2, 3, 4, 5, 6, 7, 8\}$. Depth 1: $G^1_1 = \{1, 2\}$, $G^1_2 = \{3, 4, 5, 6\}$, $G^1_3 = \{7, 8\}$. Depth 2: $G^2_1 = \{1\}$, $G^2_2 = \{2\}$, $G^2_3 = \{3, 4\}$, $G^2_4 = \{5, 6\}$.

8 Tree Structured Group Lasso

In this section, we provide the details of the Moreau-Yosida regularization associated with the tree structured group Lasso and the six functions (located at /SLEP/functions/tree) for sparse learning via the tree structured group Lasso penalty. The functions described in the following are based on our work in [42].

8.1 The Moreau-Yosida Regularization Associated with the Tree Structured Group Lasso

The Moreau-Yosida regularization associated with the tree structured group Lasso regularization for a given $\mathbf{v} \in \mathbb{R}^p$ is given by:

$$\phi_\lambda(\mathbf{v}) = \min_{\mathbf{x}} \left\{ f(\mathbf{x}) = \frac{1}{2}\|\mathbf{x} - \mathbf{v}\|^2 + \lambda \sum_{i=0}^{d} \sum_{j=1}^{n_i} w_j^i \|\mathbf{x}_{G_j^i}\| \right\}, \tag{30}$$

for some $\lambda > 0$. Here, the group indices G_j^i follow the tree structure depicted in Figure 11.

Denote the minimizer of (30) as $\pi_\lambda(\mathbf{v})$. The Moreau-Yosida regularization has many appealing properties: 1) $\phi_\lambda(\cdot)$ is continuously differentiable despite the fact that $\phi(\cdot)$ is non-smooth; and 2) $\pi_\lambda(\cdot)$ is a non-expansive operator. More properties on the general Moreau-Yosida regularization can be found in [19, 29]. In [42], we showed that, the problem (30) has a closed-form solution.

To solve (30), we provide the following two functions

$$\mathbf{x}\text{=altra(}\ \mathbf{v}, n, \text{ind, nodes)}$$

and

$$\mathbf{x}\text{=general\_altra(}\ \mathbf{v}, n, \text{G, ind, nodes),}$$

where $\mathbf{v}$ is of size n, $\mathbf{x}$ is the result, "nodes" denotes the number of nodes in the tree, and "ind" is a "3 $\times$ nodes" matrix, with ind(1,:) denoting the starting index, ind(2,:) the ending index, and ind(3,:) the weight. For "altra", we assume that the features are well-ordered in the sense that, the indices of the left nodes are smaller than those of the right nodes (see the index tree in Figure 11). For "general_altra", the features might not be well ordered. G is a row vector containing all the indices of the tree in a traverse depth manner, and G(ind(1,i): ind(2,i)) denotes the indices for the i-th group.

The functions "altra" and "general_altra" are written in the standard C language, and are located at the head files /SLEP/CFiles/tree/altra.h and /SLEP/CFiles/tree/general_altra.h, respectively.

We also provide the following two functions

$$\lambda_{\max}\text{=findLambdaMax(}\ \mathbf{v}, n, \text{ind, nodes)}$$

and

$$\lambda_{\max}\text{=general\_findLambdaMax(} \mathbf{v}, n, \text{G, ind, nodes)},$$

for computing the $\lambda_{\max}$, above which (30) shall obtain the zero solution.

8.2 tree_LeastR

The function

$$[\mathbf{x}, \text{funVal}]\text{=tree\_LeastR}(A, \mathbf{y}, \lambda, \text{opts})$$

solves the tree structured group Lasso regularized least squares problem:

$$\min_{\mathbf{x}} \frac{1}{2}\|A\mathbf{x} - \mathbf{y}\|_2^2 + \lambda \sum_{i=0}^{d} \sum_{j=1}^{n_i} w_j^i \|\mathbf{x}_{G_j^i}\|, \tag{31}$$

where $A \in \mathbb{R}^{m\times n}$, $\mathbf{y} \in \mathbb{R}^{m\times 1}$, $\mathbf{x} \in \mathbb{R}^{n\times 1}$ forms a certain tree structure, and w_j^i is the weight. The group (and weight) information is provided in 'opts.ind' (and "opts.G").

By setting 'opts.rFlag=0', the program uses the input value for λ. By setting 'opts.rFlag=1', the program automatically computes $\lambda_{\max}$ (see the discussion in Section 8.1), the maximal value of λ, above which (31) shall obtain the zero solution. In the latter case, the input λ should be specified as a ratio whose value lies in the interval $[0, 1]$, and the actual λ used in the program is $\lambda \times \lambda_{\max}$.

If A is a sparse matrix, you can store A in the sparse format in Matlab. The use of the sparse matrix format can significantly improve the efficiency. If you want to normalize the sparse matrix A, you can perform this implicitly through 'opts.nFlag', 'opts.mu', and 'opts.nu'.

By default, you can set the fields of 'opts' to empty except 'opts.ind'. For the more advanced usage, you can specify 'opts' (See Section 11). Currently, this function supports the following fields:

- Starting point: .x0, .init
- Termination: .maxIter, .tol, .tFlag
- Normalization: .nFlag, .mu, .nu
- Regularization: .rFlag
- Method: .mFlag, .lFlag
- Group & Others: .ind, .G

8.3 tree_LogisticR

The function

$$[\mathbf{x}, c, \text{funVal}]\text{=tree\_LogisticR}(A, \mathbf{y}, \lambda, \text{opts})$$

solves the tree structured group Lasso regularized logistic regression problem:

$$\min_{\mathbf{x}} \sum_{i=1}^{m} w_i \log(1 + \exp(-y_i(\mathbf{x}^T\mathbf{a}_i + c))) + \lambda \sum_{i=0}^{d} \sum_{j=1}^{n_i} w_j^i \|\mathbf{x}_{G_j^i}\|, \tag{32}$$

where $\mathbf{a}_i^T$ denotes the i-th row of $A \in \mathbb{R}^{m\times n}$, w_i is the weight for the i-th sample, $\mathbf{y} \in \mathbb{R}^{m\times 1}$, $\mathbf{x} \in \mathbb{R}^{n\times 1}$ forms a certain tree structure, and w_j^i is the weight. The group (and weight) information is provided in 'opts.ind' (and "opts.G"). For an illustration of A and $\mathbf{x}$, please refer to Figure 4.

By setting 'opts.rFlag=0', the program uses the input value for λ. By setting 'opts.rFlag=1', the program automatically computes $\lambda_{\max}$ (see the discussion in Section 8.1), the maximal value of λ, above which (32) shall obtain the zero solution. In the latter case, the input λ should be specified as a ratio whose value lies in the interval $[0, 1]$, and the actual regularization value used in the program is $\lambda \times \lambda_{\max}$.

If A is a sparse matrix, you can store A in the sparse format in Matlab. The use of the sparse matrix format can significantly improve the efficiency. If you want to normalize the sparse matrix A, you can perform this implicitly through 'opts.nFlag', 'opts.mu', and 'opts.nu'.

In this function, the elements in $\mathbf{y}$ are required to be either 1 or -1, and w_i's (specified by 'opts.sWeight') satisfy: $w_i = w_j$ if $y_i = y_j$. We normalize w_i's so that $\sum_i w_i = 1$.

By default, you can set the fields of 'opts' to empty except 'opts.ind'. For the more advanced usage, you can specify 'opts' (See Section 11). Currently, you can specify the following fields:

- Starting point: .x0, .c0, .init
- Termination: .maxIter, .tol, .tFlag
- Normalization: .nFlag, .mu, .nu
- Regularization: .rFlag
- Method: .mFlag, .lFlag
- Group & Others: .ind, .G

8.4 tree_mtLeastR

The function

[$\mathbf{x}$, funVal]=tree_mtLeastR(A, $\mathbf{y}$, λ, opts)

solves the tree structured group Lasso regularized multi-task least squares problem:

$$\min_{\mathbf{x}} \frac{1}{2}\|A\mathbf{x} - \mathbf{y}\|_2^2 + \lambda \sum_{i=0}^{d} \sum_{j=1}^{n_i} w_j^i \|\mathbf{x}_{G_j^i}\|, \tag{33}$$

where $A = [A_1; A_2; \ldots; A_k] \in \mathbb{R}^{m\times n}$, $A_i \in \mathbb{R}^{m_i \times n}$ is the data matrix for the i-th task ($m = \sum_i m_i$), $\mathbf{y} = [\mathbf{y}_1; \mathbf{y}_2; \ldots; \mathbf{y}_k] \in \mathbb{R}^{m\times 1}$, $\mathbf{y}_i \in \mathbb{R}^{m_i}$ is the response for the i-th task, $\mathbf{x} = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_k] \in \mathbb{R}^{n\times k}$, $\mathbf{x}_i \in \mathbb{R}^n$ is the weight vector for the i-th task, and w_j^i is the weight. The k tasks form a certain tree structure. The group (and weight) information is provided in 'opts.ind' (and "opts.G"). For an illustration of A and $\mathbf{x}$, please refer to Figure 6.

By setting 'opts.rFlag=0', the program uses the input value for λ. By setting 'opts.rFlag=1', the program automatically computes $\lambda_{\max}$ (see the discussion in Section 8.1), the maximal value of λ, above which (33) shall obtain the zero solution. In the latter case, the input λ should be specified as a ratio whose value lies in the interval $[0, 1]$, and the actual regularization value used in the program is $\lambda \times \lambda_{\max}$.

If A is a sparse matrix, you can store A in the sparse format in Matlab. The use of the sparse matrix format can significantly improve the efficiency. If you want to normalize the sparse matrix A, you can perform this implicitly through 'opts.nFlag', 'opts.mu', and 'opts.nu'.

By default, you can set the fields of 'opts' to empty except 'opts.ind'. For the more advanced usage, you can specify 'opts' (See Section 11). Currently, you can specify the following fields:

- Starting point: .x0, .init
- Termination: .maxIter, .tol, .tFlag
- Normalization: .nFlag, .mu, .nu
- Regularization: .rFlag
- Method: .mFlag, .lFlag
- Group & Others: .ind, .G

8.5 tree_mtLogisticR

The function

$$[\mathbf{x}, \mathbf{c}, \text{funVal}]=\text{tree\_mtLogisticR}(A, \mathbf{y}, \lambda, \text{opts})$$

solves the tree structured group Lasso regularized multi-task logistic regression problem:

$$\min_{\mathbf{x}} \sum_{l=1}^{k} \sum_{i=1}^{m_l} w_{il} \log(1+\exp(-y_{il}(\mathbf{x}_l^T \mathbf{a}_{il} + c_l))) + \lambda \sum_{i=0}^{d} \sum_{j=1}^{n_i} w_j^i \|\mathbf{x}_{G_j^i}\|, \tag{34}$$

where $\mathbf{a}_{il}^T$ denotes the i-th sample for the l-th task, w_{il} is the weight for $\mathbf{a}_{il}^T$, y_{il} is the response of $\mathbf{a}_{il}$, and c_l is the intercept (scalar) for the l-th task. We have $A \in \mathbb{R}^{m\times n}$, $\mathbf{x} \in \mathbb{R}^{n\times k}$, $\mathbf{c} \in \mathbb{R}^{1\times k}$, $\mathbf{y} \in \mathbb{R}^{m\times 1}$. The parameter 'opts.ind' provides the indices for the tasks. For an illustration of A, $\mathbf{y}$, and $\mathbf{x}$, please refer to Figure 6.

In (34), λ is the tree structured group Lasso regularization parameter. By setting 'opts.rFlag=0', the program uses the input value for λ. By setting 'opts.rFlag=1', the program automatically computes $\lambda_{\max}$, the maximal value of λ, above which (34) shall obtain the zero solution. In the latter case, the input λ should be specified as a ratio whose value lies in the interval $[0, 1]$, and the actual regularization value used in the program is $\lambda \times \lambda_{\max}$.

If A is a sparse matrix, you can store A in the sparse format in Matlab. The use of the sparse matrix format can significantly improve the efficiency. If you want to normalize the sparse matrix A, you can perform this implicitly through 'opts.nFlag', 'opts.mu', and 'opts.nu'.

By default, you can set the fields of 'opts' to empty except 'opts.ind'. For the more advanced usage, you can specify 'opts' (See Section 11). Currently, you can specify the following fields:

- Starting point: .x0, .c0, .init
- Termination: .maxIter, .tol, .tFlag
- Normalization: .nFlag, .mu, .nu
- Regularization: .rFlag
- Method: .mFlag, .lFlag
- Group & Others: .ind, .q, .fName

In this function, the elements in $\mathbf{y}$ are required to be either 1 or -1. Currently, we simply set $w_{il} = \frac{1}{m}$.

8.6 tree_mcLeastR

The function

$$[\mathbf{x}, \text{funVal}]=\text{tree\_mcLeastR}(A, \mathbf{y}, \lambda, \text{opts})$$

solves the tree structured group Lasso regularized multi-class least squares problem:

$$\min_{\mathbf{x}} \frac{1}{2}\|A\mathbf{x}-\mathbf{y}\|_2^2 + \lambda \sum_{i=0}^{d}\sum_{j=1}^{n_i} w_j^i \|\mathbf{x}_{G_j^i}\|, \tag{35}$$

where $A \in \mathbb{R}^{m\times n}$ is the data matrix, $\mathbf{y} \in \mathbb{R}^{m\times k}$ is the class label matrix, and $\mathbf{x} \in \mathbb{R}^{n\times k}$ is the weight matrix. For an illustration of A, $\mathbf{y}$, and $\mathbf{x}$, please refer to Figure 7.

In (35), λ is the tree structured group Lasso regularization parameter. By setting 'opts.rFlag=0', the program uses the input value for λ. By setting 'opts.rFlag=1', the program automatically computes $\lambda_{\max}$, the maximal value of λ, above which (35) shall obtain the zero solution. In the latter case, the input λ should be specified as a ratio whose value lies in the interval $[0, 1]$, and the actual regularization value used in the program is $\lambda \times \lambda_{\max}$.

If A is a sparse matrix, you can store A in the sparse format in Matlab. The use of the sparse matrix format can significantly improve the efficiency. If you want to normalize the sparse matrix A, you can perform this implicitly through 'opts.nFlag', 'opts.mu', and 'opts.nu'.

By default, you can set 'opts=[]' to use the default settings. For the more advanced usage, you can specify 'opts' (See Section 11). Currently, this function supports the following fields:

- Starting point: .x0, .init
- Termination: .maxIter, .tol, .tFlag
- Normalization: .nFlag, .mu, .nu
- Regularization: .rFlag
- Method: .mFlag, .lFlag
- Group & Others: .q, .fName

8.7 tree_mcLogisticR

The function

$$[\mathbf{x}, \mathbf{c}, \text{funVal}]=\text{tree\_mcLogisticR}(A, \mathbf{y}, \lambda, \text{opts})$$

solves the tree structured group Lasso regularized multi-class logistic regression problem:

$$\min_{\mathbf{x}} \sum_{l=1}^{k}\sum_{i=1}^{m} w_{il}\log(1+\exp(-y_{il}(\mathbf{x}_l^T\mathbf{a}_{il}+c_l))) + \lambda \sum_{i=0}^{d}\sum_{j=1}^{n_i} w_j^i \|\mathbf{x}_{G_j^i}\|, \tag{36}$$

where $\mathbf{a}_{il}^T$ denotes the i-th sample for the l-th task, w_{il} is the weight for $\mathbf{a}_{il}^T$, y_{il} is the response of $\mathbf{a}_{il}$, and c_l is the intercept (scalar) for the l-th task. In multi-class classification, we have $\mathbf{a}_{il} = \mathbf{a}_i, \forall l$, $A \in \mathbb{R}^{m\times n}$, $\mathbf{x} \in \mathbb{R}^{n\times k}$, $\mathbf{c} \in \mathbb{R}^{1\times k}$, $\mathbf{y} \in \mathbb{R}^{m\times k}$. For an illustration of A, $\mathbf{y}$, and $\mathbf{x}$, please refer to Figure 7.

In (36), λ is the tree structured group Lasso regularization parameter. By setting 'opts.rFlag=0', the program uses the input value for λ. By setting 'opts.rFlag=1', the program automatically computes $\lambda_{\max}$,

the maximal value of λ, above which (36) shall obtain the zero solution. In the latter case, the input λ should be specified as a ratio whose value lies in the interval $[0, 1]$, and the actual regularization value used in the program is $\lambda \times \lambda_{\max}$.

If A is a sparse matrix, you can store A in the sparse format in Matlab. The use of the sparse matrix format can significantly improve the efficiency. If you want to normalize the sparse matrix A, you can perform this implicitly through 'opts.nFlag', 'opts.mu', and 'opts.nu'.

By default, you can set 'opts=[]' to use the default settings. For the more advanced usage, you can specify 'opts' (See Section 11). Currently, you can specify the following fields:

- Starting point: .x0, .c0, .init
- Termination: .maxIter, .tol, .tFlag
- Normalization: .nFlag, .mu, .nu
- Regularization: .rFlag
- Method: .mFlag, .lFlag
- Group & Others: .q, .fName

In this function, the elements in $\mathbf{y}$ are required to be either 1 or -1. Currently, we set $w_{il} = \frac{1}{mk}$.

9 Overlapping Group Lasso

In this section, we provide the details of the Moreau-Yosida regularization associated with the overlapping group Lasso and the two functions (located at /SLEP/functions/overlapping) for sparse learning via the overlapping group Lasso penalty. The functions described in the following are based on our work in [41, 78].

9.1 The Moreau-Yosida Regularization Associated with the Overlapping Group Lasso

The Moreau-Yosida regularization associated with the overlapping group Lasso regularization for a given $\mathbf{v} \in \mathbb{R}^p$ is given by:

$$\phi_\lambda(\mathbf{v}) = \min_{\mathbf{x}} \left\{ f(\mathbf{x}) = \frac{1}{2}\|\mathbf{x}-\mathbf{v}\|^2 + \lambda_1\|\mathbf{x}\|_1 + \lambda_2 \sum_{i=}^{g} w_i^g \|\mathbf{x}_{G_i}\| \right\}, \tag{37}$$

where G_i's are the indices for the i-th group. The groups G_i's may overlap.

The function

[$\mathbf{x}$, gap, infor]=overlapping($\mathbf{v}$, p, g, λ_1, λ_2, ind, G, Y, 10^3, 2)

provides the solution to (37), where p and g denote the number features and the number of groups, ind and G contain the group information, and Y is a vector of the same length as G. Here Y is the initial guess of the dual variable. By default, Y is set to the zero vector.

The function "overlapping" (locates at the head file /SLEP/CFiles/overlapping/overlapping.h) is written in the standard C language, and plays a key building block for the functions "overlapping_LeastR" and "overlapping_LogisticR".

9.2 overlapping_LeastR

The function

[$\mathbf{x}$, funVal]=overlapping_LeastR(A, $\mathbf{y}$, λ, opts)

solves the overlapping group Lasso regularized least squares problem:

$$\min_{\mathbf{x}} \frac{1}{2}\|A\mathbf{x}-\mathbf{y}\|_2^2 + \lambda_1\|\mathbf{x}\|_1 + \lambda_2 \sum_{i=}^{g} w_i^g \|\mathbf{x}_{G_i}\|, \tag{38}$$

where $A \in \mathbb{R}^{m\times n}$, $\mathbf{y} \in \mathbb{R}^{m\times 1}$, the groups G_i may overlap, and w_i^g is the weight for the i-th group. The group information is provided in 'opts.ind' and 'opts.G'[6]. For an illustration of A and $\mathbf{x}$, please refer to Figure 4.

By setting 'opts.rFlag=0', the program uses the input values for λ_1 and λ_2. By setting 'opts.rFlag=1', the program automatically computes $\lambda_1^{\max}$, the maximal value of λ_1, above which (38) shall obtain the zero solution. The actual regularization values used in the program are $\lambda_1 \times \lambda_1^{\max}$ and $\lambda_2 \times \lambda_1^{\max}$.

If A is a sparse matrix, you can store A in the sparse format in Matlab. The use of the sparse matrix format can significantly improve the efficiency. If you want to normalize the sparse matrix A, you can perform this implicitly through 'opts.nFlag', 'opts.mu', and 'opts.nu'.

By default, you can set the fields of 'opts' to empty except 'opts.ind'. For the more advanced usage, you can specify 'opts' (See Section 11). Currently, this function supports the following fields:

[6] opts.G is a row vector whose length equals to $\sum_i |G_i|$, opts.G(opts.ind(1,i): opts.ind(2,i)) denotes the indices for the i-th group, and opts.ind(3,i) denotes the weight for the i-th group.

- Starting point: .x0, .init
- Termination: .maxIter, .tol, .tFlag
- Normalization: .nFlag, .mu, .nu
- Regularization: .rFlag
- Method: .mFlag, .lFlag
- Group & Others: .ind, .G

9.3 overlapping_LogisticR

The function

$$[\mathbf{x}, c, \text{funVal}]=\text{overlapping\_LogisticR}(A, \mathbf{y}, \lambda, \text{opts})$$

solves the overlapping group Lasso regularized logistic regression problem:

$$\min_{\mathbf{x}} \sum_{i=1}^{m} w_i \log(1+\exp(-y_i(\mathbf{x}^T\mathbf{a}_i + c))) + \lambda \sum_{i=1}^{k} w_i^g \|\mathbf{x}_{G_i}\|_q, \tag{39}$$

where $\mathbf{a}_i^T$ denotes the i-th row of $A \in \mathbb{R}^{m\times n}$, w_i is the weight for the i-th sample, $\mathbf{y} \in \mathbb{R}^{m\times 1}$, the groups G_i's may overlap, w_i^g is the weight for the i-th group, and c is the intercept (scalar). The group information is provided in 'opts.ind' and 'opts.G'[7]. The group information is provided in 'opts.ind'. For an illustration of A and $\mathbf{x}$, please refer to Figure 4.

By setting 'opts.rFlag=0', the program uses the input values for λ_1 and λ_2. By setting 'opts.rFlag=1', the program automatically computes $\lambda_1^{\max}$, the maximal value of λ_1, above which (39) shall obtain the zero solution. The actual regularization values used in the program are $\lambda_1 \times \lambda_1^{\max}$ and $\lambda_2 \times \lambda_1^{\max}$.

If A is a sparse matrix, you can store A in the sparse format in Matlab. The use of the sparse matrix format can significantly improve the efficiency. If you want to normalize the sparse matrix A, you can perform this implicitly through 'opts.nFlag', 'opts.mu', and 'opts.nu'.

In this function, the elements in $\mathbf{y}$ are required to be either 1 or -1, and w_i's (specified by 'opts.sWeight') satisfy: $w_i = w_j$ if $y_i = y_j$. We normalize w_i's so that $\sum_i w_i = 1$ holds.

By default, you can set the fields of 'opts' to empty except 'opts.ind'. For the more advanced usage, you can specify 'opts' (See Section 11). Currently, you can specify the following fields:

- Starting point: .x0, .c0, .init
- Termination: .maxIter, .tol, .tFlag
- Normalization: .nFlag, .mu, .nu
- Regularization: .rFlag
- Method: .mFlag, .lFlag
- Group & Others: .ind, .G

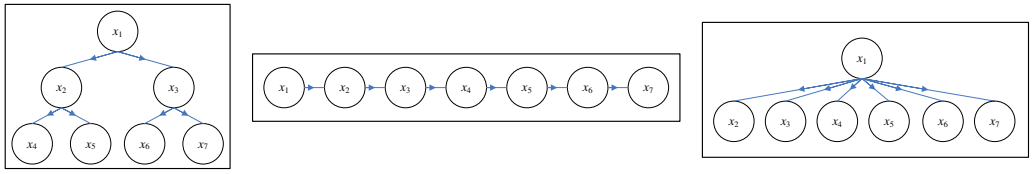
[7] opts.G is a row vector whose length equals to $\sum_i |G_i|$, opts.G(opts.ind(1,i): opts.ind(2,i)) denotes the indices for the i-th group, and opts.ind(3,i) denotes the weight for the i-th group.

10 Ordered Tree–Nonnegative Max-Heap

In the ordered tree–Non-negative Max-Heap, it is assumed that the model parameter $\mathbf{x} \in \mathbb{R}^p$ follows the non-negative max-heap structure[8]:

$$P = \{\mathbf{x} \geq 0, x_i \geq x_j \ \forall (x_i, x_j) \in E^t\}, \tag{40}$$

where $T^t = (V^t, E^t)$ is a target tree with $V^t = \{x_1, x_2, \ldots, x_p\}$ containing all the nodes and E^t all the edges. The constraint set P implies that if x_i is the parent node of a child node x_j then the value of x_i is no less than the value of x_j. In other words, if a parent node x_i is 0, then any of its child nodes x_j is also 0. Figure 12 illustrates three special tree structures: 1) a full binary tree, 2) a sequential list, and 3) a tree with depth 1.



(a) (b) (c)

Figure 12: Illustration of a non-negative max-heap depicted in (40). Plots (a), (b), and (c) correspond to a full binary tree, a sequential list, and a tree with depth 1, respectively.

In estimating the model parameter satisfying the ordered tree structure, one needs to solve the following constrained optimization problem:

$$\min_{\mathbf{x} \in P} f(\mathbf{x}) \tag{41}$$

for some convex function $f(\cdot)$. The problem (41) can be solved via many approaches including subgradient descent, cutting plane method, gradient descent, accelerated gradient descent, etc [53, 50]. In applying these approaches, a key building block is the so-called Euclidean projection of a vector $\mathbf{v}$ onto the convex set P:

$$\pi_P(\mathbf{v}) = \arg\min_{\mathbf{x} \in P} \frac{1}{2}\|\mathbf{x} - \mathbf{v}\|_2^2, \tag{42}$$

which ensures that the solution belongs to the constraint set P.

10.1 Projection onto a Non-negative Max-Heap

The function

$$\mathbf{x}\text{=orderTree(FileName, } \mathbf{v}\text{, rootNum, } n)$$

solves (42). FileName is a string that contains the information for the tree, $\mathbf{v}$ is the input, rootNum is a number that specifies the index of the root of the tree, and n denotes the number of nodes. Table 6 illustrates how to specify a tree for usage in orderTree. In this table, the root of the tree is node 18, and it has 3 child nodes: 10, 13, 17. Node 10 has three child nodes: 5, 8, 9; node 13 has 2 child nodes: 11, 12; node 17 has 3 child nodes: 14, 15, 16; node 5 has 2 child nodes: 1, 4; node 8 has 2 child nodes: 6, 7; node 4 has 2 child nodes: 2, 3; and the node with 0 child node is a leaf node.

When the input tree has some special structures, e.g., a binary tree, a sequential list, and a tree with depth 1, one can solve (42) by

[8]To deal with the negative model parameters, one can make use of the technique employed in [79], which solves the scaled version of the least square estimate.

Table 6: Illustration of how to specify "FileName" for usage in the function orderTree.

Parent node	# Child node	Child nodes
18	3	10 13 17
10	3	5 8 9
13	2	11 12
17	3	14 15 16
5	2	1 4
8	2	6 7
9	0	
11	0	
12	0	
14	0	
15	0	
16	0	
1	0	
4	2	2 3
6	0	
7	0	
2	0	
3	0	

$\mathbf{x}$=orderTreeBinary($\mathbf{v}$, n),

$\mathbf{x}$=sequence_bottomup($\mathbf{v}$, n),

and

$\mathbf{x}$=orderTreeDepth1($\mathbf{v}$, n),

respectively. In these three special cases, the index of the root node is 1.

10.2 orderLeastC

The function

[$\mathbf{x}$, funVal]=orderLeast(A, $\mathbf{y}$, λ, opts)

solves the ℓ_1-norm regularized least squares problem:

$$\min_{\mathbf{x}\in P} \frac{1}{2}\|A\mathbf{x}-\mathbf{y}\|_2^2 + \lambda\|\mathbf{x}\|_1, \tag{43}$$

where $A \in \mathbb{R}^{m\times n}$, $\mathbf{y} \in \mathbb{R}^{m\times 1}$, and $\mathbf{x} \in \mathbb{R}^{n\times 1}$.

In (43), λ is the ℓ_1-norm regularization parameter. By setting 'opts.rFlag=0', the program uses the input values for λ and ρ. By setting 'opts.rFlag=1', the program automatically computes $\lambda_{\max}$, the maximal value of λ, above which (43) shall obtain the zero solution. In the latter case, the input λ should be specified as a ratio whose value lies in the interval $[0, 1]$, and the resulting (ℓ_1) regularization used in the program is $\lambda \times \lambda_{\max}$.

If A is a sparse matrix, you can store A in the sparse format in Matlab. The use of the sparse matrix format can significantly improve the efficiency. If you want to normalize the sparse matrix A, you can perform this implicitly through 'opts.nFlag', 'opts.mu', and 'opts.nu'.

If A is a partial DCT matrix, you need to register this variable as a partialDCT class. The use of the partial DCT matrix can significantly improve the efficiency.

By default, you can set opts=[] to use the default settings. For the more advanced usage, you can specify 'opts' (See Section 11). Currently, this function supports the following fields:

- Starting point: .x0, .init
- Termination: .maxIter, .tol, .tFlag
- Normalization: .nFlag, .mu, .nu
- Regularization: .rFlag
- Method: .mFlag, .lFlag
- Group & Others: .FileName, .treeFlag,.rootNum

When opts.treeFlag=1, it indicates that the tree is a sequential list and

$$\mathbf{x}\text{=sequence\_bottomup}(\mathbf{v}, n)$$

shall be used to solve the associated subproblem (42).

When opts.treeFlag=2, it indicates that the tree is a full binary tree and

$$\mathbf{x}\text{=orderTreeBinary}(\mathbf{v}, n),$$

shall be used to solve the associated subproblem (42). In this case, n, the number of nodes, should satisfy $n = 2^d - 1$, where d is the depth of the tree.

When opts.treeFlag=3, it indicates that the tree is of depth 1 and

$$\mathbf{x}\text{=orderTreeDepth1}(\mathbf{v}, n),$$

shall be used to solve the associated subproblem (42).

When opts.treeFlag=4, it indicates that the tree is specified by an input file and

$$\mathbf{x}\text{=orderTree(FileName, }\mathbf{v}\text{, rootNum, }n)$$

shall be used to solve the associated subproblem (42). In this case, opts.FileName and opts.rootNum should be specified. In opts.FileName, the tree should be set in the same format as shown in Table 6.

11 The Optional Input Parameter—opts

The fields contained in the optional input parameter "opts" are summarized in Table 7. If one or several (even all) fields are not defined, the default settings (marked in brick red) are used. Next, we describe the meaning of these fields.

11.1 Starting Point

All the algorithms are based on the Nesterov's method, which is an iterative method. A good staring point is key to the fast convergence of the algorithm. We provide three fields '.x0','.c0', and '.init':

- '.x0' and '.c0' act as the staring points for $\mathbf{x}$ and c in the functions.
- '.init' tells the program how to process '.x0' ('.c0'). For the least squares loss, when setting '.init=0', '.x0' is processed by the function "initFactor". When '.init=1', we just use the defined '.x0' and '.c0'; when '.init=2', we set '.x0' and '.c0' to zero.
- The default value for '.init' is 0. When '.x0' and '.c0' are undefined, and the loss function is the logistic loss, the program automatically sets '.init=2'.
- When '.init=2', and the logistic loss function is used, we set .c0=$\log(m_1/m_2)$ rather than 0. This choice considers the number of samples for the two classes, and yields better convergence than '.c0=0'i our empirical studies.

11.2 Termination

To terminate the program, we currently provide six different termination criteria, depending on the termination flag '.tFlag' (the default value is 0). '.maxIter' sets an upper bound on the number of iterations (the default value is 1000). '.tol' is a tolerance parameter (the default value is 10^{-4}).

11.3 Normalization

When the data matrix A is of a special structure, e.g., sparse matrix, an explicit normalization will destroy such special structure. One way of preventing this is to use the implicit normalization.

By setting the normalization flag '.nFlag', you can choose whether to normalize the data or not. When '.nFlag=0', A is not normalized in the functions. When '.nFlag=1', A is implicitly normalized as

$$A\text{= } A\text{- repmat}(\boldsymbol{\mu}, m, 1)\ )\text{diag}(\boldsymbol{\nu})^{-1}$$

When '.nFlag=2', A is implicitly normalized as

$$A\text{= diag}(\boldsymbol{\nu})^{-1}(\ A\text{- repmat}(\boldsymbol{\mu}, m, 1)\ )$$

Here, '.mu' and '.nu' can be specified; otherwise, the default values are used. For '.nu', it is required that its elements are positive. Here, A is implicitly normalized, so that A is not changed after calling the functions.

When $\mathbf{x}$ and c are computed, you need to apply the implicit normalization process to get the desired features.

Table 7: The fields of the optional parameter "opts" and the descriptions.

Starting Point	.x0	Staring point of $\mathbf{x}$. If not defined, it will be initialized according to .init.
	.c0	Staring point of c (for the logistic loss only). If not defined, it will be initialized according to .init.
	.init	The flag .init tells how to process .x0. $0 \Rightarrow$ .x0 is set by the function initFactor (for least squares) $1 \Rightarrow$ .x0 and .c0 are defined (if they are not specified, apply .ini=2). $2 \Rightarrow$ .x0 =zeros(n,k), .c0=$\log(m_1/m_2)$.
Termination	.maxIter	The maximum number of iterations. .maxIter=10^4.
	.tol	The tolerance parameter. .tol=10^{-4}.
	.tFlag	The flag for the termination. $0 \Rightarrow$ abs(funVal(i)- funVal(i-1)) $\leq$ .tol. $1 \Rightarrow$ abs(funVal(i)- funVal(i-1)) $\leq$ .tol $\times$ max(funVal(i),1). $2 \Rightarrow$ funVal(i) $\leq$ .tol. $3 \Rightarrow \|\mathbf{x}_i - \mathbf{x}_{i-1}\|_2 \leq$ .tol. $4 \Rightarrow \|\mathbf{x}_i - \mathbf{x}_{i-1}\|_2 \leq$ .tol $\times \max(\|\mathbf{x}_i\|_2, 1)$. $5 \Rightarrow$ Run the code for .maxIter iterations.
Normalization	.nFlag	The flag for (implicit) normalization of A. $0 \Rightarrow$ Without the normalization of A $1 \Rightarrow A$=(A- repmat($\boldsymbol{\mu}$, m, 1))diag($\boldsymbol{\nu}$)$^{-1}$ $2 \Rightarrow A$=diag($\boldsymbol{\nu}$)$^{-1}$(A- repmat($\boldsymbol{\mu}$, m, 1))
	.mu	The (row) vector to be subtracted from each sample. Used when .nFlag $\neq 0$. μ=mean(A,1)
	.nu	The weight (column) vector. Used when .nFlag $\neq 0$. Ensure that $\boldsymbol{\nu} > 0$. If .nFlag=1, $\boldsymbol{\nu}$=(sum(A.^2,1)^T / m)$^{0.5}$. If .nFlag=2, $\boldsymbol{\nu}$=(sum(A.^2,2) / n)$^{0.5}$.
Method	.method	The used method. $0 \Rightarrow$ for those with "R", using the accelerated method [5, 54] $1 \Rightarrow$ for those with "R" and $q = 2$, solving the smooth (reformulated) objective function [36]
	.lFlag	The line search scheme. $0 \Rightarrow$ the Armijo Goldstein rule [50] $1 \Rightarrow$ the adaptive line search proposed in [35] (for smooth ones).
Regularization	.rFlag	The flag for ℓ_1-norm based regularization (functions with "R"). $0 \Rightarrow \lambda$ equals the input value. $1 \Rightarrow \lambda = \lambda_{\max} \times \lambda$.
	.rsL2	Regularization via the squared ℓ_2-norm (rsL2). For functions with ℓ_1 regularization only. .rsL2=0. If .rFlag=1, $\rho = \lambda_{\max} \times$.rsL2; otherwise ρ=.rsL2.
Group & Others	.ind	The indices for the groups. For functions with group information. If the features are naturally ordered, the indices for the i-th group are (ind(i)+1):ind(i+1). If opts.G exists, the indices for the i-th group are opts.G((ind(i)+1):ind(i+1))
	.G	The indices for the groups. For functions with group information. See opts.ind for discussion.
	.q	The value of q If not specified, $q = 2$.
	.sWeight	The weight of the sample (positive and negative) weight. For the Logistic Loss only. Positive sample: .sWeight(1,:); negative sample: sWeight(2,:). If not specified, equal weights for both positive and negative samples.
	.gWeight	The weight for each group (for ℓ_1/ℓ_q regularization). The weights for k groups are given by .gWeight(1:k,1). If not specified, .gWeight(1:k,1)=1.
	.fName	The name of the function. Used in the function pathSolutionLeast and pathSolutionLogistic.

11.4 Regularization

For the ℓ_1 and ℓ_1/ℓ_q norm regularization parameter λ, when 'opts.rFlag=0', the program uses exactly the input value. However, a better choice is to initialize λ as a ratio of $\lambda_{\max}$, by setting 'opts.rFlag=1'. Here $\lambda_{\max}$ is the value above which the function shall return the zero solution. When setting 'opts.rFlag=1', the input λ should be specified as a ratio whose value lies in the interval $[0, 1]$, and the actual regularization value used in the program is $\lambda \times \lambda_{\max}$.

For the ℓ_1-norm regularized or constrained problems (specifically, for the functions "LeastR", "LeastC", "LogisticR", and "LogisticC"), you can specify the regularization parameter for the squared ℓ_2 norm via 'opts.rsL2'. Similar to λ, when '.rFlag=0', 'opts.rsL2' is a given value; when 'rFlag=1', 'opts.rsL2' is a ratio of $\lambda_{\max}$. The default value for 'opts.rsL2' is 0.

11.5 Method

We provide two flags: .mFlag and .lFlag:

- .mFlag stands for the "method flag". It is only applicable to the functions with "R".
  1. When .mFlag=0, we treat the non-smooth convex objective function as the composite function, and apply the method described in [5, 54].
  2. When .mFlag=1, we solve the equivalent constrained smooth reformulation [36], which is a constrained optimization problem with a smooth objective function.
- .lFlag stands for the "line search flag".
  1. When .lFlag=0, we apply the line search scheme proposed by Nemirovski [50]; the characteristic of this line search scheme is that, the step size is monotonically decreasing.
  2. When .lFlag=1 and the optimization problem is (constrained) smooth convex, we apply the adaptive line search scheme proposed in [35]; the characteristic of this line search scheme is that, the step size can be adaptively tuned.
- For the functions with "R", we provide the implementations of the following three combinations of .mFlag and .lFlag:
  1. opts.mFlag=0, opts.lFlag=0
  2. opts.mFlag=1, opts.lFlag=0
  3. opts.mFlag=0, opts.lFlag=1
- For the functions with "C", we provide the implementations for the two combinations:
  1. opts.lFlag=0
  2. opts.lFlag=1

11.6 Group & Others

For functions "glLeastR", "glLogisticR", "mtLeastR", "mtLogisticR", "mtLeastC", and "mtLogisticC", one needs to provide the group information. This is set by 'opts.ind'.

For functions "sg_*", "tree_*", "overlapping_*", one needs to provide the group information. This is set by 'opts.ind' and 'opts.G'.

For the ℓ_1/ℓ_q regularization, one needs to specify the value of q. The default value is 2. We observed in our experiments that, when $q > 100$, the solution is quite similar to $q = \infty$.

For the functions with the logistic loss, we provide the sample weight '.sWeight'. By default, the weight for the positive and negative samples are the same.

To deal with the problem of different group sizes in 'glLeastR' and 'glLogisticR', we provide the group weight '.gWeight'. By default, the weights for different groups are the same.

When calling the functions "pathSolutionLeast" and "pathSolutionLogistic" for computing the pathwise solutions, you need to specify the function name via '.fName'. More details on pathwise solutions are given in the next section.

12 Pathwise Solutions

To compute the pathwise solutions corresponding to a series of λ's, you can use the following functions:

$$X\text{=pathSolutionLeast}(A,\ \mathbf{y},\ \boldsymbol{\lambda},\ \text{opts})$$

$$[X,\ C]\text{=pathSolutionLogistic}(A,\ \mathbf{y},\ \boldsymbol{\lambda},\ \text{opts})$$

for the least squares loss and the logistic loss, respectively.

Here $\boldsymbol{\lambda} = [\lambda_1, \lambda_2, \ldots]$ contains a series of parameters λ_i's. Next, we explain the output parameters X and C for different functions:

- For the functions "LeastR", "LeastC", "nnLeastR", "nnLeastC" and "glLeastR", $X = [\mathbf{x}_1, \mathbf{x}_2, \ldots]$, where $\mathbf{x}_i \in \mathbb{R}^{n\times 1}$ is the solution corresponding to the parameter λ_i.
- For the functions "LogisticR", "LogisticC", "nnLogisticR", "nnLogisticC" and "glLogisticR", $X = [\mathbf{x}_1, \mathbf{x}_2, \ldots]$ and $C = [c_1, c_2, \ldots]$, where $\mathbf{x}_i \in \mathbb{R}^{n\times 1}$ and $c_i \in \mathbb{R}$ are the solutions corresponding to the parameter λ_i.
- For the functions "mtLeastR", "mcLeastR", $X(:, :, i) = \mathbf{x}_i$, where $\mathbf{x}_i \in \mathbb{R}^{n\times k}$ is the solution corresponding to the parameter λ_i.
- For the functions "mtLogisticR", "mcLogisticR", $X(:, :, i) = \mathbf{x}_i$ and $C(i, :) = \mathbf{c}_i$, where $\mathbf{x}_i \in \mathbb{R}^{n\times k}$ and $\mathbf{c}_i \in \mathbb{R}^{1\times k}$ are the solutions corresponding to the parameter λ_i.

It is usually the case that, for computing the solutions corresponding to a series of parameter λ_i's, the two functions above shall outperform a naive implementation which runs the function independently with a given λ_i. The key here lies in the use of the so-called "warm-start" technique.

When using this function, you need to specify the function via 'opts.fName'. For the descriptions of A, $\mathbf{y}$, $\boldsymbol{\lambda}$, please refer to the corresponding functions.

13 Trace Norm Regularized Problems

The problem of minimizing the rank of a matrix variable subject to certain constraints arises in many fields including machine learning, automatic control, and image compression. However, the matrix rank minimization problem is NP-hard in general due to the combinatorial nature of the rank function. A commonly-used convex relaxation of the rank function is the trace norm (nuclear norm) [12], defined as the sum of the singular values of the matrix, since it is the convex envelope of the rank function over the unit ball of spectral norm. In this section, we provide details on several algorithms for solving trace norm regularized problems described in [25, 58].

13.1 accel_grad_mlr

The MATLAB function *accel_grad_mlr* located at /SLEP/functions/trace implements the accelerated gradient algorithm for multivariate linear regression described in [25]. Formally, the problem solved by this function can be expressed as:

$$\min_W \frac{1}{2}||AW - Y||_F^2 + \lambda||W||_*, \tag{44}$$

where A and Y are the data matrix and the label matrix, respectively, in which each row corresponds to a data point and $||W||_*$ denotes the trace norm of W. The inputs and outputs for this function are:

required inputs:

- A: N x D where each row is a data point and N is the sample size
- Y: N x M where M is the number of regression tasks
- lambda: regularization parameter

optional inputs:

- opt.L0: Initial guess for the Lipschitz constant
- opt.gamma: the multiplicative factor for Lipschitz constant
- opt.W_init: initial weight matrix
- opt.epsilon: precision for termination
- opt.max_itr: maximum number of iterations

outputs:

- W: the computed weight matrix
- fval_vec: a vector for the sequence of function values
- itr_counter: number of iterations executed

We provide an example in file *example_accel_grad_mlr.m* to show how to use this function on a small data set. Note that since we ignored the bias term in Eq. (44), the data and label matrices need to be centered.

13.2 accel_grad_mtl

The MATLAB function *accel_grad_mtl* located at /SLEP/functions/trace implements the accelerated gradient algorithm for multi-task learning described in [25]. Formally, the problem solved by this function can be expressed as:

$$\min_W \frac{1}{2}\sum_{i=1}^{k} ||A_i w_i - Y_i||_2^2 + \lambda ||W||_*, \tag{45}$$

where A_i and Y_i are the input and output for the ith task, k is the number of tasks, and $W = [w_1, \cdots, w_k]$.

required inputs:

- A: $k \times 1$ cell in which each cell is $n \times d$ where n is the sample size and d is data dimensionality and k is the number of tasks
- Y: $k \times 1$ cell in which each cell contains the output of the corresponding task
- lambda: regularization parameter

optional inputs:

- opt.L0: Initial guess for the Lipschitz constant
- opt.gamma: the multiplicative factor for Lipschitz constant
- opt.W_init: initial weight matrix
- opt.epsilon: precision for termination
- opt.max_itr: maximum number of iterations

outputs:

- W: the computed weight matrix
- fval_vec: a vector for the sequence of function values
- itr_counter: number of iterations executed

We provide an example in file *example_accel_grad_mtl.m* to show how to use this function on a small data set. Note that since we ignored the bias term in Eq. (45), the data and label matrices need to be centered.

13.3 accel_grad_mc

The MATLAB function *accel_grad_mc* located at /SLEP/functions/trace implements the accelerated gradient algorithm for matrix classification described in [25]. Formally, the problem solved by this function can be expressed as:

$$\min_W \sum_{i=1}^{n} \ell(y_i, \text{Tr}(W^T A_i)) + \lambda ||W||_*, \tag{46}$$

where $(A_i, y_i) \in \mathbb{R}^{m \times n} \times \mathbb{R}$ is the ith sample, ℓ is the logistic regression loss function.

required inputs:

- A: $C \times C \times N$ array where each data point is a $C \times C$ matrix and N is the sample size
- Y: $N \times 1$ vector of class labels
- lambda: regularization parameter

optional inputs:

- opt.L0: Initial guess for the Lipschitz constant
- opt.gamma: the multiplicative factor for Lipschitz constant
- opt.W_init: initial weight matrix
- opt.b_init: initial value for bias
- opt.epsilon: precision for termination
- opt.max_itr: maximum number of iterations
- opt.loss_prim: value of the primal objective function for termination

outputs:

- W: the computed weight matrix
- b: the computed bias
- fval_vec: a vector for the sequence of function values
- itr_counter: number of iterations executed

We provide an example in file *example_accel_grad_mc.m* to show how to use this function on a small data set described in [72].

13.4 mat_primal

The MATLAB function *mat_primal* located at /SLEP/functions/trace implements the primal formulation described in [58]. Formally, this function solves

$$\min_W f(W) = \frac{1}{2}\|AW - Y\|_F^2 + \|W\|_*, \tag{47}$$

where A is $p \times n$ and Y is $p \times m$ (so W is $n \times m$). The code requires that $p >= m$. This code first reduces A to have full row rank. Then it applies an accelerated proximal gradient method to solve the primal problem.

input:

- A: input matrix in which each row is a data point
- Y: output matrix in which each row is a data point
- lambda: regularization parameter

optional input:

- opt.tol: termination tolerance (default $1e-3$)

- opt.freq: frequency of termination checks (default 10)

output:

- W: the optimal solution
- pobj: optimal objective function value of the primal problem

We provide an example in file *example_mat_primal.m* to show how to use this function on a small data set.

13.5 mat_dual

The MATLAB function *mat_dual* located at /SLEP/functions/trace implements the dual formulation described in [58]. Formally, this function solves:

$$\min_W f(W) = \frac{1}{2}\|AW - Y\|_F^2 + \|W\|_*, \tag{48}$$

where A is $p \times n$ and Y is $p \times m$ (so W is $n \times m$). The code requires that $p >= m$. This code first reduces A to have full row rank. Then it applies a feasible descent method to solve the dual problem:

$$\min_U \frac{1}{2} < C, UU' > + < E', U >, \quad \text{s.t. } U'U \leq I, \tag{49}$$

where $C = (A'A)^{-1}$ and $E = Y'AC$ (so U is $m \times n$). From the solution U we obtain $W = E' + CU'$.

input:

- A: input matrix in which each row is a data point
- Y: output matrix in which each row is a data point
- lambda: regularization parameter

optional input:

- opt.alg: which algorithm to use (default 3)
  - opt.alg=1: Frank-Wolfe method with line search
  - opt.alg=2: gradient-projection method with large constant stepsize + line search
  - opt.alg=3: Accelerated grad-proj method with a small constant stepsize
- opt.tol: termination tolerance (default $1e-3$)
- opt.freq: frequency of termination checks (default 10)

output:

- W: the optimal solution
- pobj: optimal objective function value of the original problem

It is our experience that the accelerated grad-proj method with a small constant stepsize is the most efficient algorithm in practice, and hence opt.alg=3 by default. We provide an example in file *example_mat_dual.m* to show how to use this function on a small data set.

13.6 Discussions

The first four algorithms described above solve the trace norm regularized problems in their primal forms and the last one solves the problems in the dual form. The relative efficiency of the primal and dual forms depends on the value of the regularization parameter. We provide an in-depth discussion of this issue in [58].

14 Revision, Citation, and Acknowledgement

Revision History

- Version 1.0 was released on August 13, 2009.
- Version 1.1 was released on September 8, 2009. There are two major improvements: (1) The adaptive line search scheme is implemented for the functions with "R" in Section 2, 3, and 4; and (2) The ℓ_1/ℓ_2-ball constrained optimization problems were added (Section 4).
- Version 2.0 was released on October 10, 2009. The codes for solving the trace norm regularized problems were added in Section 13.
- Version 3.0 was released on February 22, 2010. The codes for solving fused Lasso and sparse inverse covariance were added in Sections 5 & 6, respectively.
- Version 4.0 was released on October 4, 2010. The codes for solving the sparse group Lasso regularization, tree structrured group Lasso regularization, and the overlapping group Lasso regularization were added in Sections 7, 8, and 9, respectively.
- Version 4.1 was released on December 28, 2011. The codes for solving the ordered tree (non-negative max-heap) were added in Section 10. Examples for the projection onto the ℓ_1 ball and FLSA were provided.

If you have any suggestions/corrections for SLEP, please contact us via emails:

{j.liu, shuiwang.ji, jieping.ye}@asu.edu.

Citation

In citing SLEP in your papers, please use the following reference:

J. Liu, S. Ji, and J. Ye. SLEP: Sparse Learning with Efficient Projections. Arizona State University, 2009.
`http://www.public.asu.edu/~jye02/Software/SLEP`

Acknowledgement

The SLEP software project has been supported by research grants from the National Science Foundation (NSF), the National Geospatial Agency (NGA) and by the Office of the Director of National Intelligence (ODNI) Intelligence Advanced Research Projects Activity (IARPA), through the US Army. All statements of fact, opinion or conclusions contained herein are those of the authors and should not be construed as representing the official views or policies of IARPA, the ODNI or the U.S. Government.

References

[1] A. Ahmed and E. P. Xing. Recovering time-varying networks of dependencies in social and biological studies. *Proceedings of the National Academy of Sciences*, 106(29):11878–11883, 2009.

[2] A. Argyriou, T. Evgeniou, and Massimiliano Pontil. Convex multi-task feature learning. *Machine Learning*, 73(3):243–272, 2008.

[3] F. R. Bach. Consistency of the group lasso and multiple kernel learning. *Journal of Machine Learning Research*, 9:1179–1225, 2008.

[4] O. Banerjee, L. Ghaoui, and A. D'Aspremont. Model selection through sparse maximum likelihood estimation for multivariate gaussian or binary data. *Journal of Machine Learning Research*, 2007.

[5] A. Beck and M. Teboulle. A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM Journal on Imaging Sciences*, 2(1):183–202, 2009.

[6] E. Berg, M. Schmidt, M. P. Friedlander, and K. Murphy. Group sparsity via linear-time projection. Tech. Rep. TR-2008-09, Department of Computer Science, University of British Columbia, Vancouver, July 2008.

[7] E. Candès and M. Wakin. An introduction to compressive sampling. *IEEE Signal Processing Magazine*, 25(2):21–30, 2008.

[8] E. J. Candés and B. Recht. Exact matrix completion via convex optimization. Technical Report 08-76, UCLA Computational and Applied Math., 2008.

[9] N. Choi, W. Li, and J. Zhu. Variable selection with the strong heredity constraint and its oracle property. *Journal of the American Statistical Association*, 105:354–364, 2010.

[10] H. Y. Chuang, E. Lee, Y. T. Liu, D. Lee, and T. Ideker. Network-based classification of breast cancer metastasis. *Molecular Systems Biology*, 3(140), 2007.

[11] J. Duchi and Y. Singer. Boosting with structural sparsity. In *International Conference on Machine Learning*, 2009.

[12] M. Fazel, H. Hindi, and S. P. Boyd. A rank minimization heuristic with application to minimum order system approximation. In *Proceedings of the American Control Conference*, pages 4734–4739. 2001.

[13] J. Friedman, T. Hastie, H. Höfling, and R. Tibshirani. Pathwise coordinate optimization. *Annals of Applied Statistics*, 1(2):302–332, 2007.

[14] J. Friedman, T. Hastie, and R. Tibshirani. Sparse inverse covariance estimation with the graphical lasso. *report*, 2007.

[15] J. Friedman, T. Hastie, and R. Tibshirani. A note on the group lasso and a sparse group lasso. Technical report, Department of Statistics, Stanford University, 2010.

[16] T. R. Golub, D. K. Slonim, P. Tamayo, C. Huard, M. Gaasenbeek, J. P. Mesirov, H. Coller, M. L. Loh, J. R. Downing, M. A. Caligiuri, and C. D. Bloomfield. Molecular classification of cancer: Class discovery and class prediction by gene expression monitoring. *Science*, 286(5439):531–537, 1999.

[17] I. Guyon, J. Weston, S. Barnhill, and V. Vapnik. Gene selection for cancer classification using support vector machines. *Machine Learning*, 46(1-3):389–422, 2002.

[18] M. Hamada and C. Wu. Analysis of designed experiments with complex aliasing. *Journal of Quality Technology*, 24:130–137, 1992.

[19] J. Hiriart-Urruty and C. Lemaréchal. *Convex Analysis and Minimization Algorithms I & II*. Springer Verlag, Berlin, 1993.

[20] T. Hromádka, M.R. DeWeese, and A.M. Zador. Sparse representation of sounds in the unanesthetized auditory cortex. *PLoS Biol*, 6(1):e16, 2008.

[21] S. Huang, J. Li, L. Sun, J. Liu, T. Wu, K. Chen, A. Fleisher, E. Reiman, and J. Ye. Learning brain connectivity of alzheimer's disease from neuroimaging data. In *Neural Information Processing Systems*, 2009.

[22] L. Jacob, G. Obozinski, and J. Vert. Group lasso with overlap and graph lasso. In *International Conference on Machine Learning*, 2009.

[23] R. Jenatton, J.-Y. Audibert, and F. Bach. Structured variable selection with sparsity-inducing norms. Technical report, arXiv:0904.3523v2, 2009.

[24] R. Jenatton, J. Mairal, G. Obozinski, and F. Bach. Proximal methods for sparse hierarchical dictionary learning. In *International Conference on Machine Learning*, 2010.

[25] S. Ji and J. Ye. An accelerated gradient method for trace norm minimization. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 457–464, 2009.

[26] S. Kim and E. P. Xing. Tree-guided group lasso for multi-task regression with structured sparsity. In *International Conference on Machine Learning*, 2010.

[27] K. Koh, S. Kim, and S. Boyd. An interior-point method for large-scale l1-regularized logistic regression. *Journal of Machine Learning Research*, 8:1519–1555, 2007.

[28] K.-A. Lê Cao, D. Rossouw, C. Robert-Granié, and P. Besse. A sparse PLS for variable selection when integrating omics data. *Statistical Applications in Genetics and Molecular Biology*, 7(1), 2008.

[29] C. Lemaréchal and C. Sagastizábal. Practical aspects of the moreau-yosida regularization i: Theoretical properties. *SIAM Journal on Optimization*, 7(2):367–385, 1997.

[30] X. Li, N. Sundarsanam, and D. Frey. Regularities in data from factorial experiments. *Complexity*, 11:32–45, 2006.

[31] H. Liu, M. Palatucci, and J. Zhang. Blockwise coordinate descent procedures for the multi-task lasso, with applications to neural semantic basis discovery. In *International Conference on Machine Learning*, 2009.

[32] H. Liu and J. Zhang. Estimation consistency of the group lasso and its applications. In *International Conference on Artificial Intelligence and Statistics*, 2009.

[33] H. Liu and J. Zhang. On the ℓ_1-ℓ_q regularized regression. Technical report, Department of Statistics, Carnegie Mellon University, 2009.

[34] H. Liu, J. Zhang, X. Jiang, and J. Liu. The group dantzig selector. In *International Conference on Artificial Intelligence and Statistics*, 2009.

[35] J. Liu, J. Chen, and J. Ye. Large-scale sparse logistic regression. In *ACM SIGKDD International Conference On Knowledge Discovery and Data Mining*, 2009.

[36] J. Liu, S. Ji, and J. Ye. Multi-task feature learning via efficient $\ell_{2,1}$-norm minimization. In *Uncertainty in Artificial Intelligence*, 2009.

[37] J. Liu, S. Ji, and J. Ye. Mining sparse representations: Formulations, algorithms, and applications. In *SIAM Conference on Data Mining*, 2010.

[38] J. Liu, L. Sun, and J. Ye. Projection onto a nonnegative max-heap. In *Advances in Neural Information Processing Systems*, 2011.

[39] J. Liu and J. Ye. Efficient ℓ_1/ℓ_q-norm regularization. Technical report, Arizona State University, 2009.

[40] J. Liu and J. Ye. Efficient Euclidean projections in linear time. In *International Conference on Machine Learning*, 2009.

[41] J. Liu and J. Ye. Fast overlapping group lasso. *arXiv:1009.0306v1*, 2010.

[42] J. Liu and J. Ye. Moreau-Yosida regularization for grouped tree structure learning. In *Advances in Neural Information Processing Systems*, 2010.

[43] J. Liu, L. Yuan, and J. Ye. An efficient algorithm for a class of fused lasso problems. In *ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2010.

[44] L. Meier, S. Geer, and P. Bühlmann. The group lasso for logistic regression. *Journal of the Royal Statistical Society: Series B*, 70:53–71, 2008.

[45] N. Meinshausen and P. Bühlmann. High dimensional graphs and variable selection with the lasso. *Annals of Statistics*, pages 1436–1462, 2006.

[46] C. Micchelli, J. Morales, and M. Pontil. A family of penalty functions for structured sparsity. In *Advances in Neural Information Processing Systems 23*, pages 1612–1623. 2010.

[47] J.-J. Moreau. Proximité et dualité dans un espace hilbertien. *Bull. Soc. Math. France*, 93:273–299, 1965.

[48] S. Negahban and M. Wainwright. Joint support recovery under high-dimensional scaling: Benefits and perils of $\ell_{1,\infty}$-regularization. In *Advances in Neural Information Processing Systems*, pages 1161–1168. 2008.

[49] J. Nelder. The selection of terms in response-surface models—how strong is the weak-heredity principle? *Annals of Applied Statistics*, 52:315–318, 1998.

[50] A. Nemirovski. *Efficient methods in convex programming*. Lecture Notes, 1994.

[51] A. Nemirovski and D. Yudin. *Problem Complexity and Method Efficiency in Optimization*. John Wiley and Sons, 1983.

[52] Y. Nesterov. A method for solving a convex programming problem with convergence rate $1/k^2$. *Soviet mathematics - Doklady*, 27(2):372–376, 1983.

[53] Y. Nesterov. *Introductory Lectures on Convex Optimization: A Basic Course*. Kluwer Academic Publishers, 2004.

[54] Y. Nesterov. Gradient methods for minimizing composite objective function. *CORE Discussion Paper*, 2007.

[55] G. Obozinski, B. Taskar, and M. I. Jordan. Joint covariate selection for grouped classification. Technical report, Statistics Department, UC Berkeley, 2007.

[56] G. Obozinski, M. Wainwright, and M. Jordan. High-dimensional support union recovery in multivariate regression. In *Advances in Neural Information Processing Systems*, pages 1217–1224. 2008.

[57] J. Peng, J. Zhu, Bergamaschi A., W. Han, D.-Y. Noh, J. R. Pollack, and P. Wang. Regularized multivariate regression for identifying master predictors with application to integrative genomics study of breast cancer. *Annals of Applied Statistics*, page to appear, 2010.

[58] T. K. Pong, P. Tseng, S. Ji, and J. Ye. Trace norm regularization: Reformulations, algorithms, and multi-task learning. *Submitted to SIAM Journal on Optimization*, 2009.

[59] A. Quattoni, X. Carreras, M. Collins, and T. Darrell. An efficient projection for $\ell_{1,\infty}$,infinity regularization. In *International Conference on Machine Learning*, 2009.

[60] F. Rapaport, E. Barillot, and J. Vert. Classification of arraycgh data using fused svm. *Bioinformatics*, 24(13):i375–i382, 2008.

[61] B. Recht, M. Fazel, and P. Parrilo. Guaranteed minimum-rank solutions of linear matrix equations via nuclear norm minimization. *Submitted to SIAM Review*, 2008.

[62] B. Recht, W. Xu, and B. Hassibi. Necessary and sufficient condtions for success of the nuclear norm heuristic for rank minimization. In *Proceedings of the 47th IEEE Conference on Decision and Control*, pages 3065–3070. 2008.

[63] A. Rinaldo. Properties and refinements of the fused lasso. *Annals of Statistics*, 37(5B):2922–2952, 2009.

[64] J. Shi, W. Yin, S. Osher, and P. Sajda. A fast algorithm for large scale ℓ_1-regularized logistic regression. Technical report, CAAM TR08-07, 2008.

[65] B.K. Sriperumbudur, D.A. Torres, and G.R.G. Lanckriet. Sparse eigen methods by D.C. programming. In *International Conference on Machine Learning*, pages 831–838, 2007.

[66] A. Subramanian and et al. Gene set enrichment analysis: A knowledge-based approach for interpreting genome-wide expression profiles. *Proceedings of the National Academy of Sciences*, 102(43):15545–15550, 2005.

[67] L. Sun, R. Patel, J. Liu, K. Chen, T. Wu, J. Li, E. Reiman, and J. Ye. Mining brain region connectivity for alzheimer's disease study via sparse inverse covariance estimation. In *ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2009.

[68] X. Tan, Y. Li, J. Liu, and L. Jiang. Face liveness detection from a single image with sparse low rank bilinear discriminative model. In *European Conference on Computer Vision*, 2010.

[69] R. Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society Series B*, 58(1):267–288, 1996.

[70] R. Tibshirani, M. Saunders, S. Rosset, J. Zhu, and K. Knight. Sparsity and smoothness via the fused lasso. *Journal Of The Royal Statistical Society Series B*, 67(1):91–108, 2005.

[71] R. Tibshirani and P. Wang. Spatial smoothing and hot spot detection for cgh data using the fused lasso. *Biostatistics*, 9(1):18–29, 2008.

[72] R. Tomioka and K. Aihara. Classifying matrices with a spectral regularization. In *Proceedings of the International Conference on Machine Learning*, pages 895–902. 2007.

[73] P. Tseng. Convergence of block coordinate descent method for nondifferentiable minimization. *Journal of Optimization Theory and Applications*, 109:474–494, 2001.

[74] M. J. Van de Vijver and et al. A gene-expression signature as a predictor of survival in breast cancer. *The New England Journal of Medicine*, 347(25):1999–2009, 2002.

[75] D.M. Witten, R. Tibshirani, and T. Hastie. A penalized matrix decomposition, with applications to sparse canonical correlation analysis and principal components. *preprint*, 10(3):515–534, 2009.

[76] J. Ye, J. Chen, R. Janardan, and S. Kumar. Developmental stage annotation of *Drosophila* gene expression pattern images via an entire solution path for LDA. *ACM Transactions on Knowledge Discovery from Data*, 2(1):1–21, 2008.

[77] K. Yosida. *Functional Analysis*. Springer Verlag, Berlin, 1964.

[78] L. Yuan, J. Liu, and J. Ye. Efficient methods for overlapping group lasso. In *Advances in Neural Information Processing Systems*, 2011.

[79] M. Yuan, V. R. Joseph, and H. Zou. Structured variable selection and estimation. *Annals of Applied Statistics*, 3:1738–1757, 2009.

[80] M. Yuan and Y. Lin. Model selection and estimation in regression with grouped variables. *Journal Of The Royal Statistical Society Series B*, 68(1):49–67, 2006.

[81] P. Zhao, G. Rocha, and B. Yu. The composite absolute penalties family for grouped and hierarchical variable selection. *Annals of Statistics*, 37(6A):3468–3497, 2009.

[82] J. Zhu, S. Rosset, T. Hastie, and R. Tibshirani. 1-norm support vector machines. In *Neural Information Processing Systems*, pages 49–56, 2003.

[83] H. Zou, T. Hastie, and R. Tibshirani. Sparse principle component analysis. *Journal of Computational and Graphical Statistics*, 15(2):262–286, 2006.