Quick Manual for Sparse Logistic Regression ToolBox ver1.2.1

By Okito Yamashita

Updated in Feb.2011

First version : Sep.2009

The **S**parse **L**ogistic **R**egression toolbox (**SLR toolbox** hereafter) is a suite of MATLAB functions for solving classification problems. It provides a solution for binary or multi-class classification problem. The unique feature is the weight parameters of the classifier are learned in a sparse way. Thus the algorithm estimates the weight parameters while it automatically finds important features. Due to this unique feature, SLR is applicable to a high-dimensional classification problem where the number of samples is much less than the number of features without suffering from 'the overfitting problem' to some extent. In addition, SLR releases users from the feature selection task that is usually very time-consuming and requires experience. The algorithms are designed for problems of which feature size is up to several thousand and of which sample size is up to several thousand.

~~~~~~

Installation

~~~~~~

SLR toolbox is a suite of MATLAB functions and scripts. MATLAB, a commercial engineering mathematics package, is required to use SLR toolbox. A couple of functions require the optimization toolbox (see below). Codes in the toolbox were written for MATLAB ver7.0.1 or later under LINUX and most functions has been test in MATLAB 7.0.1 and 7.5.

To get installed the toolbox, you just download and unzip the file (for example SLR1.2alpha.zip) wherever you like. Some functions require "Optimization toolbox". See the section **List of "Run-level functions"** for more details.

~~~~~~

Demonstration

~~~~~~

This toolbox has two demonstrations in order to learn how the functions in SLR toolbox work. So working with these demo functions ('demo_*.m') is a good starting point.

In two demo functions ('demo_{binary, multiclass}_classification.m'), you can see how to use SLR toolbox functions for binary or multi-class classification problems with simulated data sets as well as real experimental data (after feature computations). By changing a variable 'data' on the top of each code, you can switch dataset. Real experimental data must be downloaded in addition to SLR toolbox functions. Please download 'TESTDATA.zip' and unzip it in the parallel level where you unzip SLR toolbox.

~~~~~~

Functions in the toolbox

~~~~~~

Functions in SLR toolbox can be separated into 5 classes; "low-level functions", "run-level functions", "demo functions", "common functions", "functions from others" (see figure 1).

- "Low-level functions" are the functions that execute learning classifier parameters. Most functions take a pair of a label vector and a feature matrix as the inputs and output estimated weight parameters.
- "Run-level functions" are the functions that implement whole procedure to solve a classification problem (i.e. normalizing features → learning parameters → testing learned classifier). Most functions take a pair of a label vector and a feature matrix as inputs. Then classifier parameters are learned with training data and the learned classifier is evaluated with test data. Learned parameters as well as some performance measure will be output. There are seven functions for the binary classification problem and six functions for the multi-class classification problem (see below). It should be sufficient for most of users to know how to use these functions.
- "Demo functions" are for demonstration purpose. You can learn how functions in the toolbox work using some examples. Please run these functions at first.
- "Common functions" are the functions that are commonly used in toolbox.
- "Functions from others" are the functions that are borrowed from EEGLAB toolbox and MATLAB File exchange.

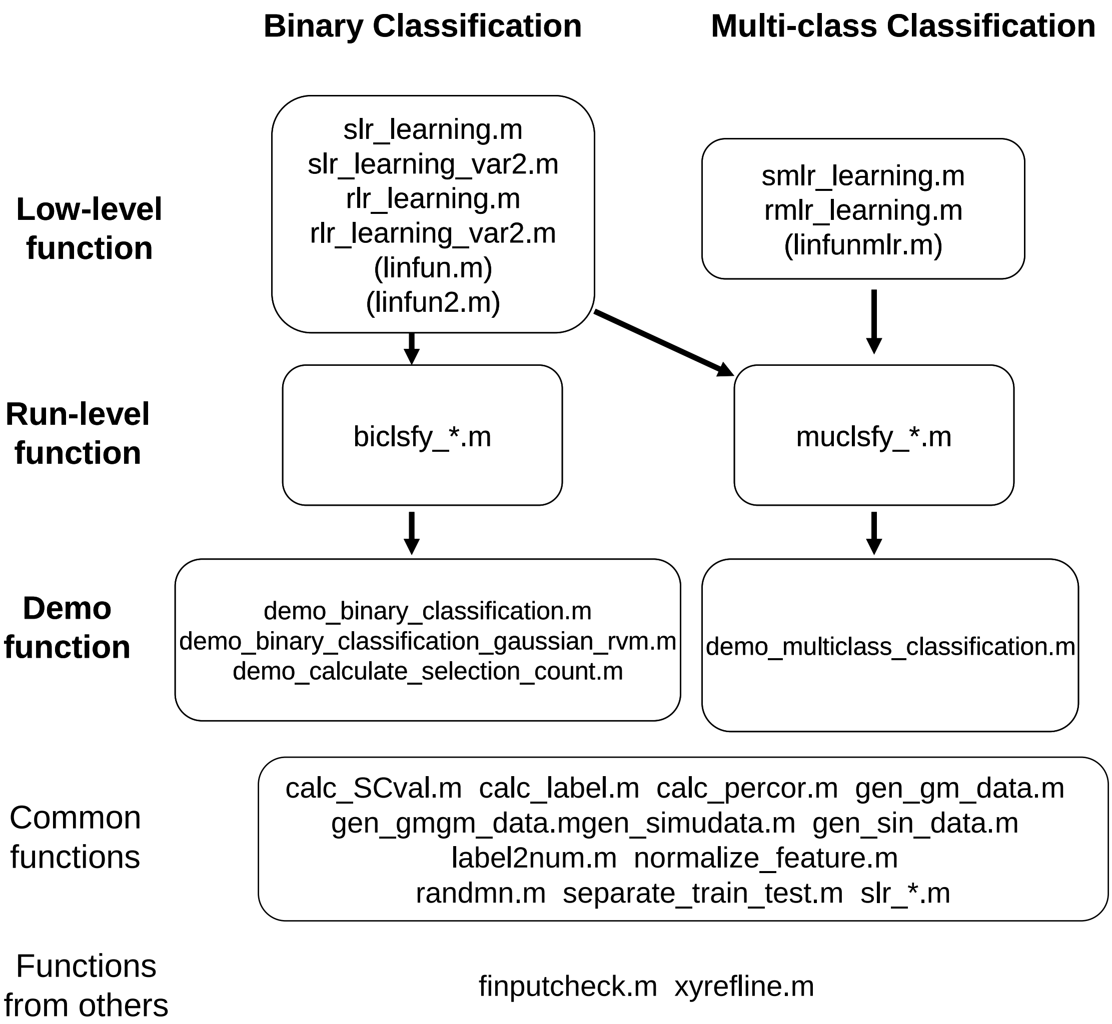

Figure 1: Functions in SLR toolbox

~~~~~~

List of "Run-level functions"

~~~~~~

In "run-level functions", each of seven functions of which name starts from 'biclsfy_*' implements binary classifiers and each of six functions of which name starts from 'muclsfy_*' implements multi-class classifiers (see Table 1). Classifier in the row of 'Sparse' has feature selection property while classifiers in the row of 'Others' do not (for comparison purpose).

Table 1: Supported classifiers in SLR toolbox (Run-level functions)

	Binary classification	Multi-class classification
Sparse	SLR-LAP (biclsfy_slrlap.m) SLR-VAR (biclsfy_slrvar.m) L1-SLR-LAP (biclsfy_l1slrlap.m) L1-SLR-C (biclsfy_l1slrc.m)	SMLR (muclsfy_smlr.m) SLR-LAP-1vsR (muclsfy_slrlapovrm.m) SLR-VAR-1vsR (muclsfy_slrvarovrm.m) SLR-VAR-1vs1 (muclsfy_slrvarovo.m)
Others	RLR-LAP (biclsfy_rlrlap.m) RLR-VAR (biclsfy_rlrvar.m) RVM (biclsfy_rvm.m)	RMLR (muclsfy_rmlr.m) RLR-VAR-1vsR (muclsfy_rlrvarovrm.m)

SLR = Sparse Logistic Regression
L1-SLR = L1-norm Sparse Logistic Regression
RLR = Regularized Logistic Regression
RVM = Relevance Vector Machine
SMLR = Sparse Multinomial Logistic Regression
RMLR = Regularized Multinomial Logistic Regression

LAP = with Laplace approximation
VAR = with variational approximation
C = with component-wise update
1vsR = One-Versus-Rest
1vs1 = One-Versus-One

Binary classifiers

Two sparse classifiers and three non-sparse classifiers are supported. Another two sparse classifiers were implemented (2009/06).

- **SLR-LAP**: SLR with Laplace approximation. The marginal posterior-distribution of weight parameters is approximated by multivariate Gaussian distribution (see ref.[1] for details). This was developed for the research in ref.[1]. The optimization toolbox is required (‘fminunc.m’).

- **SLR-VAR**: SLR with variational approximation. The logistic function is approximated by Gaussian distribution using a variational parameter (see ref.[2.3] or section 10.6 of ref.[4]). The faster and the less memory.
- **RLR-LAP**: Regularized logistic regression with Laplace approximation. This is not sparse algorithm. The regularization parameter is automatically determined by the algorithm. The optimization toolbox is required (‘fminunc.m’).
- **RLR-VAR**: Regularized logistic regression with variational approximation. This is not sparse algorithm. The regularization parameter is automatically determined by the algorithm.
- **RVM**: Relevance Vector Machine as proposed by Tipping (see ref.[5]). Bayesian version of Support Vector Machine (SVM). The linear and Gaussian kernels are supported. The Gaussian kernel RVM is only non-linear classifier supported in this toolbox.
- **L1-SLR-LAP**: L1-norm based sparse logistic regression (see ref.[7]). This classifier also provides sparse solution. The user must tune one parameter that determines extent of sparsity. This program has not been debugged carefully yet. The optimization toolbox is required (‘fminunc.m’).
- **L1-SLR-C**: L1-norm based sparse logistic regression (see ref.[6]). This is fast and requires less memory. This program has not been debugged carefully yet.

Two sparse classifiers, SLR-LAP and SLR-VAR, are derived from the identical probabilistic model (see ref[1]) but different approximation to the posterior distribution. The difference between RLR-LAP and RLR-VAR is as in the same way. Among two sparse classification methods, I recommend SLR-VAR because it is faster and require less memory. L1-SLR-LAP and L1-SLR-C are also sparse classifiers that is based on the different probabilistic model form SLR-LAP(-VAR). I have not debugged these two codes carefully yet.

<u>Multi-class classifier</u>

Four sparse classifiers and two non-sparse classifiers are supported so far.

- **SMLR :** Sparse Multinomial Logistic Regression (see ref.[1]). The multinomial distribution is used for observation. In general, memory and time required are huge. The optimization toolbox is required (‘fminunc.m’).
- **SLR-LAP-1vsR :** Combination of SLR-LAP classifiers. One-versus-the rest scheme is

used. The optimization toolbox is required (‘fminunc.m’).

- **SLR-VAR-1vsR :** Combination of SLR-VAR classifiers. One-versus-the rest scheme is used. The faster computation and the less memory.
- **RMLR :** Regularized Multinomial Logistic Regression. One regularization parameter common to all the classes is automatically estimated. The optimization toolbox is required (‘fminunc.m’).
- **RLR-VAR-1vsR :** Combination of SLR-VAR classifiers. One-versus-the rest schemes is used. One regularization parameter per each class is automatically estimated.
- **SLR-VAR-1vs1**: Combination of SLR-VAR classifiers. One-versus-the one scheme is used. The faster computation and the less memory.

SMLR is a true multinomial classifier that uses multinomial distribution as the likelihood function. SLR-LAP-1vsR and SLR-VAR-1vsR are consisting of combination of sparse binary classifiers (see one-versus-the-rest scheme in chapter4 of ref[4] for example). In this release, one-versus-one scheme is also supported for SLR-VAR.

In theory, SMLR is the best classifier for multi-class problem since the model learning take into account all the information among classes. But in my experience SLR-LAP-1vsR or SLR-VAR-1vsR, SLR-VAR-1vs1 do perform as well as SMLR probably due to a small number of training samples. Since SMLR requires rather large amount of memory and long computation time whereas SLR-VAR-1vs1 requires less memory and computation time, I recommend SLR-VAR-1vs1 for the first trial. It should be noted that the current implementation of SLR-VAR-1vs1 assumes that each label has the same number of samples (this assumption is used when combining multiple binary classifiers results.).

~~~~~~

How to use “Run-level functions”

~~~~~~

The details of “biclsfy_slrvar.m” are explained as an example of binary classifiers. You can easily understand the structure and options of other run-level functions since they share the same structure and the similar but slightly different options with “biclsfy_slrvar.m”.

The function has following format;

- Syntax

[ww, ix_eff_all, errTable_tr, errTable_te, parm, AXall,Ptr,Pte] =…

biclsfy_slrvar(x_train, t_train, x_test, t_test, varargin).

- Example of Usage

[ww, ix_eff, errTable_tr, errTable_te] = biclsfy_slrvar(xtr, ttr, xte, tte, ‘nlearn’, 300, ‘mean’, ‘displaytext’, 0);

- Input variables

Name	Brief explanation	Type	Details
x_train	Training feature matrix	N x D matrix	Each row represents a feature vector for a specific sample.
t_train	Training label vector	N x 1 vector	The *n*th element represents the task label for the *n*th row of training feature matrix (i.e. feature vector of the *n*th sample). Elements of this vector must consist of any two integer values (e.g. {0,1} or {1,2}). .
x_test	Test feature matrix	M x D matrix	Same form as x_train
t_test	Test label vector	M x 1 vector	Same form as t_train

* The variables N and M represents the number of training and test samples, respectively.

* The variable D represents the number of features.

- Optional Input variables

Name	Brief explanation	Type	Details
scale_mode	Mode of scaling when normalizing the feature matrices	String [‘each’]	Each row represents a feature vector for a specific sample.
mean_mode	Mode of mean correction when normalizing the feature matrices	String [‘each’]	The *n*th element represents the task label for the *n*th row of training feature matrix (i.e. feature vector of the *n*th sample). Elements of this vector must consist of any two integer values (e.g. {0,1} or {1,2}). .
ax0	Initial relevance parameter	Scalar [1]	This value is set to all the relevance parameters.
nlearn	Number of iterations for learning a classifier	Scalar[1000]	

nstep	Number of iterations for displaying an intermediate result during learning.	Scalar[100]	Invalid when 'displaytext' option is 0.
amax	Maximum relevance parameter	Scalar [10^8]	Features of which relevance parameters exceed this value are removed from a classifier. This threshold is important to avoid computational instability.
usebias	Flag for adding a constant term or not	Boolean [1]	If 1, the constant vector ones(N,1) is added to the last column of feature matrices.
norm_sep	Flag for applying normalization to training and test data using different scaling and mean correction factors.	Boolean [0]	
displaytext	Flag for displaying a result during learning iterations	Boolean [1]	
invhessian	Flag for using inverse of Hessian matrix during learning a classifier	Boolean [1]	If 1, an inverse of Hessian matrix (size D x D) is used in every learning iteration. If 0, equivalent matrix manipulations (size N x N) is used.

* The default values usually work reasonably well.

- Output variables

Name	Brief explanation	Type	Details
ww	Weight vector,	D (or D+1) x 1 matrix	The last element corresponds to a constant vector if usebias = 1 or the size of a vector is D+1.
ix_eff_all	Indicies of features selected	1 x 1 cell	
errTable_tr	Confusion matrix for training data	2 x 2 matrix	The 1st row represents the number of samples classified as 1 or 2 when the true label is 1. The 2nd row represents the number of samples classified as 1 or 2 when the true label is 2.
errTable_te	Confusion matrix for test	2 x 2 matrix	Same form as errTable_tr

	data		
parm	Parameter struct	struct	
Axall	History of relevance vector updates	x D vector	
Ptr	Probabilistic output of training data	N x 2 vector	The 1st and 2nd column represent the probability of training samples classified as label 1 and 2, respectively.
Pte	Probabilistic output of test data	M x 2 vector	Same form as Ptr

* The variables N and M represents the number of training and test samples, respectively.

* The variable D represents the number of features.

In the function, the following processing is successively executed;

1. check optional variables
2. normalize the feature matrices
3. add a bias regressor to the feature matrices depending on 'parm.usebias'
4. learn weight parameters (boundary parameters) of a classifier
5. evaluate percent correct for training and test data.

The step 4 is the main step where the low-level function "slr_learning_var2" is called.
If you would like to modify this run-level function for your purpose, please understand what each 5 step is doing and refer documents of low-level functions to learn how to call these low-level functions.

~~~~~~

FAQ

~~~~~~

Q1. What does the normalization step exactly do ?
A1. The normalization is simply to apply scaling and mean-correction to each element in the feature matrix so that all the elements have appropriate range (basically -1 to 1). This step is important to avoid an ill-conditioned problem in computation.
If 'mean_mode' = 'each' (default), the mean is calculated in a feature-wise way (i.e. each feature has different mean). If 'mean_mode' = 'all', the mean is calculated as average over all the elements (i.e all the feature shares one common mean value). In the same way, 'scale_mode' defines the way to compute scaling factors.

If ‘norm_sep’ = 0 (default), the mean and scaling is computed using only training data and then they are applied to both training and test feature matrices. On the other hand, if ‘norm_sep’ = 1, the mean and scaling is separately computed for training and test data (this is not recommended).
If both ‘mean_mode’ and ‘scale_mode’ are each, the following affine transformation is applied,

$$\tilde{\mathbf{x}}_d = \frac{\mathbf{x}_d - m_d}{a_d} \quad \begin{cases} m_d = mean(\mathbf{x}_d) \\ a_d = \max_n \{ | x_{dn} - m_d | \} \end{cases}$$

where $X = [\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_D] : N \times D$ is a feature matrix.

Q2. How do I compute percent correct from ‘errTable_tr’ or ‘errTable_te’?
A2. Use the “calc_percor.m” function in the toolbox like
> percent_tr = calc_percor(errTable_tr)

Q3. Why is the size of the output variable ‘ww’ the number of feature plus one?
A3. If ‘use_bias’ = 1 (default), the bias-term is automatically concatenated to the last row of the input feature matrix. The last row or value of ‘ww’ is the weight of this bias regressor.

Q4. Is it possible to speed up the learning step?
A4. If you use the ‘biclsfy_slrvar.m’ function, changing optional input ‘invhessian’ may improve the computation speed dramatically without affecting classification results. If the dimension of your feature matrix is larger than the number of training samples, setting ‘invhessian’ to 0 is fast. Otherwise setting ‘invhessian’ to 1 is fast.

~~~~~~

Mathematical Basics

~~~~~~

Logistic regression (LR) is a well-known classifier originally developed in statistics. SLR is a Bayesian extension of LR in which a sparseness prior is imposed on LR. In the literature, two kinds of the sparsenss prior has been suggested; Automatic Relevance Determination (ARD) prior ([8,9]) and Laplace prior ([5]). In this toolbox the ARD prior is employed. Please see the appendix of [1] for equations of the model and derivation of the algorithm. It should be noted that for SLR having feature selection property, the boundary function must be linear (but not be linear kernel representation). This is because each feature has its own weight parameter and thereby sparse estimation of weight parameters can be interpreted as removing irrelevant features. For more details on the derivation of the algorithms, please read “Mathematical_Issue.pdf” in this toolbox.

~~~~~~
Referencing the toolbox
~~~~~~

When using this tool for a paper please refer to the following paper:

Yamashita O, Sato MA, Yoshioka T, Tong F, Kamitani Y (2008). Sparse estimation automatically selects voxels relevant for the decoding of fMRI activity patterns. *Neuroimage*. Oct 1;42(4):1414-29.

The above manuscript contains basics of SLR (SLR-LAP and SMLR) and applications to fMRI decoding.

~~~~~~
Feedback & Bug report
~~~~~~

Any feedback and bug report are welcome. Please keep contact with me (oyamashi@atr.jp). I would like to respond as quickly as possible.

~~~~~~
Licensing & Copy Right
~~~~~~

Acknowledgements

This toolbox is brought to you by ATR Computational Neuroscience laboratories in Kyoto. This research was supported in part by the NICT, Honda Research Institute, the SCOPE, SOUMU, the Nissan Science Foundation, and grants from the National Eye Institute to FT (R01 EY017082 and R01 EY14202).

References

----

[1] Yamashita O, Sato MA, Yoshioka T, Tong F, Kamitani Y (2008).
Sparse estimation automatically selects voxels relevant for the decoding of fMRI activity patterns. *Neuroimage*. Oct 1;42(4):1414-29.

[2]Jaakkola TS, Jordan MI (2000), Bayesian parameter estimation via variational methods, *Statistics and Computing*, 10, pp.25--37

[3] Bishop C, Tipping ME (2000),Variational relevance vector machines, *Proceedings of the 16th Conference in Uncertainty in Artificial Intelligence*, pp.46-53

[4] Bishop C (2006), Pattern recognition and machine learning, Springer, New York

[5] Tipping ME(2001), Sparse Bayesian Learning and the Relevance Vector Machine, *J Machine Learning Research*, 1, pp.211-244

[6] Krishnapuram B, Carin L, Figueiredo MAT, and Hartemink AJ (2005), Sparse Multinomial Logistic Regression Fast Algorithms and Generalization Bounds, IEEE Trans. Pattern Analysis and Machine Intelligence, 27, pp.957-968

[7] Krishnapuram B, Hartemink AJ, Carin L and Figueiredo MAT (2004), A Bayesian Approach to Joint Feature Selection and Classifier Design, IEEE Trans. Pattern Analysis and Machine Intelligence, 26, pp.1105-1111

[8] MacKay D (1992), Bayesian interpolation, Neural Computation, 4, pp.415-447

[9] Neal RM(1996), Bayesian Learning for Neural Networks, Lecture Notes in Statistics 118,. Springer