MMT: A Matlab Library for Multi-task Learning

Xing Xu @ TTIC

April 27, 2012

Contents

1 Multi-task Learning

Multi-task learning is a machine learning problem with a wide range of applications, including conjoint analysis in marketing science and matrix completion in recommendation systems. The idea of multi-task learning is instead of learning machine learning problems seperately, one can construct algorithm to do multiple learning tasks simutaenously in order to share information among similar learning tasks, it is especially useful when we only have few samples for each task. This library includes three popular methods to deal with multi-task scenarios, which are lasso, multi-task lasso (group lasso) and graph guided multi-task lasso.

Here first introduce the notations, we have n samples stacked to a n by J observation matrix, with each row represents a sample and each column is a feature. We use B_i denotes the i-th column of matrix B, while B^j is the j-th row of B. Y is a n by K label matrix, with K be the number of tasks.

In this library, we assume a linear model underlying the data we observed, that is, for each task $t \in \{1 \dots K\}$, we have

$$Y_t = XB_t + \epsilon$$

or equivalently

$$Y = XB + \epsilon$$

where ϵ is a relatively small gaussian noise, we want to estimate association matrix B from X and Y. One natural way to solve this without any futher assumptions is by least squares

$$\hat{B} = \arg\min_B \|Y - XB\|_2^2$$

Three models we provide are as follows:

1.1 lasso

Modern machine learning applications are typically in high dimensions, which means the number of samples we have is significantly smaller than the number of features, we have to do feature selection, otherwise it would be easily get overfitting and so that the model we learned has a bad generalizaion ability. A classic way to do is by adding an ℓ_1 term, which is the convex surrogate of ell_0 penalty. Lasso estimated $\hat{B}$ is sparse, which means only a small number of positions in $\hat{B}$ will be nonzero.

$$\hat{B} = \arg\min_B \|Y - XB\|_2^2 + \lambda\|B\|_1$$

1.2 multi-task lasso

The lasso method actually doesn't use any shared information between tasks. Here the multi-task lasso assumes a small common subset of features among all tasks, this is done by adding an $\ell_{2,1}$ term, which makes $\hat{B}$ has a lot of rows of zeros.

$$\hat{B} = \arg\min_B \|Y - XB\|_2^2 + \lambda \|B\|_{2,1}$$

or its group lasso form

$$\hat{B} = \arg\min_B \|Y - XB\|_2^2 + \lambda \sum_i B_{g_i}$$

1.3 graph guided multi-task

In real applications, we might have further prior informations about features and tasks in the form of a feature graph and a task graph. A feature (task) graph is a graph of features (tasks), with an edge connect two feature (task) nodes if these two features (tasks) are essentially similar. $G_1 = (V_1, E_1)$ is the graph on the features, while $G_2 = (V_2, E_2)$ is the graph of tasks. Then connected features and tasks are fused together.

$$\hat{B} = \arg\min_B \|Y - XB\|_2^2 + \lambda \|B\|_1 + \gamma_1 \sum_{(i,j)\in E_1} \|B_i - B_j\|_1 + \gamma_2 \sum_{(i,j)\in E_2} \|B^i - B^j\|_1$$

All the regularization parameters above $(\lambda, \gamma_1, \gamma_2)$ are chosen by cross validation.

2 Algorithm

A fast coordinate descent algorithm is adopted in this library. The objective function with ℓ_1 penalty is non-differentiable and its optimization is achieved by transforming it to a series of smooth functions that can be efficiently minimized by the coordinate-descent algorithm. Specifically, our algorithm works as follows. First, we consider the following constrained ridge-type

optimization

$$\text{minimize}_{B,d_{jk},d1_{jml},d2_{kfg}} \qquad \|Y - XB\|_F^2 + \lambda \sum_{j=1}^{J}\sum_{k=1}^{K} \frac{b_{jk}^2}{d_{jk}}$$

$$+ \gamma_1 \sum_{e_{m,l}\in E_1} w^2(e_{m,l}) \sum_{j=1}^{J} \frac{(b_{jm} - \text{sign}(r_{m,l})b_{jl})^2}{d1_{jml}}$$

$$+ \gamma_2 \sum_{e_{f,g}\in E_2} w^2(e_{f,g}) \sum_{k=1}^{K} \frac{(b_{fk} - \text{sign}(r_{f,g})b_{gk})^2}{d2_{kfg}},$$

subject to

$$\sum_{j,k} d_{jk} = 1, \qquad \sum_{e_{m,l}\in E_1, j} d1_{jml} = 1,$$

$$\sum_{e_{f,g}\in E_2, k} d2_{kfg} = 1, \qquad d_{jk}, d1_{jml}, d2_{kfg} \geq 0.$$

This can be analytically solved via its Lagrangian form. For an initial value of B, we optimize over $d_{jk}, d1_{jml}, d2_{kfg}$ by setting their corresponding derivatives to zeros; hence we obtain

$$d_{jk} = \frac{|b_{jk}|}{\sum_{j',k'} |b_{j'k'}|},$$

$$d1_{jml} = \frac{w(e_{m,l})|b_{jm} - \text{sign}(r_{m,l})b_{jl}|}{\sum_{e_{m',l'}\in E_1, j'} w(e_{m',l'})|b_{j'm'} - \text{sign}(r_{m',l'})b_{j'l'}|},$$

$$d2_{kfg} = \frac{w(e_{f,g})|b_{fk} - \text{sign}(r_{f,g})b_{gk}|}{\sum_{e_{f',g'}\in E_2, k'} w(e_{f',g'})|b_{f'k'} - \text{sign}(r_{f',g'})b_{g'k'}|}.$$

Then conditioning on the current estimate of $d_{jk}, d1_{jml}, d2_{kfg}$, we optimize over B. The solution of this minimization can be found as

$$
\begin{aligned}
b_{jk} =& \Big\{ \sum_{i=1}^{n} x_{ij}(y_{ik} - \sum_{j' \neq j} x_{ij'} b_{j'k}) \\
&+ \gamma_1 \sum_{e_{m,k} \in E_1} w^2(e_{m,k}) \frac{b_{jm}\text{sign}(r_{m,k})}{d1_{jmk}} + \gamma_1 \sum_{e_{k,l} \in E_1} w^2(e_{k,l}) \frac{b_{jl}\text{sign}(r_{k,l})}{d1_{jkl}} \\
&+ \gamma_2 \sum_{e_{f,j} \in E_2} w^2(e_{f,j}) \frac{b_{fk}\text{sign}(r_{f,j})}{d2_{kfj}} + \gamma_2 \sum_{e_{j,g} \in E_2} w^2(e_{j,g}) \frac{b_{gk}\text{sign}(r_{j,g})}{d2_{kjg}} \Big\} \\
&\Big/ \Big\{ \sum_{i=1}^{n} x_{ij}^2 + \frac{\lambda}{d_{jk}} \\
&+ \gamma_1 \sum_{e_{m,k} \in E_1} \frac{w^2(e_{m,k})}{d1_{jmk}} + \gamma_1 \sum_{e_{k,l} \in E_1} \frac{w^2(e_{k,l})}{d1_{jkl}} \\
&+ \gamma_2 \sum_{e_{f,j} \in E_2} \frac{w^2(e_{f,j})}{d2_{kfj}} + \gamma_2 \sum_{e_{j,g} \in E_2} \frac{w^2(e_{j,g})}{d2_{kjg}} \Big\}.
\end{aligned}
$$

These two steps alternate until $\left\| B^{(t+1)} - B^{(t)} \right\|_1 \leq \varepsilon$ for some small $\varepsilon > 0$.

Tuning parameters $\lambda, \gamma_1, \gamma_2$ are determined by K-fold cross-validations (CVs). Since an exhaustive search of the optimal triplet on a three-dimensional lattice is computationally infeasible for large-scale multi-task learning problems, we adopt a gradient-descent approach to iteratively update $(\lambda, \gamma_1, \gamma_2)$. In particular, three line searches in the descent direction of minimizing the current CV error are sequentially applied to each component in $(\lambda, \gamma_1, \gamma_2)$ while holding the other two components. The coordinate gradients for the three components are approximated by their finite differences.

3 Features

3.1 Fast

The program is faster than its competitive methods such as subgradient descent.

3.2 Scalable

The matlab library uses C/mex programs to accelerate computation and to make it scalable. It has been heavily tested on matrices as large as $10^4 \times 10^4$. See first several lines of each file for a detailed description.

4 Install

4.1 Preparation

Make sure the following things are done before installation:

1. You have a MATLAB software installed on your computer.
2. You have a C compiler that is compatible with your MATLAB version. To see which compiler fits your MATLAB, visit `http://www.mathworks.com/support/sysreq/previous_releases.html`
3. Your MATLAB is correctly configured to build MEX files. To do this, run "mex -setup" from the MATLAB command prompt and select a compatible C compiler you would like to use to compile the code.

4.2 Installation

To install this library, first unzip and move all source codes (.m and .c files) to a desired folder(make sure you have the rights to do this), for example, /lib/MMT/. Then put the following commands in MATLAB:

```
cd /lib/MMT;
mex grouplasso_CD.c;
mex mtlasso2G_CD.c;
```

You are done and Let's begin to enjoy the library!

5 Usage

5.1 Add MMT to path

Each time you open a new MATLAB session, run the following command in MATLAB prompt to make sure the library is in the path so that you can use it.

```
addpath(genpath('/lib/MMT/'));
```

5.2 A One-Station Experience

The file "pilot.m" contains all typical uses of functions provided in the library. It first generate a synthetic dataset and then run the three multi-task learning methods to estimate association matrix B, finally the results of all three methods and the true association matrix are visualized in one figure. To try it, just run

```
pilot
```

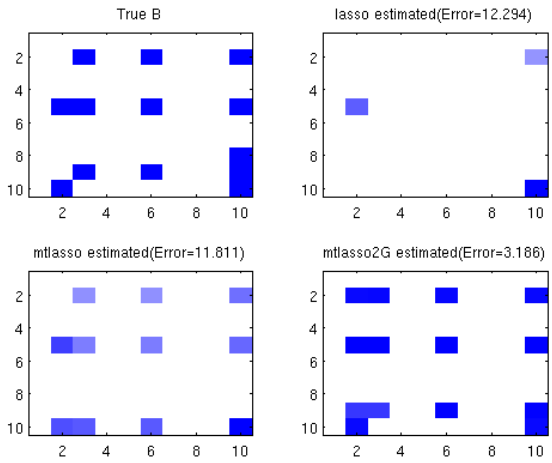

Figure 1: Figure by pilot.m

Above is a figure generated by "pilot.m". Here we begin to introduce usages of each functions.

5.3 Run lasso

The lasso method has three required input (X, Y, lambda) and two optional input (tol, max_it), and ouputs estimated association matrix (B_hat).

```
function B_hat = lasso(X, Y, lambda, tol, max_it)
```

```
% Input – X, observation matrix, size n by J
%         Y, label matrix, size n by K
%         lambda, ell_1 regularization parameter(>0)
%         tol, convergence criterion
%         max_it, maximum iteration allowed
% Output – B_hat, estimated coefficient matrix
```

5.4 Run multi-task lasso

The multi-task lasso method has the same input and output variables with lasso.

```
function B = mtlasso(X, Y, lambda, tol, max_it)
```

5.5 Run graph guided multi-task

The graph guided multi-task lasso has three required input (X, Y, lambdas) and four optional (G1, G2, tol, max_it). Different from previous two methods, graph guided multi-task has three regularization parameters, so its lambdas is a length 3 vector. Also you can use any graph through G1 and G2, both of which are a structure with G1.W(G2.W) be weights of edges, G1.C(G2.C) is correlations on edges and G1.E(G2.E) indicate which two nodes the edges connect to.

```
function B_hat = mtlasso2G(X, Y, lambdas, G1, G2, tol, max_it)
% Input – X, observation matrix, size n by J
%         Y, label/task matrix, size n by K
%         lambdas, a vector of three regularization parameters
%         G1, information for the task graph
%         G2, information for the feature graph
%         tol, convergence criterion
%         max_it, maximum iteration allowed
% Output – B_hat, estimated coefficient matrix
```

5.6 Cross-validation

A typical method to choose regularization parameters is cross-validation(CV). Here the library provides a gradient descent based CV, the GDCV method is implemented as a black box so you don't need to care the details.

```
function [lambdas history] = GDCV(F, X, Y, lambdas, k, tol, ...
    max_it)
% Input – F, function handler to evaluate, e.g. lasso, mtlasso2G
%         X, observation matrix, size n by J
%         Y, label matrix, size n by K
%         lambdas, initial values
%         k, number of folds for cross–validation
%         tol, allowed difference between two iterations, stop ...
    otherwise
%         max_it, maximum number of iterations allowed
% Output – lambdas, optimization result
%          history, trace histroy of parameters and errors
```

The last three input (k, tol, max_it) are optional. Here the lambdas can be vector of any size. history records trace of lambdas and CV errors.

5.7 Visualization

MATLAB has a spy function to visualize the sparsity pattern of a matirx, but it only show whether one position is sparse. Here the library provides a more conprehensive way to visualize a matrix.

```
function colorspy(M)
% Input – M, the matrix to visualize
```

6 Future direction

More methods for multi-task learning will be added to this library in the near future, including the popular trace norm and max norm minimization methods. Contact the author if you have any ideas, advices, or suggestions. Thanks!

7 Reference

[1] X. Chen, X. Shi, X. Xu etc, A Two Graph Guided Multi-task Lasso Approach for eQTL Mapping. Proceedings of AISTATS 2012, also Journal of Machine Learning Research Volume 22.