Dynamic Stochastic Block Models MATLAB Toolbox User Manual

Kevin S. Xu
University of Toledo
January 3, 2018

Introduction

This toolbox contains MATLAB implementations of two stochastic block models (SBMs) for analyzing dynamic network data in the form of network snapshots at discrete time. The first model, the dynamic SBM (DSBM), also referred to as the hidden Markov SBM (HM-SBM), was proposed by (Xu & Hero III, 2014) and makes a hidden Markov assumption on the edge dynamics. The second model, the stochastic block transition model (SBTM) was proposed by (Xu, 2015) and offers additional model flexibility compared to the DSBM at the cost of a larger number of parameters and slower inference procedure.

The toolbox currently includes the following:

- Implementations of the DSBM and SBTM inference procedures for both a priori (known classes) and a posteriori (estimated classes) block models.
- Demo of DSBM applied to simulated network datasets.
- Demo of DSBM and SBTM applied to dynamic email network constructed from Enron corpus (Priebe, Conroy, Marchette, & Park, 2009).

Usage

The inputs and outputs for each function are documented in the function header. For most variables, the last dimension in the array denotes the time step. Adjacency matrices are stored as $n \times n \times T$ arrays, where n denotes the number of nodes, and T denotes the number of time steps. Class memberships are stored in an $n \times T$ matrix.

Dynamic Stochastic Block Model (DSBM)

A Priori Block Models (Known Classes)

In the a priori block model setting, class memberships are known or assumed, and the objective is to estimate the time series of edge probability matrices $\Theta^{1:t}$, where θ_{ab}^{t} denotes the probability of forming an edge between nodes in classes a and b at time step t. For a priori blockmodeling, the time

series of block density matrices $Y^{1:t}$ is a sufficient statistic for estimating the edge probability matrices $\Theta^{1:t}$. Near-optimal estimates of the edge probability matrices in the maximum a posteriori (MAP) sense can be obtained using the `ekfDsbm` function, which performs extended Kalman filtering on the time series of block density matrices $Y^{1:t}$. $Y^{1:t}$ can be obtained from the time series of adjacency matrices $W^{1:t}$ and class membership vectors $\mathbf{c}^{1:t}$ using the function `calcBlockDens`.

A Posteriori Block Models (Estimated Classes)

In the a posteriori block model setting, class memberships are estimated along with the edge probability matrices. The estimates of the class memberships and edge probability matrices can be obtained using the `ekfDsbmLocalSearch` function, which alternates between applying a local search over the class memberships and applying the extended Kalman filter (EKF) on the block densities for the current class membership estimates. `ekfDsbmLocalSearch` operates directly on the time series of adjacency matrices $W^{1:t}$.

State Representation

For undirected graphs, the number of states is $p = k(k+1)/2$, where k denotes the number of classes. For directed graphs, $p = k^2$. Many of the toolbox functions involve manipulation of $k \times k$ matrices, which are symmetric for undirected graphs and asymmetric for directed graphs. We often represent these matrices by vectors of length p, which is the typical means for representing states in dynamic systems. The function `blockmat2vec` converts these quantities from matrix to vector form, while the function `blockvec2mat` does the reverse.

Setting Hyperparameters

Four hyperparameters need to be set for both the `ekfDsbm` and `ekfDsbmLocalSearch` functions:

- `initMean`: The mean of the initial state
- `initCov`: The covariance matrix of the initial state
- `obsCov`: The covariance matrix of the observation noise
- `transCov`: The covariance matrix of the process noise driving the state dynamics

The hyperparameters `initMean` and `initCov` constitute the Gaussian prior on the initial state at time 0. The first observation is generated at time 1, so it has covariance given by `initCov + transCov`. If no prior knowledge on the initial state is available, set `initMean` to `[]` and `initCov` to a diagonal matrix with large diagonal elements, e.g. 100, to minimize the effect of the prior. It is important to set `initMean` to `[]` and not choose it arbitrarily because this affects the linearization point in the EKF. If you find that your edge probability estimates are unstable, especially towards the beginning of the data, a stronger prior is probably necessary, i.e. decrease the variances in `initCov`.

The hyperparameter `obsCov` denotes the covariance of the observation noise and is a function of the edge probabilities in each block at the current time step. It can be selected automatically by using a plug-in estimate from the EKF predicted state. To use this estimate, set `obsCov` to `[]`.

The final hyperparameter `transCov` relates to the state dynamics. We have not provided a method to automatically select `transCov`. The function `generateStateCov` will generate a covariance

matrix for the state transition process noise given two parameters: the variance of the process noise and the covariance of the process noises for any two neighboring blocks. The covariances for all non-neighboring blocks is taken to be 0. As shown in the paper (Xu & Hero III, 2014), the estimation error of the EKF is not overly sensitive to the choice of `transCov` so it does not have to be chosen as carefully as the other hyperparameters; if additional fine-tuning is desired, one can check the forecasting error of the block densities using the EKF predicted states as a measure of goodness-of-fit.

Optional Inputs

Optional inputs (parameters) are specified using the struct `Opt`, which is an input to many functions. The optional inputs are specified as fields of `Opt`. Not all optional inputs are used in all functions; see the function headers for the applicable optional inputs. The list of all optional inputs, with default values, are as follows (in alphabetical order):

- `'classInit'`: Class estimates to initialize local search at initial time step $t = 1$ for a posteriori blockmodeling. If not specified (set to `[]`), spectral clustering will be performed to obtain the initial class estimates. Default: `[]`
- `'directed'`: Whether the networks are directed(set to true or false). *Default:* `true`
- `'eigShift'`: Amount to shift eigenvalues of estimated state covariance matrix if positive definiteness is lost due to numerical rounding errors. Set to [] to ignore check for positive definiteness. *Default:* `[]`
- `'embedDim'`: Dimensionality of spectral embedding to use in spectral clustering, which is typically the same as the number of classes. Default: `k`
- `'emptyVal'`: Block density value to set for empty blocks, which occur when a class has no nodes. *Default:* `0.001`
- `'innovCov'`: Covariance matrix of innovation $\mathbf{e}^t = \mathbf{y}^t - h\left(\widehat{\boldsymbol{\psi}}^{t|t-1}\right)$. This can be specified to save computation in the a posteriori setting, because the innovation does not change with respect to the class membership estimates in the local search. Set to `[]` to compute innovation covariance using the EKF. *Default:* `[]`
- `'kalmanGain'`: Kalman gain matrix. Similar to innovCov, this can be specified to save computation. Set to `[]` to compute Kalman gain using the EKF. *Default:* `[]`
- `'maxIter'`: Maximum number of iterations of local search to perform. *Default:* `100`
- `'model'`: Model index vector for state transition model and state transition covariance matrices $F^{1:t}$ and $\Gamma^{1:t}$, respectively. `model(t) = m` denotes that the mth slice of the state transition model and covariance matrices should be used at time t. *Default:* all ones vector
- `'nClasses'`: Maximum number of classes. It usually does not need to be manually specified, but is available to set the maximum number of classes larger than the number of classes in the estimated class membership vector. *Default:* `max(class)`
- `'nKmeansReps'`: Number of replicates (random initializations) of k-means clustering to use in spectral clustering. *Default:* `1`
- `'nPairs'`: Number of pairs of nodes (possible edges) in each block. This is used to estimate the covariance matrix Σ^t of the block densities, if Σ^t is not already specified (set to `[]`). If Σ^t is specified, this input is not used. *Default:* `[]`

- `'obsClip'`: How much to clip the range of block densities away from the boundaries of 0 and 1, for which the logit is undefined. *Default:* `0.001`
- `'svdType'`: Type of singular value decomposition to use for spectral clustering, either `'full'` or `'sparse'`. `'sparse'` is recommended for large networks (more than a few hundred nodes). *Default:* `'full'`
- `'output'`: Level of output to print to command window. Higher values result in more output, and `0` results in no output. *Default:* `0`
- `'stateClip'`: Magnitude to clip state estimates. Since states are logits of edge probabilities, they should not deviate too far from 0 so that edge probabilities are not too close to 0 or 1. *Default:* `10`

Stochastic Block Transition Model (SBTM)

The SBTM differs from the DSBM by assigning two edge probabilities $\theta_{ab}^{t|0}$ and $\theta_{ab}^{t|1}$ to each pair of classes a, b at each time $t \geq 2$. $\theta_{ab}^{t|0}$ denotes the probability of forming an edge at time t between a pair of nodes in classes a, b that did not have an edge at time $t-1$, while $\theta_{ab}^{t|1}$ denotes this probability for pairs of nodes that did have an edge at time $t-1$. The initial time step $t = 1$ follows a DSBM.

The SBTM can be fit in the a priori setting using the `ekfSbtm` function and in the a posteriori setting using the `ekfSbtmLocalSearch` function. Both functions operate directly on the time series of adjacency matrices $W^{1:t}$. Since each pair of classes has 2 probabilities, the SBTM state vector is twice as long as the DSBM state vector and contains the states corresponding to new edge probabilities $\theta_{ab}^{t|0}$ first followed by the states corresponding to existing edge probabilities $\theta_{ab}^{t|1}$. The hyperparameters and optional inputs for the SBTM functions are the same as in the DSBM.

Recommended Toolboxes

- Parallel Computing Toolbox: required to conduct local search over SBM classes in parallel for a posteriori blockmodeling. This toolbox is highly recommended to speed up the computation time for a posteriori blockmodel inference.
- Statistics Toolbox: required to use spectral clustering to estimate the initial class estimates. If the Statistics Toolbox is not available, substitute an alternate function for k-means clustering in `spectralClusterSbm.m`.

Description of files

- `blockmat2vec.m`: Convert $k \times k$ matrix of representation of state or observation into p-dimensional vector representation (see State Representation section for details).
- `blockvec2mat.m`: Convert p-dimensional vector representation of state or observation into $k \times k$ matrix representation (see State Representation section for details).
- `calcBlockDens.m`: Calculate block densities given time series of adjacency matrices and class memberships.

- `calcEdgeDurationsIntervals.m`: Calculate durations and intervals of edges, where the duration of an edge is the number of consecutive time snapshots in which the edge appears.
- `calcNewExistDens.m`: Calculate densities of new and existing edges in blocks. This function is used for the SBTM in the way that `calcBlockDens` is used for the DSBM.
- `calcObsCovSbtm.m`: Calculate observation covariance matrix for SBTM, which no longer follows a block structure like in the DSBM.
- `calcSbtmScaleFactors.m`: Calculate matrix of scaling factors for the SBTM used to account for new nodes and nodes changing classes over time.
- `ekfDsbm.m`: Fit dynamic stochastic block model in a priori setting with specified classes using extended Kalman filter.
- `ekfDsbmLocalSearch.m`: Fit dynamic stochastic blockmodel in a posteriori setting using extended Kalman filter and local search to estimate class memberships.
- `ekfPredict.m`: Prediction phase of extended Kalman filter.
- `ekfSbtm.m`: Fit stochastic block transition model in a priori setting with specified classes using extended Kalman filter.
- `ekfSbtmLocalSearch.m`: Fit stochastic block transition model in a posteriori setting using extended Kalman filter and local search to estimate class memberships.
- `ekfUpdate.m`: Update (correction) phase of extended Kalman filter.
- `estimateDsbmProb.m`: Estimate edge probability matrix for dynamic stochastic block model at time t.
- `estimateSbmProb.m`: Estimate edge probability matrix for (static) stochastic block model.
- `estimateSbtmProb.m`: Estimate edge probability matrices for stochastic block transition model at time t.
- `generateDsbm.m`: Generate time series of adjacency matrices according to a dynamic stochastic block model.
- `generateSbm.m`: Generate an adjacency matrix according to a static stochastic block model.
- `generateSbtm.m`: Generate time series of adjacency matrices according to a stochastic block transition model.
- `generateStateCov.m`: Generate state covariance matrix such that all state variances are identical, and all states corresponding to neighboring blocks have the same covariance.
- `localSearchDsbm.m`: Perform local search to estimate class memberships at a given time step for a dynamic stochastic block model.
- `localSearchSbm.m`: Perform local search to estimate class memberships for a (static) stochastic block model.
- `localSearchSbtm.m`: Perform local search to estimate class memberships at a given time step for a stochastic block transition model.
- `Manual.pdf`: This user manual.
- `permuteClasses.m`: Permute a class membership vector for maximum agreement with a reference class membership vector.
- `predAdjMatDsbm.m`: Forecast adjacency matrix at next time step using dynamic stochastic block model states.

- `predAdjMatSbtm.m`: Forecast adjacency matrix at next time step using stochastic block transition model states.
- `README.html`: Quick reference guide in HTML format.
- `README.md`: Quick reference guide in Github flavored Markdown format.
- `spectralClusterSbm.m`: Perform spectral clustering on adjacency matrix to estimate class membership vector for a static stochastic block model.
- `updateSbtmScaleFactorsNode.m`: Function to update the SBTM scaling factor estimates involving just a single node (rather than re-estimating the whole matrix).

Appendix: Description of additional files

Most users should not need to interact with the following files. Unless otherwise noted, all files were written by the author of this toolbox.

- `adjmat2vec.m`: Convert adjacency matrix to a vector representation. Used to evaluate link forecasting.
- `adjvec2mat.m`: Convert vector representation of adjacency matrix back to matrix representation.
- `bernrnd.m`: Generate random samples from a Bernoulli distribution.
- `cluvec2mat.m`: Convert class membership vector to binary indicator matrix form.
- `clumat2vec.m`: Convert binary class indicator matrix to class membership vector.
- `isNodeActive.m`: Check if a node is active (forms at least 1 edge) at a given time step.
- `setDefaultParam.m`: Used to set default values for optional parameters in functions.

`valid_RandIndex.m` is redistributed without modification from the MATLAB toolbox for estimating the number of clusters (Wang, 2009).

References

Priebe, C. E., Conroy, J. M., Marchette, D. J., & Park, Y. (2009). Scan statistics on Enron graphs. Retrieved from http://cis.jhu.edu/~parky/Enron/enron.html

Wang, K. (2009). (Simple) Tool for estimating the number of clusters. Retrieved from http://www.mathworks.com/matlabcentral/fileexchange/13916--simple--tool-for-estimating-the-number-of-clusters

Xu, K. S. (2015). Stochastic block transition models for dynamic networks. In *Proceedings of the 18th International Conference on Artificial Intelligence and Statistics* (pp. 1079–1087). Retrieved from http://arxiv.org/abs/1411.5404

Xu, K. S., & Hero III, A. O. (2014). Dynamic stochastic blockmodels for time-evolving social networks. *IEEE Journal of Selected Topics in Signal Processing*, *8*(4), 552–562. https://doi.org/10.1109/JSTSP.2014.2310294