STA663 Final Project

John Pura

April 30, 2015

1 Introduction

Estimating sparse precision matrices is an important problem in various statistical areas, such as principal component analysis and graphical models. For example, under Gaussian graphical models, estimating the support of the precision matrix allows one to recover the conditional dependence between components of a graph. Precision matrix estimation is generally computationally expensive, which is further exarcerbated by high-dimensional settings, particularly when the number of parameters, p, exceeds the sample size, n. Several algorithms have been developed in recent years to address these issues. The goal of this project is to implement one such algorithm - the constrained $\ell 1$ minimization estimator (CLIME), which is an efficient and accurate tool in estimating sparse precision matrices (Cai, et al., 2011).

This project implements a fast version of CLIME using the parametric simplex method (PSM) (Vanderbei, 2008; Pang, et al., 2014). A Python implementation would enhance efficiency and scalability compared to the existing `R` package `fastclime`. Additionally, the code leverages existing C code for the PSM linear programming (LP) solver to further speed up performance.

In the following report, I present a stable, well-tested version of `fastclime` in Python. Results include numerical benchmarking under simulated data and comparison to existing state-of-the-art algorithms that solve a similar family of problems in precision matrix estimation. The Python implementation of `fastclime` performs comparably to the `R` version in terms of speed when tested against a wide range of data with fixed sample size and varying number of predictors. Additionally, `fastclime` results are favorable compared to existing algorithms for precision matrix estimation. A section on ongoing/future work extends the CLIME estimation problem to the case when covariates are involved.

2 CLIME method

Let $\mathbf{x} = (x_1, ..., x_n) \in \mathbb{R}^{nxp}$ be n observations of a p-dimensional random vector $\mathbf{X} = (X_1, ..., X_p)^T$. For the $n \times p$ data matrix, $\mathbf{x}$, or its corresponding $p \times p$ sample covariance matrix, $\Sigma_n = \frac{1}{n}\sum_{k=1}^{n}(x_k - \bar{x})(x_k - \bar{x})^T$, the CLIME method solves the following optimization problem:

$$\hat{\Omega} = \arg_{\Omega} \min \|\mathbf{\Omega}\|_1 \text{ subject to } \|\mathbf{\Sigma}_n\mathbf{\Omega} - \mathbf{I}\|_\infty \leq \lambda_n, \, \mathbf{\Omega} \in \mathbb{R}^{p \times p}$$

where $\hat{\Omega}$ is the estimated precision matrix and λ_n is a tuning parameter.

This minimization problem can be further decomposed into p smaller problems, allowing us to recover the precision matrix in a column by column fashion (i.e. solving p optimization problems).

$$\hat{\omega}_i = \arg_{\omega} \min |\omega|_1 \text{ subject to } |\mathbf{\Sigma}_n\omega - \mathbf{e}_i|_\infty \leq \lambda_n, \, \omega \in \mathbb{R}^p \tag{1}$$

where $\mathbf{e}_i$ is the standard basis vector.

2.1 Parametric Simplex Method

The simplex method is a linear programming method that can be used to solve the following constrained problem:

$$\max c^T x \text{ subject to } Ax \leq b, \quad x \geq 0$$

where $A \in \mathbb{R}^{n \times d}$, $c \in \mathbb{R}^d$, and $b \in \mathbb{R}^n$.

The parametric simplex method (PSM) is an alternative formulation of the simplex method with the following rule:

$$\max(c + \lambda c^*)^T x \text{ subject to } Ax \leq b + \lambda b^*, \quad x \geq 0 \tag{2}$$

where A, b, and c are the same as above and $b^* \geq 0$ and $c^* \leq 0$ are perturbation vectors.

Here, λ is related to the tuning parameter above in the CLIME problem. The PSM algorithm performs pivots to reduce λ until the optimal solution is reached (when $\lambda = 0$). Therefore, by reformulating (1) as (3) the entire solution path for the original CLIME problem can be determined from the solution path of a single regularized LP problem using PSM. The optimal solution is achieved in only a few iterations.

2.2 CLIME Pseudocode

1. Normalize data, $\mathbf{x}$, to have zero mean and unit standard deviation along each column.
2. Estimate empirical covariance matrix, $\Sigma_n = \frac{n-1}{n}\mathbf{X}\mathbf{X}^T$, where $\mathbf{X}$ is the normalized data.
3. Initialize λ_{min} and path length size.
4. For $1 \leq i \leq p$ columns: Reformulate the CLIME problem to use PSM:

$$\hat{\omega}_i^1 \leftarrow \arg_{\omega_i} \min(\omega^+ - \omega^-) \text{ subject to } \begin{pmatrix} \Sigma_n & -\Sigma_n \\ -\Sigma_n & \Sigma_n \end{pmatrix} \begin{pmatrix} \omega^+ \\ \omega^- \end{pmatrix} \leq \begin{pmatrix} \lambda + e_i \\ \lambda - e_i \end{pmatrix}$$

where $\omega = \omega^+ - \omega^-$ and $\|\omega\|_1 = \omega^+ + \omega^-$, $\omega^+ \geq 0, \omega^- \geq 0$.
Comparing above to (1) and (2), $A = \begin{pmatrix} \Sigma_n & -\Sigma_n \\ -\Sigma_n & \Sigma_n \end{pmatrix}$, $b = \begin{pmatrix} e_i \\ -e_i \end{pmatrix}$, $c = \mathbf{-1}^T$, $b^* = \mathbf{1}^T$, and $c^* = \mathbf{0}^T$

5. Symmetrize $\hat{\Omega} = (\hat{\omega}_{ij}) = (\hat{\omega}_{ji}) \leftarrow \omega_{ij}^1 I\{|\hat{\omega}_{ij}^1| \leq |\hat{\omega}_{ji}^1|\} + \omega_{ji}^1 I\{|\hat{\omega}_{ij}^1| > |\hat{\omega}_{ji}^1|\}$

2.3 Python Implementation

Using Python/C API, I wrapped the `parametric.c` function from the `src` directory of the `fastclime R` package and imported it as its own module in Python for solving LP problems. All other functions are implemented using the `Numpy` package in Python.

I also included several functions not found in the original `fastclime R` package. Specifically, I have implemented regularization parameter selection, which is important in estimation of both precision matrices and high-dimensional undirected graphs. The regularization parameter, λ is important in controlling the sparsity of the graph. Therefore its choice is critical in maintaining valid statistical inferences regarding the conditional independence between nodes or features in the graph. I considered popular metrics, such as the Akaike Information Criterion (AIC) and Bayesian Information Criterion (BIC).

3 Numerical Results

3.1 Unit Tests

Unit tests were performed to verify correctness of the Python implementation subject to various inputs. Each function, if possible, was subjected to several inputs and tested to produce the correct value or an exception error. All tests are documented in the file `test_fastclime.py`. As seen below, all tests passed, suggesting correct functionality of `fastclime` in Python.

```
In [1]: ! py.test
```

```
============================= test session starts ==============================
platform linux2 -- Python 2.7.9 -- py-1.4.25 -- pytest-2.6.3
collected 21 items

test_fastclime.py ...

========================== 21 passed in 3.22 seconds ===========================
```

3.2 Profiling

The line profiler in Python was used to identify areas of bottleneck within the integrated functions, `fastclime_R` and `fastclime_est_select`. The profiling code is located in the file `Profiling.ipynb`. In both situations, over 90% of the runtime was due to the `parametric.mainfunc` function in the `parametric` module. This scenario is optimal as this module is purely compiled in C, which provides a fast, efficient way of running the PSM solver.

3.3 Numerical Simulations

3.3.1 Benchmarking Python and R

First I benchmarked performance times of the Python and `R` implementations of `fastclime` for a sparse random graph with varying number of predictors and fixed sample size, $n = 200$. Using the `timeit` module, the runtime was estimated for predictor sizes $p = 50, 100, 200, 400,$ and 800. For the purposes of this project, only one instance of the function was timed. As seen in Table 1 below, the times (in seconds) are comparable between Python and R. Any speedboost to the Python program would potentially require coding the entire suite of functions in C.

```
In [2]: import pandas as pd
        print "Table 1. Timing performance of fastclime implementations in Python and R in seconds"
        print pd.read_csv("benchmark.csv", sep=',', skipinitialspace=False)
```

```
Table 1. Timing performance of fastclime implementations in Python and R in seconds
        p    50   100   200    400     800
   Python  0.04  0.33  4.37  32.97  237.96
        R  0.18  0.46  4.82  33.04  241.79
```

3.3.2 Comparison with existing algorithms for precision matrix estimation

`TIGER` is a tuning insensitive graph estimation and regression procedure that is implemented in the `flare` package in `R` (original paper: Liu and Wang, 2012, R package: Li, et al., 2013). It is closely related to the SQRT-Lasso and solves the following problem:

$$\min \|X - XB\|_{2,1} + \lambda \|B\|_1 \text{ subject to } B_{jj} = 0 \tag{3}$$

where $\|\cdot\|_{2,1}$ is the $L_{2,1}$ norm.

One advantage of `TIGER` over other regularization methods, is that it does not require a model selection procedure like `fastclime` for selecting the regularization parameter. Instead the user manually selects the parameter to be $\sqrt{\frac{\log p}{n}}$, which is theoretically consistent and does not depend on unknown quantities.

To determine the efficiency of these two methods, a single run was performed for each method, for fixed $n = 100$ and varying $p = 100, 200, 300, 400$.

```
In [3]: import pandas as pd
        print "Table 2. Timing performance of fastclime and TIGER solvers in seconds"
        print pd.read_csv("tigerfc.csv", sep=',', skipinitialspace=False)
```

Table 2. Timing performance of fastclime and TIGER solvers in seconds

p	100	200	300	400
fastclime	0.72	4.70	14.20	31.06
TIGER	0.86	2.76	5.83	11.83

From the table above, it seems that `TIGER` outperforms `fastclime` in terms of efficiency. This is likely due to the slightly less complex formulation of `TIGER`. However, more runs would be needed to confirm this observation.

I then considered two simulated models of the sparse precision matrix.

The first model considers a banded precision matrix, such that $\Omega_0 = \omega^0_{ij} = \mathbb{I}\{1 \leq |i-j| \leq 10\}$.

The second model considers a random, sparse precision matrix, such that $\Omega_0 = \omega^0_{ij} = \omega^0_{ji} = 1$ for $i \neq j$ with pr. 0.05 and zero otherwise.

The BIC metric was used in `fastclime_select` to obtain the optimal regularization parameter and corresponding precision matrix. Default settings were used in `flare`, with the exception of precision, which was set to 1e-5.

Figures 1 and 2 show heatmaps of the recovered precision matrices. Black pixels represent zeros identified in the estimation. In the banded model, `fastclime` tended to recover less of the off-diagonal elements along the band. The estimate from `TIGER` also shows stronger intensities along the main diagonal, which is consistent with the true model. In the random graph, the sparsity patterns recovered by `fastclime` and `TIGER` are approximately the same as the that of the ground truth. However, caution should be taken in interpreting these graphs, as these only represent a single run for each of the solvers. A better alternative would be to examine the averaged heatmap over several runs for each solver.

Next the accuracy of `fastclime` and `TIGER` estimates with respect to the true precision matrix is quantified for several models of the true precision matrix. The random precision matrix structure is considered for fixed $n = 200$ and varying $p = 100, 200,$ and 300. Accuracy is quantified using several matrix norms: infinity (or sup), Frobenius, and L_2 norms.

```
In [4]: import pandas as pd
        df = pd.read_csv("normerrors.csv", sep=',', skipinitialspace=True)
        print "Table 3a. Absolute errors for fastclime estimate of precision matrix"
        print df.pivot(index='Norm', columns='p', values='fastclime')
        print
        print "Table 3b. Absolute errors for TIGER estimate of precision matrix"
        print df.pivot(index='Norm', columns='p', values='TIGER')
```

Table 3a. Absolute errors for fastclime estimate of precision matrix

p	100	200	300	400
Norm				
Frobenius	29.28	33.51	85.35	73.23
Infinity	23.91	23.35	50.09	31.32
L2	10.97	12.12	46.68	24.42

Table 3b. Absolute errors for TIGER estimate of precision matrix

p	100	200	300	400
Norm				
Frobenius	4.93	6.78	7.93	9.81
Infinity	2.39	3.08	2.91	3.42
L2	1.37	1.35	1.26	1.57

Overall, the normed differences between the true and estimate precision matrices were lowest for `TIGER`. The Frobenius norm yielded the largest differences, while the L_2 norm yielded the lowest. As expected, BIC was not a good metric for selecting the optimal regularization solution for $p > n$ under `fastclime`. On the other hand, errors for `TIGER` were roughly consistent across varying predictor levels. One can potentially improve on the estimation errors for the `fastclime` method through the use of a more robust metric in the regularization selection.

4 Future Work

I am currently in the process of extending CLIME to the estimation of precision matrices under covariate adjustment, also known as CAPME (Covariate-adjustmed precision matrix estimation) (Cai, et al., 2013). This will be a Python implementation of the current `R` package `capme`. The current `R` package uses different LP solvers and is generally slow. For comparison, the `fastclime` package in `R` provides results for a 200×800 matrix in under 7 minutes, while the `clime` package (used in `capme`) cannot produce results in an hour. Therefore, it would be advantageous, speed-wise, to implement `capme` in Python. So far, I have modified the `parametric.c` source code (`parametric2.c`) to leverage PSM and elements of `fastclime` and provide a dramatic speed-up compared to `R`.

4.1 CAPME problem

Consider the following regression model with covariates:

$$\mathbf{y} = \mathbf{x}\Gamma_0^T + \mathbf{z}$$

where $\mathbf{y} \in \mathbb{R}^{n \times p}$ is a collection of random vectors of responses, $\mathbf{x} \in \mathbb{R}^{n \times q}$ is a collection of random vectors of covariates or features, Γ_0 is an unknown $p \times q$ coefficient matrix, $\mathbf{z}$ is a $n \times p$ normal random vector with mean zero, covariance $\Sigma_0 \in \mathbb{R}^{p \times p}$ and precision matrix $\Omega_0 = \Sigma_0^{-1}$. Assume $\mathbf{x}$ and $\mathbf{z}$ are independent and that we have n *iid* observations $(\mathbf{x}_k, \mathbf{y}_k)$, $(k = 1, ..., n)$ for the model.

Using $\ell 1$ regularization (e.g. LASSO), we first estimate the coefficient matrix Γ_0. Then we estimate Ω_0 using CLIME above. Like before, we can estimate both Γ_0 and Ω_0 by performing the optimization on each column separately.

In order leverage the previously created `fastclime` module, both estimation stages were reformulated to the LP form used in PSM. The following pseudocode illustrates this reparametrization:

1. Normalize $\mathbf{x}$ and $\mathbf{y}$ to have zero mean and unit standard deviation along each column.
2. Compute the sample covariances $S_{xy} = \frac{n-1}{n}\mathbb{E}\mathbf{X}\mathbf{Y}^T$ and $S_{xx} = \frac{n-1}{n}\mathbb{E}\mathbf{X}\mathbf{X}^T$, where $\mathbf{X}$ and $\mathbf{Y}$ are the normalized data.
3. For $1 \leq i \leq p$ columns

   Estimate

   $$\hat{\gamma}_i \leftarrow \arg_{\gamma_i} \min |\gamma_i|_1 \text{ subject to } |S_{xy,i} - \gamma_i^T S_{xx}|_\infty \leq \lambda_n \tag{4}$$

   where $\hat{\Gamma} = (\hat{\gamma}_1, ..., \hat{\gamma}_p)^T$ This can be reformulated as follows:

   $$\hat{\gamma}_i^1 \leftarrow \arg_{\gamma_i} \min(\gamma^+ - \gamma^-) \text{ subject to } \begin{pmatrix} S_{xx} & -S_{xx} \\ -S_{xx} & S_{xx} \end{pmatrix} \begin{pmatrix} \gamma^+ \\ \gamma^- \end{pmatrix} \leq \begin{pmatrix} \lambda + S_{xy,i} \\ \lambda - S_{xy,i} \end{pmatrix}$$

   where $\gamma = \gamma^+ - \gamma^-$ and $\|\gamma\|_1 = \gamma^+ + \gamma^-$, $\gamma^+ \geq 0, \gamma^- \geq 0$. Comparing above to (2) and (4), $A = \begin{pmatrix} S_{xx} & -S_{xx} \\ -S_{xx} & S_{xx} \end{pmatrix}$, $b = \begin{pmatrix} S_{xy,i} \\ -S_{xy,i} \end{pmatrix}$, $c = \mathbf{-1}^T$, $b^* = \mathbf{1}^T$, and $c^* = \mathbf{0}^T$
4. Substitute the estimated $\hat{\Gamma}$ in (4) and compute the sample covariance, S_{yy}, substituting the column means with $\hat{\Gamma}x_k$, $1 \leq k \leq p$.
5. The optimization problem for estimating ω_i^1 for each of the p columns is then:

   $$\omega_i^1 \leftarrow \min |\omega_i|_1 \text{ subject to } |e_i - S_{yy}\omega_i|_\infty \leq \tau_n \text{ where } \tau_n \text{ is a tuning parameter.}$$

   This can be solved using PSM as in CLIME above.
6. Symmetrize the final estimator as in step 6 in the CLIME algorithm.

4.2 Additional Functionality

4.2.1 Regularization parameter selection in `fastclime` and `CAPME`

As mentioned above, the choice of regularization parameter is critical to valid statistical inference of precision matrices. In this project, AIC and BIC were considered. However, these metrics work well for the case when $n > p$, but not $n < p$. Cross-validation is another popular method, but does not perform well when $p > n$, is computationally expensive, and wastes valuable training data. An alternative method, which I plan to make available in a future version of this project, is the stability approach to regularization selection (stars), which has been shown to outperform competing state-of-the-art procedures in regularization parameter selection (Liu, et al., 2010).

4.2.2 GPU processing

To my knowledge, the current implementation of `fastclime` in Python is the fastest implementation of the CLIME algorithm. However, it can be made even faster through parallezation via GPU processing. The solution to the CLIME/CAPME problems are easily parallelizable as one only needs to solve to solution for each column in the dataset.

5 References

1. T. Cai, W. Liu and X. Luo. A constrained 1 minimization approach to sparse precision matrix estimation. J. Am. Statist. Assoc., 2011.

2. T. Cai, H. Li, W. Liu, and J. Xie. Covariate adjusted precision matrix estimation with an application in genetical genomics. Biometrika, 2011.

3. H. Liu, K. Roeer, L. Wasserman. Stability Approach to Regularization Selection (StARS) for High Dimensional Graphical Models. arXiv, 2010

4. H. Liu and L. Wang. TIGER: A Tuning-Insensitive Approach for Optimal Graph Estimation. arXiv, 2012

5. H. Pang, H. Liu, R. Vanderbei. The fastclime Package for Linear Programming and Large-Scale Precision Matrix Estimation in R, J. Machine Learning Res., 2014

6. R. Vanderbei. Linear Programming, Fundations and Extensions. Springer, 2008.

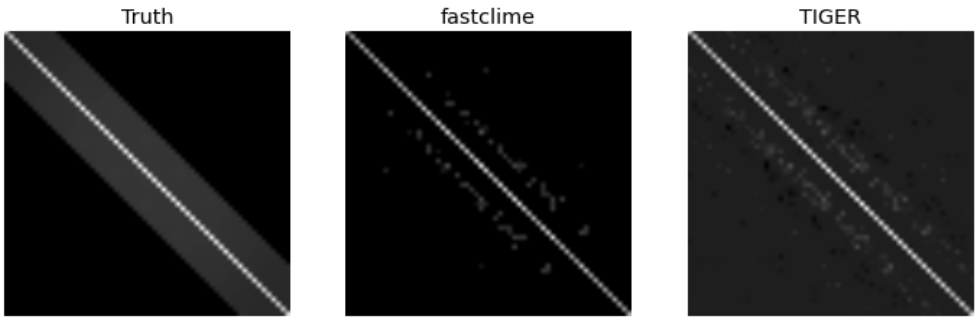

Figure 1: Simulation results for banded precision matrix, n = 200, p = 60

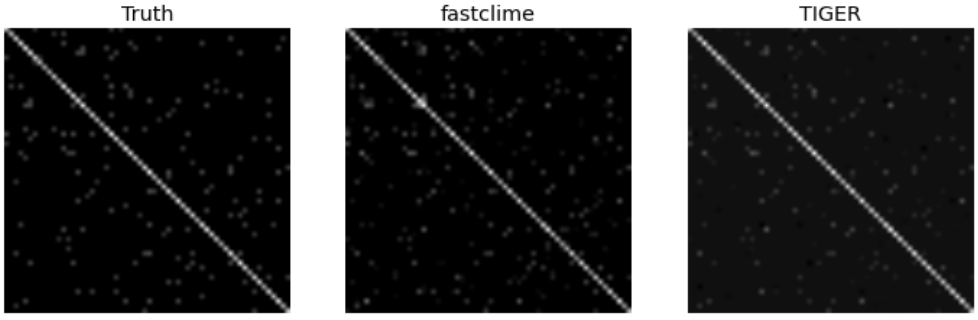

Figure 2: Simulation results for random precision matrix, n = 200, p = 60