Introduction to the GFLasso

Francisco de Abreu e Lima
Kris Sankaran

2018-07-11

This document is a revision from a GFLasso tutorial authored by Francisco de Abreu e Lima, published in DataCamp.

While the field of machine learning advances swiftly with the development of increasingly more sophisticated techniques, little attention has been given to high-dimensional multi-task problems that require the simultaneous prediction of multiple responses. This tutorial will show you the power of the Graph-Guided Fused Lasso (GFLasso) in predicting multiple responses under a single regularized linear regression framework.

Introduction

In supervised learning, one usually aims at predicting a dependent variable (*i.e.* response) from a set of explanatory variables (*i.e.* predictors) over a set of samples (*i.e.* observations). Regularization methods introduce penalties that prevent overfitting of high-dimensional data, particularly when the number of predictors exceeds the number of observations. These penalties are added to the objective function so that the coefficient estimates of uninformative predictors (that contribute little to the minimization of the error) are minimized themselves. The least absolute shrinkage and selection operator (Lasso) [1] is one such method.

What is the Lasso?

Compared to a ordinary least squares (OLS), the Lasso is capable of shrinking coefficient estimates (β) to exactly zero, thereby ruling out uninformative predictors and performing feature selection, via

$$argmin_\beta \sum_n (y_n - \hat{y_n})^2 + \lambda \sum_j |\beta_j|$$

where n and j denote any given observation and predictor, respectively. The residual sum of squares (RSS), the sole term used in OLS, can equivalently be written in the algebraic form $RSS = \sum_n (y_n - \hat{y_n})^2 = (y - X\beta)^T.(y - X\beta)$. The Lasso penalty is $\lambda \sum_j |\beta_j|$, the $L1$ norm of the coefficient estimates weighted by λ.

Why Graph-Guided Fused Lasso (GFLasso)?

What if you set to predict multiple related responses at once, from a common set of predictors? While effectively you could fare well with multiple independent Lasso models, one per response, you would be better off by coordinating those predictions with the strength of association among responses. This is because such coordination cancels out the response-specific variation that includes noise - the key strength of the GFLasso. A good example is provided in the original article, in which the authors resolve the associations between 34 genetic markers and 53 asthma traits in 543 patients [2].

What is the GFLasso?

Let $\mathbf{X}$ be a matrix of size $n \times p$, with n observations and p predictors and $\mathbf{Y}$ a matrix of size $n \times k$, with the same n observations and k responses, say, 1390 distinct electronics purchase records in 73 countries, to

predict the ratings of 50 Netflix productions over all 73 countries. Models well poised for modeling pairs of high-dimensional datasets include orthogonal two-way partial least squares (O2PLS), canonical correlation analysis (CCA) and co-inertia analysis (CIA), all of which involving matrix decomposition [3]. Additionally, since these models are based on latent variables (*i.e.* projections based on the original predictors), the computational efficiency comes at a cost of interpretability. However, this trade-off does not always pay off, and can be reverted with the direct prediction of k individual responses from selected features in $\mathbf{X}$, in a unified regression framework that takes into account the relationships among the responses. Mathematically, the GFLasso borrows the regularization of the Lasso [1] discussed above and builds the model on the graph dependency structure underlying $\mathbf{Y}$, as quantified by the $k \times k$ correlation matrix (*i.e.* the 'strength of association' aforementioned). As a result, similar (resp. dissimilar) responses will be explained by a similar (resp. dissimilar) subset of selected predictors. More formally, and following the notation used in the original manuscript [2], the objective function of the GFLasso is

$$argmin_{\beta} \sum_{k} (y_k - X\beta_k)^T.(y_k - X\beta_k) + \lambda \sum_{k} \sum_{j} |\beta_{jk}| + \gamma \sum_{(m,l) \in E} f(r_{ml}) \sum_{j} |\beta_{jm} - sign(r_{ml})\beta_{jl}|$$

where, over all k responses, $\sum_k (y_k - X\beta_k)^T.(y_k - X\beta_k)$ provides the RSS and $\lambda \sum_k \sum_j |\beta_{jk}|$ the regularization penalty borrowed from the Lasso, weighted by the parameter λ and acting on the coefficients β of every individual predictor j. The novelty of the GFLasso lies in $\gamma \sum_{(m,l) \in E} f(r_{ml}) \sum_j |\beta_{jm} - sign(r_{ml})\beta_{jl}|$, the fusion penalty weighted by γ that ensures the absolute difference between the coefficients β_{jm} and β_{jl}, from any predictor j and pair of responses m and l, will be the smaller (resp. larger) the more positive (resp. more negative) their pairwise correlation, transformed or not, $f(r_{ml})$. This fusion penalty favours globally meaningful variation in the responses over noise from each of them. When the pairwise correlation is close to zero, it does nothing, in which case you are left with a pure Lasso. This underlying correlation structure of all k responses, which can be represented as a weighted network structure, defaults to the absolute correlation, $f(r_{ml}) = |r_{ml}|$ but can be transformed to create GFLasso variants with any user-specified function, such as

1. Squared correlation, $f(r_{ml}) = r_{ml}^2$ (weighted)
2. Thresholded correlation, $f(r_{ml}) = \begin{cases} 1, & \text{if } r_{ml} > \tau \\ 0, & \text{otherwise} \end{cases}$ (unweighted)

with plenty more room for innovation. Although 2. is much less computationally intensive compared to 1. and the default absolute correlation [2], it does require a predefined cutoff, for example, $\tau = 0.8$.

To sum up, to fit a GFLasso model you will need a predictor matrix $\mathbf{X}$, a response matrix $\mathbf{Y}$ and a correlation matrix portraying the strength of association between all pairs of responses in $\mathbf{Y}$. Note that the GFLasso yields a $p \times k$ matrix of β, unlike the Lasso ($p \times 1$), and this coefficient matrix carries the associations between any given response k and predictor j.

Get started

The `gflasso` package offers reproducible n-fold cross-validation with multi-threading borrowed from the `doParallel` package, as well as visualization tools. To run GFLasso in R you will need to install `devtools`, load it and install the `gflasso` package from the GitHub repository. The demonstration will be conducted on a dataset contained in the package `bgsmtr`. We also recommend installing `corrplot` and `pheatmap` to visualize the results.

```
# Install the packages if necessary:
# install.packages("devtools")
# install.packages("bgsmtr")
# install.packages("corrplot")
# install.packages("pheatmap")
library(devtools)
```

```
## Warning: package 'devtools' was built under R version 3.4.4
```

```
library(bgsmtr)
```

```
## Warning: package 'Matrix' was built under R version 3.4.4
```

```
## Warning: package 'mvtnorm' was built under R version 3.4.4
```

```
library(corrplot)
library(pheatmap)
library(gflasso)
```

Simulation

You can easily run the simulation outlined in the help page for the CV function `cv_gflasso`. By default, the CV computes the root mean squared error (RMSE) across a single repetition of a 5-fold CV, over all possible pairs between $\lambda \in \{0, 0.1, 0.2, ..., 0.9, 1\}$ and $\gamma \in \{0, 0.1, 0.2, ..., 0.9, 1\}$, the tuning grid.

Note that user-provided error functions also work!

Besides the inherent statistical assumptions and speed performance, the choice of tuning grid ranges depends largely on mean-centering and unit-variance scaling all columns in `X` and `Y`. Note that mean-centering and scaling are not implemented in `gflasso`, so we strongly recommend centering and scaling `X` and `Y` beforehand, as necessary. We can try deriving the fusion penalty from an unweighted correlation network, with a cutoff of $r > 0.8$:

```
?cv_gflasso
set.seed(100)
X <- matrix(rnorm(100 * 10), 100, 10)
u <- matrix(rnorm(10), 10, 1)
B <- u %*% t(u) + matrix(rnorm(10 * 10, 0, 0.1), 10, 10)
Y <- X %*% B + matrix(rnorm(100 * 10), 100, 10)
R <- ifelse(cor(Y) > .8, 1, 0)
system.time(testCV <- cv_gflasso(scale(X), scale(Y), R, nCores = 1))
```

```
## [1] 0.9939133 0.6505041 0.7371684 0.7093391 0.7460661
```

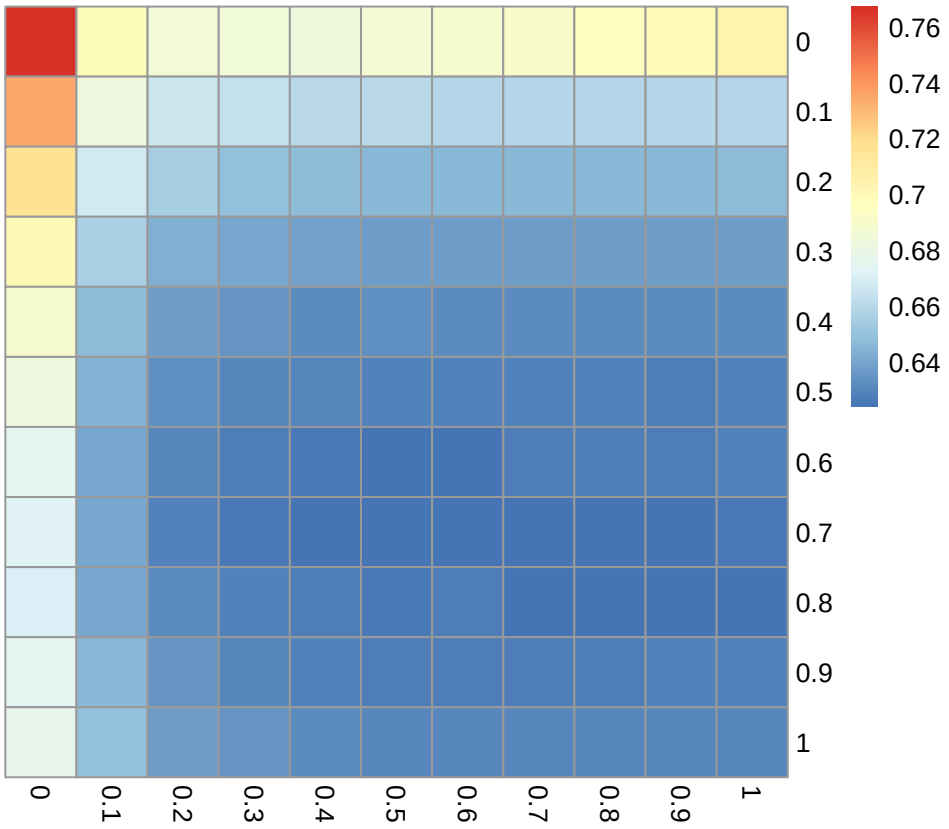
```
##    user  system elapsed
##  20.444   0.854  21.304
```

```
system.time(testCV <- cv_gflasso(scale(X), scale(Y), R, nCores = 2))
```

```
## [1] 0.9939133 0.6505041 0.7371684 0.7093391 0.7460661
```

```
##    user  system elapsed
##  10.636   0.747  11.431
```

```
cv_plot_gflasso(testCV)
```



The optimal values of λ (rows) and γ (columns) that minimize the RMSE in this simulation, 0.7 and 0.5 respectively, do capture the imposed relationships.

Tip: Try re-running this example with a different metric, the coefficient of determination (R^2). One key advantage of using R^2 is that it ranges from 0 to 1.

Keep in mind that when you provide a custom goodness-of-fit function `err_fun`, you have to define whether to maximize or minimize the resulting metric using the argument `err_opt`.

The following example aims at maximizing R^2, using a weighted association network with squared correlation coefficients (*i.e.* $f(r_{ml}) = r_{ml}^2$). If you have more than 2 cores, you might as well change the `nCores` argument and give it a boost!

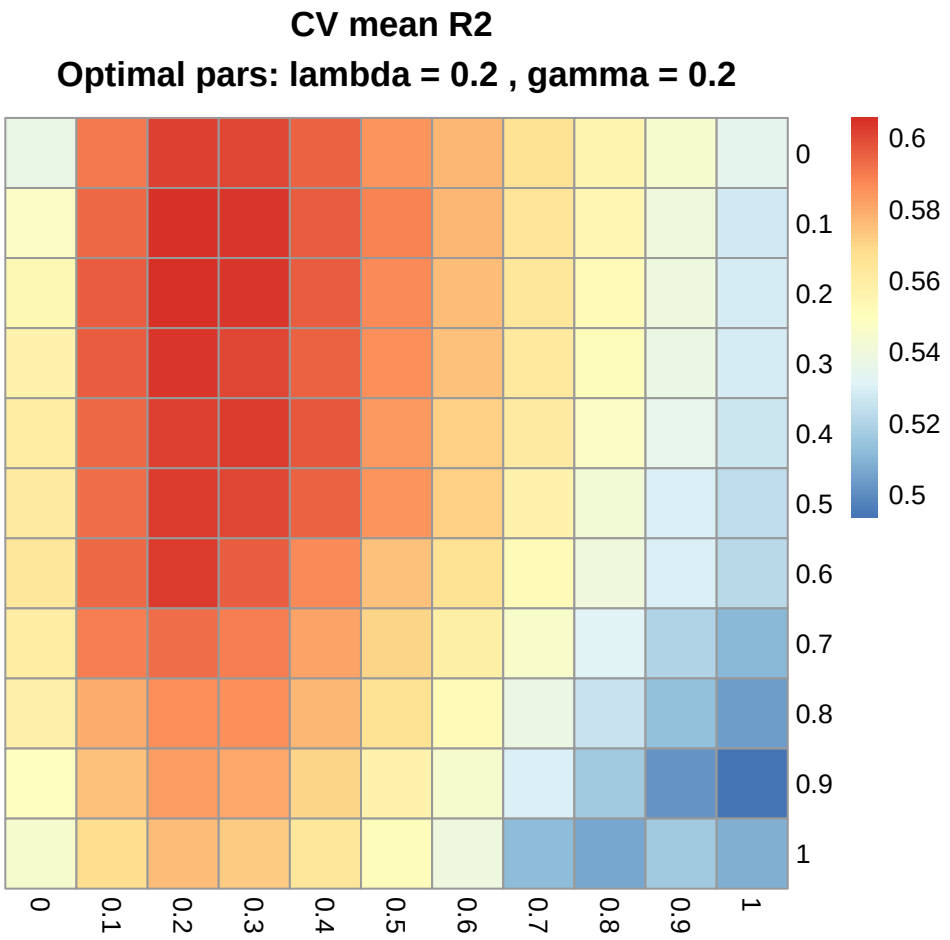
```
# Write R2 function
R2 <- function(pred, y){
      cor(as.vector(pred), as.vector(y))**2
}

# X, u, B and Y are still in memory
R <- cor(Y)**2

# Change nCores if you have more than 2, re-run CV
testCV <- cv_gflasso(scale(X), scale(Y), R, nCores = 5, err_fun = R2, err_opt = "max")
```

```
## [1] 0.4590031 0.5774414 0.5394026 0.5858527 0.5233597
```

```
cv_plot_gflasso(testCV)
```



The optimal parameters λ and γ are now 0.2 and 0.2, respectively.

Also, note that `cv_gflasso` objects comprise single lists with four elements: the mean (`$mean`) and standard error (`$SE`) of the metric over all cells of the grid, the optimal λ and γ parameters (`$optimal`) and the name of the goodness-of-fit function (`$err_fun`). The cross-validated model from the present example clearly favors both sparsity (λ) and fusion (γ).

Finally, bear in mind you can fine-tune additional parameters, such as the Nesterov's gradient convergence threshold δ and the maximum number of iterations by passing `delta_conv` and `iter_max` to `additionalOpts`, respectively. These will be used in the following example.

Determining SNP-neuroimaging associations with the GFLasso

To demonstrate the simplicity and robustness of the GFLasso in a relatively high-dimensional problem, you will next model the `bgsmtr_example_data` datasets obtained from the Alzheimer's Disease Neuroimaging Initiative (ADNI-1) database.

This is a 3-element list object, part of the `bgsmtr` package that consists of 15 structural neuromaging measures and 486 single nucleotide polymorphisms (SNPs, genetic markers) determined from a sample of 632 subjects. Importantly, the 486 SNPs cover 33 genes deemed associated with Alzheimer's disease.

Your task is to predict the morphological, neuroimaging measures from the SNP data, leveraging the

correlation structure of the former.

Let's start by organizing the data and exploring the inter-dependencies among all neuroimaging features:

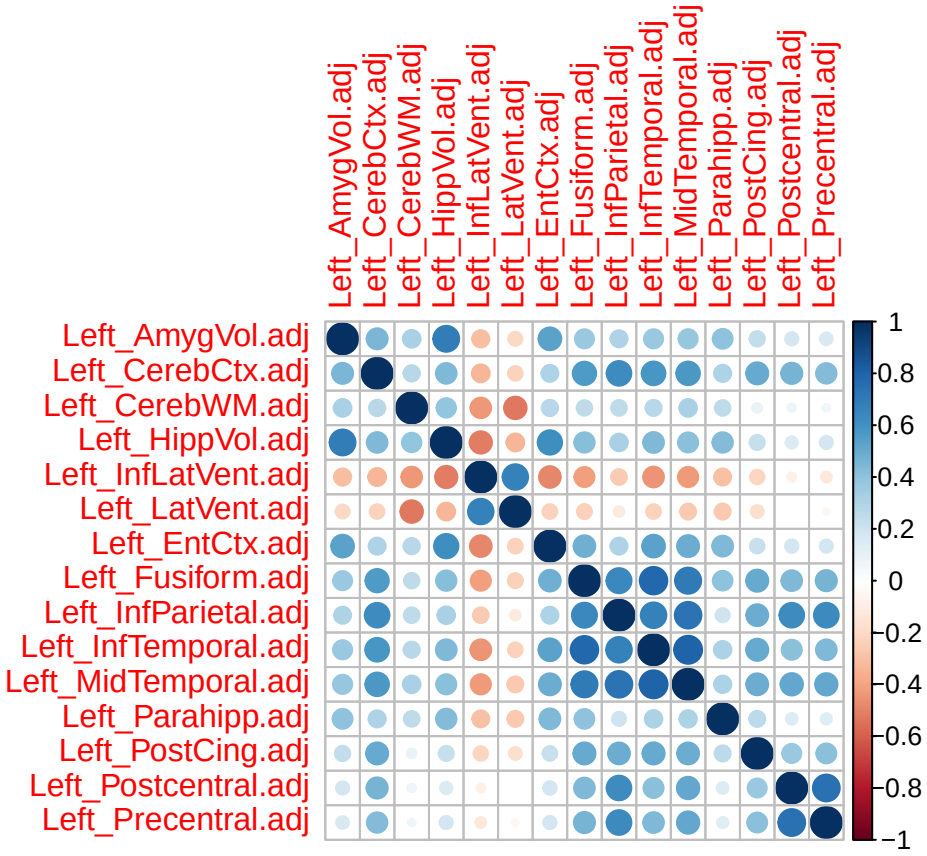
```
data(bgsmtr_example_data)
str(bgsmtr_example_data)
```

```
## List of 3
##  $ SNP_data     : int [1:486, 1:632] 2 0 2 0 0 0 0 1 0 1 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : chr [1:486] "rs4305" "rs4309" "rs4311" "rs4329" ...
##   .. ..$ : chr [1:632] "V1" "V2" "V3" "V4" ...
##  $ SNP_groups   : chr [1:486] "ACE" "ACE" "ACE" "ACE" ...
##  $ BrainMeasures: num [1:15, 1:632] 116.5 4477.9 28631.9 34.1 -473.4 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : chr [1:15] "Left_AmygVol.adj" "Left_CerebCtx.adj" "Left_CerebWM.adj" "Left_HippVol.adj" .
##   .. ..$ : chr [1:632] "V1" "V2" "V3" "V4" ...
```

```
# Transpose, so that samples are distributed as rows, predictors / responses as columns
SNP <- t(bgsmtr_example_data$SNP_data)
BM <- t(bgsmtr_example_data$BrainMeasures)

# Define dependency structure
DS <- cor(BM)

# Plot correlation matrix of the 15 neuroimaging measures
corrplot(DS)
```



The figure above points to the interdependencies among the neuroimaging features. Try now cross-validating

the GFLasso (can take up to a couple of hours in a laptop!) and determine SNP-neuroimaging associations. Generally, it is not necessary to center or scale SNP data.

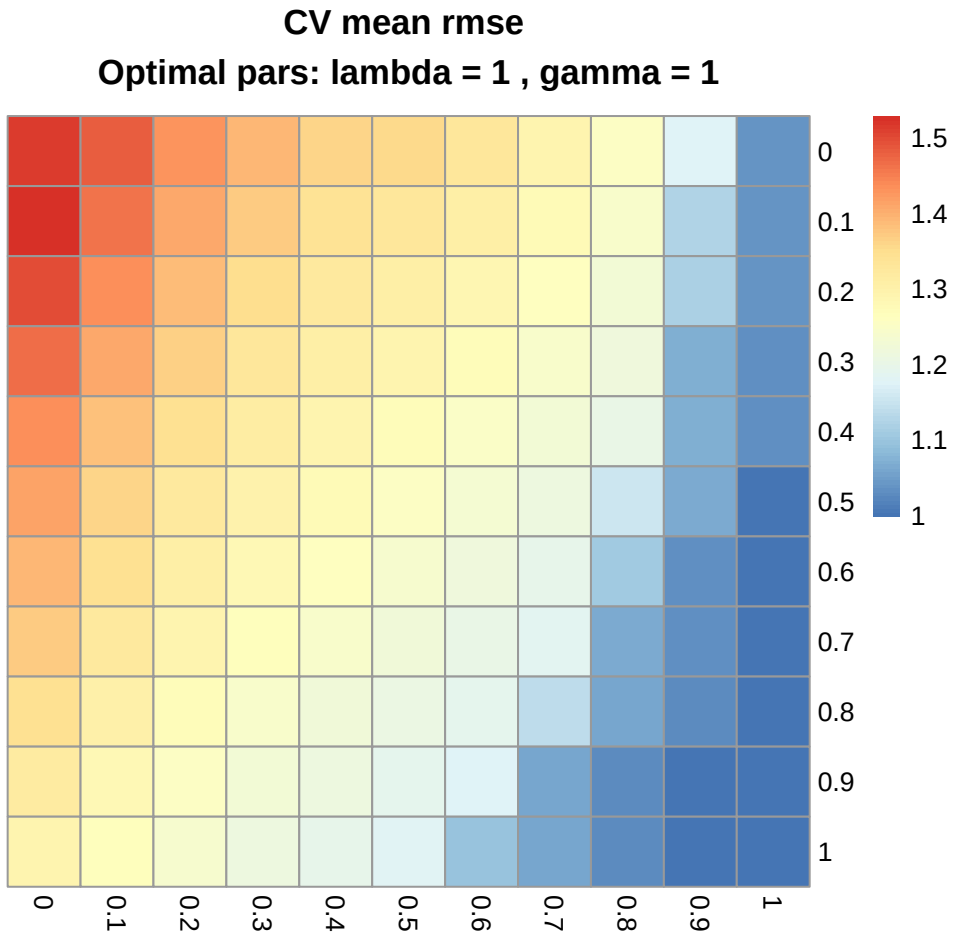
Note that in the example below, the convergence tolerance and the maximum number of iterations are specified. Feel free to try different values on your own!

```
system.time(CV <- cv_gflasso(X = SNP, Y = scale(BM), R = DS, nCores = 2,
                 additionalOpts = list(delta_conv = 1e-5, iter_max = 1e5)))
```

```
## [1] 1.500581 1.478456 1.516942 1.544581 1.527932
```

```
##     user   system  elapsed
## 1385.841  168.505 3632.138
```

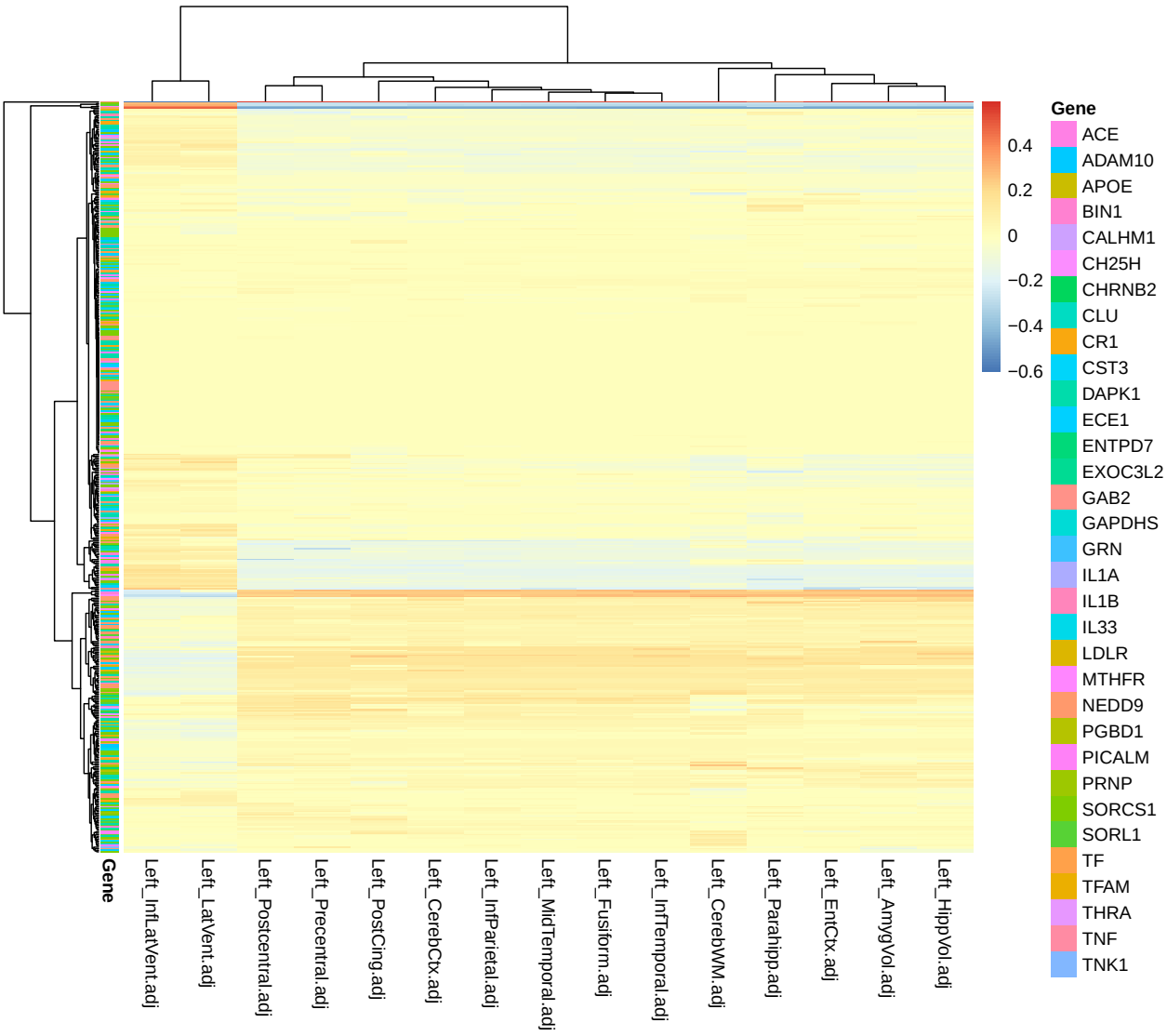
```
cv_plot_gflasso(CV)
```



By challenging the GFLasso with pure Lasso models ($\gamma = 0$, first column), a pure fusion least squares ($\lambda = 0$, first row) and OLS ($\gamma = 0$ and $\lambda = 0$, top-left cell), we can conclude the present example is best modeled with non-zero penalty weights and consequently, with the full GFLasso. Take the optimal CV parameters ($\lambda = 1$ and $\gamma = 1$) to build a GFLasso model and interpret the resulting coefficient matrix:

```
gfMod <- gflasso(X = SNP, Y = scale(BM), R = DS, opts = list(lambda = CV$optimal$lambda,
                                                             gamma = CV$optimal$gamma,
                                                             delta_conv = 1e-5,
                                                             iter_max = 1e5))
colnames(gfMod$B) <- colnames(BM)
pheatmap(gfMod$B, annotation_row = data.frame("Gene" = bgsmtr_example_data$SNP_groups,
```

```
                                        row.names = rownames(gfMod$B)),
        show_rownames = F)
```



The figure above depicts a very large proportion of coefficients with zero or near-zero values. Although there is no obvious clustering of SNPs with respect to the genes (see row-wise annotation), there are clear associations between certains SNPs and traits.

In order to ascertain the existence of a non-random predictive mechanism, you could next repeat the procedure after permutation of the values in either X or Y. Experimental work could help elucidate causality and mechanisms from selected SNPs. For example, SNPs that affect protein sequence and structure, disrupting the clearance of the β-Amyloid plaques that underlie Alzheimer's disease. For predicting on new samples, you could use the `predict_gflasso` function.

Wrap-up

The GFLasso employs both regularization and fusion when modeling multiple responses, thereby facilitating the identification of associations between predictors (X) and responses (Y). It is best used when handling high-dimensional data from very few observations, since it is much slower than contending methods. Sparse conditional Gaussian graphical models [4] and Bayesian group-sparse multi-task regression model [5], for example, might be favoured chiefly for performance gains. Nevertheless, the GFLasso is highly interpretable. The GFLasso was recently used in a omics-integrative approach to uncover new lipid genes in maize [6].

References

1. Robert Tibshirani (1994). Regression shrinkage and selection via the Lasso. *Journal of the Royal Statistical Society*, 58, 267-288.
2. Seyoung Kim, Kyung-Ah Sohn, Eric P. Xing (2009). A multivariate regression approach to association analysis of a quantitative trait network. *Bioinformatics*, 25, 12:i204–i212.
3. Chen Meng, Oana A. Zeleznik, Gerhard G. Thallinger, Bernhard Kuster, Amin M. Gholami, Aedín C. Culhane (2016). Dimension reduction techniques for the integrative analysis of multi-omics data. *Briefings in Bioinformatics*, 17, 4:628–641.
4. Lingxue Zhang, Seyoung Kim (2014). Learning Gene Networks under SNP Perturbations Using eQTL Datasets. *PLoS Comput Biol*, 10, 2:e1003420.
5. Keelin Greenlaw, Elena Szefer, Jinko Graham, Mary Lesperance, Farouk S. Nathoo (2017). A Bayesian group sparse multi-task regression model for imaging genetics. *Bioinformatics*, 33, 16:2513–2522.
6. Francisco de Abreu e Lima, Kun Li, Weiwei Wen, Jianbing Yan, Zoran Nikoloski, Lothar Willmitzer, Yariv Brotman (2018). Unraveling the lipid metabolism in maize with time-resolved multi-omics data. *The Plant Journal*, 93, 6:1102-1115.