EBglmnet Vignette

Anhui Huang and Dianting Liu

Jan. 15, 2016

Introduction

Acronyms to be used EBglmnet is a package that implements the empirical Bayesian Lasso (EBlasso) and Elastic Net (EBEN) method for generalized linear models (GLMs). Additionally, in `EBlasso`, two different prior distributions are also developed: one with two-level hierarchical Normal + Exponential prior (denoted as `NE`), and the other one with three-level Normal + Exponential + Gamma prior (denoted as `NEG`). The following names should not be confused with the `lasso` and `elastic net` method from the comparison package `glmnet`:

`EBglmnet`: package that implements `EBlasso` and `EBEN` methods.

`EBlasso`: empirical Bayesian method with `lasso` prior distribution, which includes two sets of prior distributions: `NE` and `NEG`.

`EBEN`: empirical Bayesian method with `elastic net` prior distribution.

`lasso` prior: the hierarchical prior distribution that is equivalent with `lasso` penalty term when the marginal probability distribution for the regression coefficients is considered.

`elastic net` prior: the hierarchical prior distribution that is equivalent with `elastic net` penalty term when the marginal probability distribution for the regression coefficients is considered.

`EBlasso-NE`: `EBlasso` method with `NE` prior.

`EBlasso-NEG`: `EBlasso` method with `NEG` prior.

Generalized Linear Models (GLMs) In a GLM

$$\boldsymbol{\eta} = \mu\boldsymbol{I} + \mathbf{X}\boldsymbol{\beta},$$

where $\mathbf{X}$ is an $n \times p$ matrix containing p variables for n samples (p can be $\gg n$). $\boldsymbol{\eta}$ is an $n \times 1$ linear predictor and is related to the response variable $\boldsymbol{y}$ through a link function g: $\mathrm{E}(\mathbf{y}|\mathbf{X})=g^{-1}(\mu\boldsymbol{I} + \mathbf{X}\boldsymbol{\beta})$, and $\boldsymbol{\beta}$ is a $p \times 1$ vector of regression coefficients. Depending on certain assumption of the data distribution on $\boldsymbol{y}$, the GLM is generally inferred through finding the set of model parameters that maximize the model likelihood function $p(\boldsymbol{y}|\mu, \boldsymbol{\beta}, \varphi)$, where φ denotes the other model parameters of the data distribution. However, such Maximum Likelihood (ML) approach is no longer applicable when $p \gg n$. With Bayesian lasso and Bayesian elastic net (EN) prior distribution on $\boldsymbol{\beta}$, `EBglmnet` solves the problem by inferring a sparse posterior distribution for $\hat{\boldsymbol{\beta}}$, which includes exactly zero regression coefficients for irrelevant variables and both posterior mean and variance for non-zero ones. Comparing with the `glmnet` package, not only does `EBglmnet` provide features including both sparse outcome and hypothesis testing, simulation study and real data analysis in the reference papers also demonstrates the better performance in terms of Power of Detection (PD), False Discovery Rate (FDR), as well as Power Detecting Group Effects when applicable. While mathematical details of the `EBlasso` and `EBEN` methods can be found in the reference papers, the principle of the methods and differences on the prior distributions will be briefly introduced here.

Lasso and its Bayesian Interpretation

Lasso applies a penalty term on the log likelihood function and solves for $\hat{\boldsymbol{\beta}}$ by maximizing the following penalized likelihood:

$$\hat{\boldsymbol{\beta}} = \arg_{\boldsymbol{\beta}} \max \left[\log p(\boldsymbol{y}|\mu, \boldsymbol{\beta}, \varphi) - \lambda||\boldsymbol{\beta}||_1\right].$$

The l_1 penalty term can be regarded as a mixture of hierarchical prior distribution:

$$\beta_j \sim N(0, \sigma_j^2), \sigma_j^2 \sim \exp(\lambda), j = 1, \ldots, p,$$

and maximizing the penalized likelihood function is equivalent to maximizing the marginal posterior distribution of $\boldsymbol{\beta}$:

$$\hat{\boldsymbol{\beta}} = \arg_{\boldsymbol{\beta}} \max \log p(\boldsymbol{\beta}|\boldsymbol{y}, \mathbf{X}, \mu, \lambda, \varphi) \approx \arg_{\boldsymbol{\beta}} \max \log \int \left[p(\boldsymbol{y}|\mu, \boldsymbol{\beta}, \varphi) \cdot (2\pi)^{-p/2} |\mathbf{A}|^{1/2} \exp\{-\frac{1}{2}\boldsymbol{\beta}^T \mathbf{A} \boldsymbol{\beta}\} \cdot \prod_{j=1}^{p} \lambda \exp\{-\lambda\sigma_j^2\} \right] d\boldsymbol{\sigma}^2,$$

where $\mathbf{A}$ is a diagonal matrix with $\boldsymbol{\sigma}^{-2}$ on its diagonal. Of note, `lasso` integrates out the variance information $\boldsymbol{\sigma}^2$ and estimates a posterior mode $\hat{\boldsymbol{\beta}}$. The l_1 penalty ensures that a sparse solution can be achieved. In Bayesian lasso (Park and Casella, 2008), the prior probability distribution is also conditional on the residual variance so that it has a unique mode.

Empirical Bayesian Lasso (`EBlasso`)

`EBglmnet` keeps the variance information, while still enjoying the sparse property by taking a different and slightly complicated approach as showed below using `EBlasso-NE` as an example:

In contrary to the marginalization on $\boldsymbol{\beta}$, the first step in `EBlasso-NE` is to obtain a marginal posterior distribution for $\boldsymbol{\sigma}^2$:

$$p(\boldsymbol{\sigma}^2|\boldsymbol{y},\mathbf{X},\mu,\lambda,\varphi)=\int c\left[p(\boldsymbol{y}|\mu,\boldsymbol{\beta},\varphi)\cdot(2\pi)^{-p/2}|\mathbf{A}|^{1/2}\exp\{-\frac{1}{2}\boldsymbol{\beta}^T\mathbf{A}\boldsymbol{\beta}\}\cdot\prod_{j=1}^{p}\lambda\exp\{-\lambda\sigma_j^2\}\right]d\boldsymbol{\beta},$$

where c is a constant. While the integral in `lasso` is achieved through the conjugated Normal + Exponential (NE) prior, the integral in `EBlasso-NE` is completed through mixture of two normal distributions: $p(\boldsymbol{\beta}|\boldsymbol{\sigma}^2)$ and $p(\boldsymbol{y}|\mu,\boldsymbol{\beta},\varphi)$, and the latter one is typically approximated to a normal distribution through Laplace approximation if itself is not a normal PDF. Then the estimate $\hat{\boldsymbol{\sigma}}^2$ can be obtained by maximizing this marginal posterior distribution, which has the following form:

$$\hat{\boldsymbol{\sigma}}^2=\arg_{\boldsymbol{\sigma}^2}\max\log p(\boldsymbol{\sigma}^2|\boldsymbol{y},\mathbf{X},\mu,\lambda,\varphi)=\arg_{\boldsymbol{\sigma}^2}\max\left[\log p(\boldsymbol{y}|\mu,\boldsymbol{\sigma}^2,\varphi,\lambda)-\sum_{j=1}^{p}\lambda\sigma_j^2+c\right].$$

Given the constraint that $\boldsymbol{\sigma}^2>0$, the above equation essentially maximizes the l_1 penalized marginal likelihood function of $\boldsymbol{\sigma}^2$, which images the l_1 penalty in `lasso` with the beauty of producing a sparse solution for $\hat{\boldsymbol{\sigma}}^2$. Note that if $\hat{\sigma}_j^2=0$, $\hat{\beta}_j$ will also be zero and variable x_j will be excluded from the model. Finally, with the sparse estimate of $\hat{\boldsymbol{\sigma}}^2$, the posterior estimate of $\hat{\boldsymbol{\beta}}$ and other nuance parameters can then be obtained accordingly.

Hierarchical Prior Distributions in `EBglmnet`

Prior 1: EBlasso-NE

$$\beta_j\sim N(0,\sigma_j^2),\sigma_j^2\sim\exp(\lambda),j=1,\ldots,p$$

As illustrated above, assuming a Normal + Exponential hierarchical prior distribution on $\boldsymbol{\beta}$ (`EBlasso-NE`) will yield exactly the lasso prior. `EBlasso-NE` accommodates the properties of sparse solution and hypothesis testing given both the estimated mean and variance information in $\hat{\boldsymbol{\beta}}$ and $\hat{\boldsymbol{\sigma}}^2$. The NE prior is "peak zero and flat tails", which can select variables with relatively small effect size while shrinking most of irrelevant effects to exactly zero. `EBlasso-NE` can be applied to natural population analysis when effect sizes are relatively small.

Prior 2: EBlasso-NEG The prior in `EBlasso-NE` has a relatively large probability mass on the nonzero tails, resulting in a large number of non-zero small effects with large p-values in simulation and real data analysis. We further developed another well studied conjugated hierarchical prior distribution under the empirical Bayesian framework, the Normal + Exponential + Gamma (NEG) prior:

$$\beta_j\sim N(0,\sigma_j^2),\sigma_j^2\sim\exp(\lambda),\lambda\sim gamma(a,b),j=1,\ldots,p$$

Comparing with `EBlasso-NE`, the NEG prior has a larger probability centered at 0, and will only yield nonzero regression coefficients for effects having relatively large signal to noise ratio.

Prior 3: Elastic Net Prior for Grouping Effect Similar to `lasso`, `EBlasso` typically selects one variable out of a group of correlated variables. While `elastic net` was developed to encourage a grouping effect by incorporating an l_2 penalty term, `EBglmnet` implemented an innovative `elastic net` hierarchical prior:

$$\beta_j\sim N\left[0,(\lambda_1+\tilde{\sigma}_j^{-2})^{-1}\right],\tilde{\sigma_j}^2\sim generalized\ gamma(\lambda_1,\lambda_2),j=1,\ldots,p.$$

The generalized gamma distribution has PDF: $f(\tilde{\sigma_j}^2|\lambda_1,\lambda_2)=c(\lambda_1\tilde{\sigma_j}^2+1)^{-1/2}\exp\{-\lambda_2\tilde{\sigma_j}^2\},j=1,\ldots,p$, with c being a normalization constant. The property of this prior can be appreciated from the following aspects:

(1): $\lambda_1 = 0$ When $\lambda_1 = 0$, the generalized gamma distribution becomes an exponential distribution: $f(\tilde{\sigma_j}^2|\lambda_2) = c\exp\{-\lambda_2\tilde{\sigma_j}^2\}, j = 1, \ldots, p$, with $c = \lambda_2$, and the elastic net prior is reduced to the two level `EBlasso-NE` prior.

(2): $\lambda_1 > 0$ When $\lambda_1 > 0$, the generalized gamma distribution can be written as a shift gamma distribution having the following PDF: $f(\tilde{\sigma_j}^2|a, b, \gamma) = \frac{b^a}{\Gamma(a)}(\tilde{\sigma_j}^2 - \gamma)^{a-1}\exp\{-b(\tilde{\sigma_j}^2 - \gamma)\}$, where $a = 1/2, b = \lambda_2$, and $\gamma = -1/\lambda_1$. In (Huang A. 2015), it is proved that the marginal prior distribution of β_j can be obtained as $p(\beta_j) \propto \exp\{-\frac{\lambda_1}{2}\beta_j^2 - \sqrt{2\lambda_2}|\beta_j|\}$, which is equivalent to the `elastic net` method in `glmnet`.

(3): structure of $\boldsymbol{\sigma}^2$ and interpretation of the `elastic net` prior Note that the prior variance for the regression coefficients has this form: $\boldsymbol{\sigma}^2 = \tilde{\boldsymbol{\sigma}}^2/(\lambda_1\tilde{\boldsymbol{\sigma}}^2 + \boldsymbol{I})$. This structure seems counter-intuitive at first glance. However, if we look at it from precision point of view, i.e., $\boldsymbol{\alpha} = \boldsymbol{\sigma}^{-2}$, and $\tilde{\boldsymbol{\alpha}} = \tilde{\boldsymbol{\sigma}}^{-2}$, then we have:

$$\boldsymbol{\alpha} = \lambda_1\boldsymbol{I} + \tilde{\boldsymbol{\alpha}}.$$

The above equation demonstrates that we actually decompose the precision of the normal prior into a fixed component λ_1 shared by all explanatory variables and a random component $\tilde{\boldsymbol{\alpha}}$ that is unique for each explanatory variable. This design represents the mathematical balance between the inter-group independence and intra-group correlation among explanatory variables, and is aligned with the objective of sparseness while encouraging grouping effects.

The empirical Bayesian elastic net (EBEN) in `EBglmnet` is solved similarly as `EBlasso` using the aforementioned empirical Bayesian approach. Research studies presented in the reference papers demonstrated that `EBEN` has better performance comparing with `elastic net`, in terms of PD, FDR, and most importantly, Power of Detecting Groups.

`EBglmnet` Implementation and Usage

The `EBglmnet` algorithms use greedy coordinate descent, which successively optimizes the objective function over each parameter with others fixed, and cycles repeatedly until convergence. Key algorithms are implemented in C/C++ with matrix computation using the BLAS/LAPACK packages. Due to closed form solutions for $\hat{\boldsymbol{\sigma}}^2$ in all prior setups and other algorithmic and programming techniques, the algorithms can compute the solutions very fast.

We recommend using `EBlasso-NEG` when there are a large number of candidate effects (eg., $\geq 10^6$ number of effects such as whole-genome epistasis analysis and GWAS), and using `EBEN` when groups of highly correlated variables are of interest.

The authors of `EBglmnet` are Anhui Huang and Dianting Liu. This vignette describes the principle and usage of `EBglmnet` in R. Users are referred to the papers in the Reference section for details of the algorithms.

Installation

`EBglmnet` can be installed directly from CRAN using the following command in R console:

```
install.packages("EBglmnet", repos = "http://cran.us.r-project.org")
```

which will download and install the package to the default directory. Alternatively, users can download the pre-compiled binary file at http://cran.r-project.org/web/packages/EBglmnet/index.html, and install it from local package.

Back to Top

Quick Start

We will give users a general idea of the package by using a simple example that demonstrates the basis package usage. Through running the main functions and examining the outputs, users may have a better idea on how the package works, what functions are available, which parameters to choose, as well as where to seek help. More details are given in later sections.

Let us first clear up the workspace and load the `EBglmnet` package:

```
rm(list = ls())
library(EBglmnet)
```

We will use an R built-in dataset `state.x77` as an example, which includes a matrix with 50 rows and 8 columns giving the following measurements in the respective columns: Population, Income, Illiteracy, Life Expectancy, Murder Rate, High School Graduate Rate, Days Below Freezing Temperature, and Land Area. The default model used in the package is the Gaussian linear model, and we will demonstrate it using Life Expectancy as the response variable and the remaining as explanatory variables. We create the input data as showed below, and users can load their own data and prepare variable `y` and `x` following this example.

```
varNames = colnames(state.x77);
varNames
```

```
## [1] "Population" "Income"     "Illiteracy" "Life Exp"   "Murder"
## [6] "HS Grad"    "Frost"      "Area"
```

```
y= state.x77[,"Life Exp"]
xNames = c("Population","Income","Illiteracy", "Murder","HS Grad","Frost","Area")
x = state.x77[,xNames]
```

We fit the model using the most basic call to `EBglmnet` with default prior

```
set.seed(1)
output = EBglmnet(x,y,hyperparameters = c(0.1, 0.1))
```

"output" is a list containing all the relevant information of the fitted model. Users can examine the output by directly looking at each element in the list. Particularly, the sparse regression coefficients can be extracted as showed below:

```
glmfit = output$fit
variables = xNames[glmfit[,1,drop=FALSE]]
cbind(variables,as.data.frame(round(glmfit[,3:6,drop=FALSE],4)))
```

```
##   variables    beta posterior variance t-value p-value
## 1    Murder -0.2716              2e-04 19.1011       0
```

The hyperparameters in each of the prior distributions control the number of non-zero effects to be selected, and cross-validation (CV) is perhaps the simplest and most widely used method in deciding their values. `cv.EBglmnet` is the main function to do cross-validation, which can be called using the following code.

```
cvfit = cv.EBglmnet(x, y)
```

```
## EBLASSO Linear Model, NEG prior,Epis:  FALSE ; 5 fold cross-validation
```

`cv.EBglmnet` returns a `cv.EBglmnet` object, which is a list with all the ingredients of CV and the final fit results using CV selected hyperparameters. We can view the CV results, selected hyperparameters and the corresponding coefficients. For example, result using different hyperparameters and the corresponding prediction errors are shown below:

```
cvfit$CrossValidation
```

```
##           a    b Mean Square Error standard error
##  [1,]  0.01 0.01         0.7930915      0.7329892
##  [2,]  0.05 0.05         0.8673170      0.7273493
##  [3,]  0.10 0.10         0.8858297      0.6849729
##  [4,]  0.50 0.50         0.8787371      0.5705269
##  [5,]  1.00 1.00         1.3098721      0.7438981
##  [6,] -0.01 0.01         0.7930823      0.7337260
##  [7,] -0.10 0.01         0.7205029      0.4003129
##  [8,] -0.20 0.01         0.8058868      0.4886126
##  [9,] -0.30 0.01         0.9844774      0.8831515
## [10,] -0.40 0.01         0.8247224      0.5932214
## [11,] -0.50 0.01         0.8603200      0.5962155
## [12,] -0.60 0.01         0.8672803      0.6027390
## [13,] -0.70 0.01         0.9110442      0.6710180
## [14,] -0.80 0.01         0.9106710      0.6728358
## [15,] -0.90 0.01         0.9108844      0.6773827
## [16,] -0.10 0.05         0.8714122      0.5887262
## [17,] -0.10 0.10         0.9880775      0.7285293
## [18,] -0.10 0.50         0.9132672      0.5987704
## [19,] -0.10 1.00         0.9822634      0.6155989
```

The selected parameters and the corresponding fitting results are:

```
cvfit$hyperparameters
```

```
##     a     b
## -0.10  0.01
```

```
cvfit$fit
```

```
##      locus1 locus2       beta posterior variance   t-value      p-value
## [1,]      4      4 -0.2598186       6.244212e-04 10.397560 5.417888e-14
## [2,]      5      5  0.0129420       1.417103e-05  3.437959 1.204829e-03
```

Back to Top

GLM Family

Two families of models have been developed in `EBglmnet`, the `gaussian` family and the `binomial` family, which are essentially different probability distribution assumptions on the response variable `y`.

Gaussian Model

`EBglmnet` assumes a Gaussian distribution on `y` by default, i.e., $p(\boldsymbol{y}|\mu, \boldsymbol{\beta}, \varphi) = N(\mu \boldsymbol{I} + \mathbf{X}\boldsymbol{\beta}, \sigma_0^2 \boldsymbol{I})$, where $\varphi = \sigma_0^2$ is the residual variance. In the above example, both $\hat{\mu}$ and $\hat{\sigma_0}^2$ are listed in the output:

```
output$Intercept
```

```
## [1] 72.88376
```

```
output$residual
```

```
## [1] 0.6912821
```

Binomial Model

If there are two possible outcomes in the response variable, a binomial distribution assumption on `y` is available in `EBglmnet`, which has $p(\boldsymbol{y}|\mu, \boldsymbol{\beta}, \varphi)$ following a binomial distribution and $\varphi \in \emptyset$. Same as the widely-used logistic regression model, the link function is $\eta_i = logit(p_i) = \log(\frac{Pr(y_i)=1}{1-Pr(y_i=1)}), i = 1, \ldots, n$. To run `EBglmnet` with binomial models, users need to specify the parameter `family` as `binomial`:

```
yy = y>mean(y);
output = EBglmnet(x,yy,family="binomial", hyperparameters = c(0.1, 0.1))
```

For illustration purpose, the above codes created a binary variable `yy` by set the cutoff at the mean Life Expectancy value.

Back to Top

Prior, Hyperparameters and Epistasis

The three sets of hierarchical prior distribution can be specified by `prior` option in `EBglmnet`. By default, `EBglmnet` assumes the `lassoNEG` prior. The following example changes the prior via `prior` parameter:

```
output = EBglmnet(x,yy,family="binomial", prior = "elastic net", hyperparameters = c(0.1, 0.1))
```

Note that the hyperparameters setup is associated with a specific prior. In `lasso` prior, only one hyperparameter λ is required, while in `elastic net` and `lassoNEG`, two hyperparameters need to be specified. For `EBEN` having the `elastic net` prior distribution, the two hyperparameters λ_1 and λ_2 are defined in terms of other two parameters $\alpha \in [0, 1]$ and $\lambda > 0$ same as in `glmnet` package, such that $\lambda_1 = (1 - \alpha)\lambda$ and $\lambda_2 = \alpha\lambda$. Therefore, users are asked to specify $hyperparameters = c(\alpha, \lambda)$.

In genetic and population analysis, sometimes it is interested in analyzing the interaction terms among the variables (epistasis). `EBglmnet` provides a feature that can incorporate all pair-wise interactions into analysis, which is achieved by setting `Epis` as `TRUE`:

```
output = EBglmnet(x,yy,family="binomial", prior = "elastic net",
                  hyperparameters = c(0.1, 0.1),Epis = TRUE)
output$fit
```

```
##      locus1 locus2          beta posterior variance   t-value    p-value
## [1,]      4      4 -5.318341e-02       1.491454e-03 1.3771184 0.17473457
## [2,]      1      7  1.719555e-10       5.184252e-20 0.7552192 0.45373244
## [3,]      2      4 -8.894085e-06       6.408661e-11 1.1110091 0.27198645
## [4,]      3      4 -3.785224e-02       4.632023e-04 1.7587586 0.08486232
## [5,]      4      5 -3.100472e-04       2.447304e-07 0.6267348 0.53374233
## [6,]      4      6 -4.688616e-04       1.273471e-07 1.3138633 0.19501041
```

When `Epis = TRUE`, both p number of main effects and $p(p-1)/2$ number of interaction effects are considered in the model. In the output, `locus1` and `locus2` denote the pair of interaction variables, and if the numbers are the same, the corresponding effect is from a main effect. Users should be aware of the significant larger number variables (i.e., $p(p-1)/2$ more variables), thus a longer time to finish the computation.

Back to Top

Example of p>n Data

Gaussian Model with Main Effects

We will demontrate the application of EBglmnet in multiple QTL mapping using a simulated F2 population, which is available along with EBglmnet package. The genotype of the F2 population is simulated from cross of two inbred lines. A total of $p = 481$ gentic markers were simulated to be evenly spaced on a large chromosome of 2400 centi-Morgan (cM) with an interval of d = 5 cM. Theoretically, two adjacent markers have a correlation coefficient $R = e^{-2d} = 0.9048$ since the Haldane map function is assumed. The dummy variable for the three genotypes, AA, Aa and aa of individual i at marker j was defined as $x_{ij} = 1, 0, -1$, respectively.

```
data(BASIS)#this is the genotype of the the F2 population
N = nrow(BASIS)
p = ncol(BASIS)
j = sample((p-2),1)
cor(BASIS[,c(j,j+1,j+2)]) #Correlation structure among neighboring markers
```

```
##           m229      m230      m231
## m229 1.0000000 0.9038431 0.8143151
## m230 0.9038431 1.0000000 0.9023130
## m231 0.8143151 0.9023130 1.0000000
```

Let us first simulate a quantitative phenotype with population mean 100 and residual variance that contribute to 10% of population variance. Ten QTLs were simulated from the 481 markers, and effect sizes are randomly generated from [2,3]. We assume that QTLs were coincided with markers. If QTLs were not on markers, they may still be detected given the above correlation structure, although a slightly larger sample size may be needed to give the same PD:

```
set.seed(1);
Mu = 100; #population mean;
nTrue = 10; # we assume 10 out of the 481 effects are true QTLs
trueLoc = sort(sample(p,nTrue));
trueEff = runif(nTrue,2,3); #effect size from 2-3
xbeta = BASIS[,trueLoc]%*%trueEff;
s2 =  var(xbeta)*0.1/0.9 #residual variance with 10% noise
residual = rnorm(N,mean=0,sd=sqrt(s2))
y = Mu + xbeta + residual;
```

To demonstrate the performance of EBglmnet in analyzing dataset with $p > n$, we will analyze the simulated dataset using a smaller sample size $n = 300$:

```
n = 300;
index = sample(N,n);
CV = cv.EBglmnet(x=BASIS[index,],y=y[index],family="gaussian",prior= "lassoNEG",nfold= 5)
```

```
## EBLASSO Linear Model, NEG prior,Epis:  FALSE ; 5 fold cross-validation
```

With 5 fold CV, EBlasso-NEG identified the following effects:

```
CV$fit
```

```
##       locus1 locus2     beta posterior variance  t-value p-value
##  [1,]     30     30 2.527794         0.03160368 14.21912       0
##  [2,]     97     97 2.347208         0.03473698 12.59376       0
##  [3,]    128    128 2.466978         0.03685331 12.85071       0
##  [4,]    178    178 2.461466         0.03379284 13.39003       0
##  [5,]    275    275 2.389084         0.03847676 12.17958       0
##  [6,]    298    298 2.434124         0.03277078 13.44619       0
##  [7,]    314    314 2.340964         0.03717382 12.14161       0
##  [8,]    428    428 2.737366         0.04559953 12.81896       0
##  [9,]    435    435 2.316184         0.05005450 10.35265       0
## [10,]    449    449 3.070134         0.03764603 15.82332       0
```

```
trueLoc
```

```
##  [1]  30  97 128 179 275 298 314 428 435 449
```

Comparing with the true QTL locations, EBlasso-NEG successfully identified all QTLs. Of note, an identified marker that is <20cM from a true QTL is generally not considered as a false effect, given the small distance and high genotype correlation. Since there is limited prior information in this simulation, other methods in EBglmnet will yield similar results.

Binomial Model and Separation Problem

In many genetics and population analysis such as binary QTL mapping or GWAS, both genetic effects (eg., $\mathbf{X}$ takes values of 1, 0, -1 denoting three genotype values AA, Aa, aa) and response variable (eg., $\boldsymbol{y}$ takes values of 0 and 1 denoting the two phenotype classes) are discrete values. Separation problem (complete separation and semi-complete separation) occurs often in such data, especially when epistasis is considered due to the larger number of variables $p' = p(p+1)/2$. Of note, separation is a problem in logistic regression where there exist some coefficients $\boldsymbol{\beta}$ such that $y_i = 1$ whenever $\mathbf{x}_i^T\boldsymbol{\beta} > 0$, and $y_i = 0$ whenever $\mathbf{x}_i^T\boldsymbol{\beta} < 0, i = 1, \ldots, n$. Unless the phenotype is a Mendelian trait, finding a genetic factor/set of factors that perfectly predict the phenotype outcome is too good to be true. While separation problem has been well documented in many statiscal textbook (eg., Ch9 of Altman M, Gill J, Mcdonald M P. (2005)), it is less studied in high dimensional sparse modeling methods. We next examine the problem using lasso as an example.

Benchmarking using EBglmnet In the simulated F2 population, if we consider both main and epistatic effects, there will be a total number $p' = 115,921$ candidate effects. We will randomly select 10 main and 10 interaction effects as true effects. we will use sample size $n = 300$, and also randomly generate effect sizes from unif[2,3]. Note that an epistatic effect is generated by dot product of two interacting main effects:

```
n = 300;
set.seed(1)
index = sample(nrow(BASIS),n)
p = ncol(BASIS);
m = p*(p+1)/2;
#1. simulate true QTL locations
nMain = 10;
nEpis  = 10;
mainLoc = sample(p,nMain);
episLoc = sample(seq((p+1),m,1),nEpis);
trueLoc = sort(c(mainLoc,episLoc)); #a vector in [1,m]
nTrue = length(trueLoc);
trueLocs = ijIndex(trueLoc, p); #two columns denoting the pair (i,j)
#2. obtain true QTL genotype
basis = matrix(0,n,nTrue);
for(i in 1:nTrue)
{
    if(trueLocs[i,1]==trueLocs[i,2])
    {
        basis[,i] = BASIS[index,trueLocs[i,1]]
    }else
    {
        basis[,i] = BASIS[index,trueLocs[i,1]]*BASIS[index,trueLocs[i,2]]
    }
}
#3. simulate true QTL effect size
trueEff  = runif(nTrue,2,3);
#4. simulate phenotype
xbeta = basis%*%trueEff;
vary = var(xbeta);
Pr = 1/(1+ exp( -xbeta));
y = rbinom(n,1,Pr);
```

Now we have the phenotype simulated. Let us demonstrate the data analysis using EBlasso-NE as an example. Given the significant larger number of candidate effects (200 times more effects), this will take a longer time for CV to finish ($n_{folds} \times n_{hyperparameters} + 1$) times of calling EBglmnet algorithm. In fact, the computational time is mostly determined by the number of nonzero effects selected by the model. This can be seen if a larger sample size is used (note: if you set $n = 1000$, this will take several more hours to finish the ($n_{folds} \times n_{hyperparameters} + 1$) times computation (Altough a higher PD will be obtained).

```
CV = cv.EBglmnet(x=BASIS[index,],y=y,family="binomial",prior="lasso",nfold=5,Epis =TRUE)
ind = which(CV$fit[,6]<=0.1)#p-value cutoff
CV$fit[ind,]
```

By comparing `trueLocs` and the effects identified by EBglmnet, it is demonstrated that EBlasso-NE has PD =0.50, FDR =0.09 using p-value cutoff = 0.10. As discussed earlier, lasso prior assigns a large probability mass in the two tails, resulting in a large number of small effects with large p-values. Without filtering using $p - value$, EBlasso-NE has PD = 0.85 and FDR = 0.65.

Separation Problem Compared with `EBglmnet`, `glmnet` does not have a reasonable result in analyzing this dataset, partially because of the separation problem. Let us first show the result using lasso approach.

Since glmnet has no built-in facility for epistasis analysis, we will manually create a genotype matrix $\mathbf{X}$ containing all main and epistasis effects.

```
X = matrix(0,n,m);
X[,1:p] = BASIS[index,];
kk = p + 1;
for(i in 1:(p-1))
{
    for(j in (i+1):p)
    {
        X[,kk] = BASIS[index,i] * BASIS[index,j];
        kk = kk + 1;
    }
}
```

Let us analyze the dataset using lasso, examine the lasso selection path:

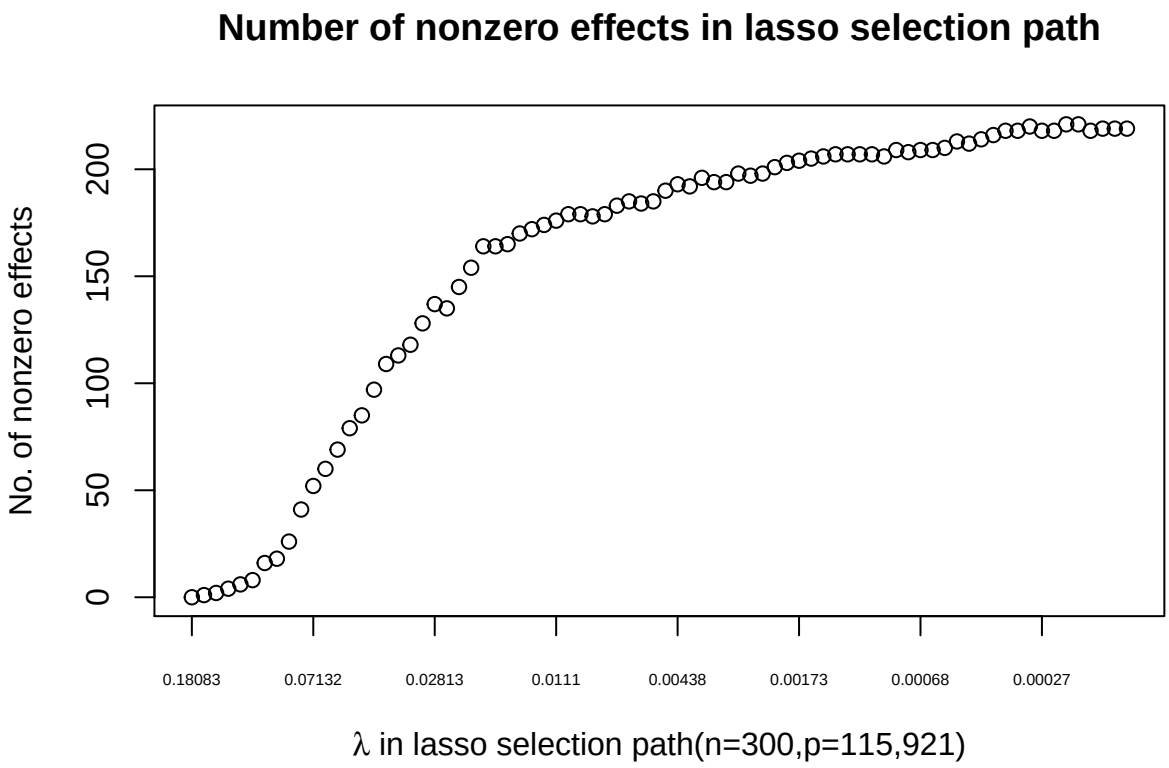
```
library(glmnet);
```

```
## Loading required package: Matrix
## Loading required package: foreach
## Loaded glmnet 2.0-2
```

```
alpha = 1
lambdaRatio = 1e-4; #same as in EBlasso
cv = cv.glmnet(X, y, alpha = alpha,family="binomial",nfolds = 5,lambda.min.ratio=lambdaRatio)
nLambda = length(cv$lambda)
nLambda
```

```
## [1] 78
```

```
nbeta = rep(0,nLambda);
fit0 = cv$glmnet.fit;
for(i in 1:nLambda)
{
  nbeta[i] = length(which(fit0$beta[,i]!=0))
}
plot(nbeta,xlab=expression(paste(lambda, " in lasso selection path(n=300,p=115,921)")),
     ylab="No. of nonzero effects",xaxt="n")#
ticks = seq(1,nLambda,10)
axis(side=1, at= ticks,labels=round(cv$lambda[ticks],5), las=1,cex.axis = 0.5)
title("Number of nonzero effects in lasso selection path")
```



The result above demonstrated that lasso was not able to complete the selection path, and exited at the 78 out of 100 candidate λs. From the scatterplot, it is shown that the number of nonzero effects has stablized after the 25th λ due to semi-complete separation that fewer candidate variables can be selected, even lambda is decreasing exponentially (will be explained in lasso discarding rule in the ensuing section). See the `glmnet` user manual for the built-in exit mechanism.

We can also take a closer look by re-fitting the lasso selected effects in an ordinary logistic regression model, which will explicitly print out the warning message of separation:

```
lambda= cv$lambda.min
coefs = fit0$beta
ind = which(cv$lambda==cv$lambda.min)
beta = coefs[,ind]
betaij = which(beta!=0)
Xdata = X[,betaij];
colnames(Xdata) = betaij;
refit = glm(y ~ Xdata, family = binomial(link = "logit"))#separation occurs
```

```
## Warning: glm.fit: algorithm did not converge
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

The warning message describes that separation occurs. Separation problem is detrimental to lasso/elastic net due to the discarding rules (Tibshirani et al., 2012). Of note, the lasso discarding rule for logistic regression states that variable $\boldsymbol{x}_j$ can be discarded if:

$$|\boldsymbol{x}_j^T(\boldsymbol{y} - \boldsymbol{p}(\hat{\boldsymbol{\beta}}_{\lambda_{k-1}}))| < 2\lambda - \lambda_{k-1}, \forall j$$

where $\hat{\boldsymbol{\beta}}_{\lambda_{k-1}}$ is the nonzero coefficients found by lasso using λ_{k-1} at the $k-1$ step. Suppose with λ_{k-1}, lasso selected a set of variables $\tilde{\boldsymbol{X}}$ that perfectly separate $\boldsymbol{y}$, which lead to $\boldsymbol{y} - \boldsymbol{p}(\hat{\boldsymbol{\beta}}_{\lambda_{k-1}}) \approx 0$. Then, the above discarding rule will have most of the remaining variables discarded. Note that $\boldsymbol{y} - \boldsymbol{p}(\hat{\boldsymbol{\beta}}_{\lambda_{k-1}})$ will not be exactly 0 due to numerical estimations, and as λ decreases exponentially, a few variables can still pass the discarding rule (shown from 25th - 78th λ in the above example). However, finding a genetic factor/set of factors that perfectly predict the phenotype outcome is unlikely, and it is the case given the simulation setup. With the perfect separation provided by $\tilde{\boldsymbol{X}}$, other true effects cannot be selected into the zon-zero set, resulting in limited PD.

EBglmnet doesn't implement such type of discarding rule and is more numerically stable. In the above simulation, EBglmnet can still identify several true effects with reasonable FDR. More simulation results using EBglmnet are available in (Huang et al., 2013) and the EBglmnet Application Note.

Parallel Cross Validation

Although `EBglmnet` is efficient in inferring model parameters with given hyperparameters, there is no efficient selection path and discarding rules implemented. Additionally, when epistasis is considered, `EBglmnet` allocates more computational resources to dynamically create the interaction variables so as to get around of memory shortage problem (by not creating the giant input matrix of $n \times p'$ and other temporary variables). Evaluating $(n_{folds} \times n_{hyperparameters} + 1)$ times in serial can take many hours for a large dataset. Alternatively, users can take advantage of parallel computational resources provided by R to reduce the computational time. In the following example, the `snow` package will be used to demonstrate parallel CV.

```
#1. create the hyperparameters to be evaluated
familyPara = "gaussian";
priorPara = "elastic net";
epis =TRUE;
lambda_Max = lambdaMax(BASIS[index,],y,epis);
nStep = 9;#steps from lambda_max to 0.0001*lambda_min
lambda_Min = 0.0001*lambda_Max;
step = (log(lambda_Max) - log(lambda_Min))/nStep;
Lambda  = exp(seq(from = log(lambda_Max),to=log(lambda_Min),by= -step))
N_step  = length(Lambda);
Alpha = seq(from = 1, to = 0.1, by = -0.1)# values of alpha
nAlpha = length(Alpha);
nPara = nAlpha * N_step;
HyperPara = matrix(0,nPara,2);
for(j in 1:nAlpha)
{
    strt = (j-1)*N_step + 1;
    ed  = (j-1)*N_step + N_step;
    HyperPara[strt:ed,1] = rep(Alpha[j],N_step);
    HyperPara[strt:ed,2] = Lambda;
}

#2. create a function for parallel computaiton
EBglmnet.CVonePar <-function(iHyper,X,y,nFolds,foldId,hyperPara,...)
{
  parameters= hyperPara[iHyper,];
  SSE = CVonePair(X,y,nFolds,foldId,parameters,...);
  return(SSE);
}
```

```
#3. setup parallel computation
library(snow)
library(EBglmnet)
ncl = 4; #use 4 CPUs
cl<-makeCluster(ncl,type="SOCK")
clusterEvalQ(cl,{library(EBglmnet)})
iPar = matrix(seq(1,nPara,1),nPara,1);
nFolds = 5;#5 fold CV
if(n%%nFolds!=0){
    foldId= sample(c(rep(1:nFolds,floor(n/nFolds)),1:(n%%nFolds)),n);
}else{
    foldId= sample(rep(1:nFolds,floor(n/nFolds)),n);
}
#call parRapply to perform parallel computation
SSE = parRapply(cl,iPar,EBglmnet.CVonePar,BASIS[index,],y,nFolds,foldId,HyperPara,
                Epis = epis, prior = priorPara, family= familyPara);
#collect the result in CVresult
CVresult =matrix(SSE,nPara,4,byrow=TRUE)#4 columns of cv$
stopCluster(cl)
```

References

EBglmnet Algorithms:

Anhui Huang, Shizhong Xu, and Xiaodong Cai. (2015). Empirical Bayesian elastic net for multiple quantitative trait locus mapping. Heredity, Vol. 114(1), 107-115.

Anhui Huang, Shizhong Xu, and Xiaodong Cai. (2014a). Whole-genome quantitative trait locus mapping reveals major role of epistasis on yield of rice. PLoS ONE, Vol. 9(1) e87330.

Anhui Huang, Eden Martin, Jeffery Vance, and Xiaodong Cai (2014b). Detecting genetic interactions in pathway-based genome-wide association studies. Genetic Epidemiology, 38(4), 300-309.

Anhui Huang, Shizhong Xu, and Xiaodong Cai. (2013). Empirical Bayesian LASSO-logistic regression for multiple binary trait locus mapping. BMC Genetics, 14(1),5.

Xiaodong Cai, Anhui Huang, and Shizhong Xu (2011). Fast empirical Bayesian LASSO for multiple quantitative trait locus mapping. BMC Bioinformatics, 12(1),211.

Lasso and glmnet package

Tibshirani, R., Bien, J., Friedman, J., Hastie, T., Simon, N., Taylor, J.,and Tibshirani, R.J., 2012. Strong rules for discarding predictors in lasso-type problems. J. R. Stat. Soc. Series B. Stat. Methodol. 74, 245-266.

Friedman, J., T. Hastie, et al. (2010). Regularization paths for generalized linear models via coordinate descent. J. Stat. Softw. 33(1): 1-22.

Park, T. and G. Casella (2008). The Bayesian lasso. J. Am. Stat. Assoc. 103(482): 681-686.

Yi, N., and S. Xu (2008). Bayesian LASSO for quantitative trait loci mapping. Genetics 179(2): 1045-1055.

Zou, H., and T. Hastie (2005). Regularization and variable selection via the elastic net. J. Roy. Stat. Soc. B. Met. 67(2): 301-320.

Tibshirani, R. (1996). Regression shrinkage and selection via the lasso. J. Roy. Stat. Soc. B. Met. 58(1): 267-288.

Separation in logistic regression

Altman M, Gill J, and Mcdonald M P. Numerical Issues in Statistical Computing for the Social Scientist. Journal of the American Statistical Association, 2005, 100(470):707-708.