`llagcdnet` Vignette

He Zhou (zhou1354@umn.edu)

May 3, 2021

Contents

1 Introduction

`llagcdnet` is a package that uses a generalized coordinate descent (GCD) algorithm (Yang and Zou, 2013) for computing the solution path of the LASSO, elastic net (adaptive), and folded concave (SCAD) penalized least squares, logistic regression, HHSVM, squared hinge loss SVM, expectile regression and probit regression.

For LASSO and elastic net (adaptive) penalized problems, most of the part is contributed by Yi Yang and Hui Zou. The probit regression, which can also be formulated as a large margin classifier, is contributed by He Zhou.

For folded concave (SCAD) penalized problems, `llagcdnet` uses the local linear approximation (LLA) (Fan et al., 2014) along with the GCD algorithm for computing the solution path of the least squares, logistic regression, HHSVM, squared hinge loss SVM, expectile regression and probit regression. This part of the package is contributed by He Zhou.

1.1 LASSO and Elastic Net (adaptive) Penalty

Function `gcdnet` solves the following problem

$$\min_{\beta} \frac{1}{N} \sum_{i=1}^{N} \mathrm{Loss}(y_i, \beta_0 + \mathbf{x}_i^{\intercal} \beta) + \lambda ||\beta||_1 + \frac{\lambda_2}{2} ||\beta||_2^2,$$

for a fixed value of λ_2 and a grid of values of λ covering the entire range, where λ, $\lambda_2 \geq 0$ are regularization parameters. Here $\mathrm{Loss}(y_i, \eta_i)$ is the loss function for observation i; e.g. for the least square case it is $\frac{1}{2}(y_i - \eta_i)^2$. The *elastic-net penalty* is controlled by λ and λ_2, and bridges the gap between lasso ($\lambda_2 = 0$, the default) and ridge ($\lambda = 0$).

Function `gcdnet` also allows adaptive LASSO or adaptive elastic net that set different weights for different coefficient. Then the penalty becomes

$$\lambda \sum_j w_j |\beta_j| + \frac{\lambda_2}{2} \sum_j w_j^{(2)} \beta_j^2.$$

For fixed value of λ_2 and fixed adaptive weights w_j's, $w_j^{(2)}$'s, a solution path for a grid of values of λ is solved.

The `gcdnet` algorithms use generalized coordinate descent which can be applied to the loss function that does not have a smooth first derivative everywhere, such as the hybrid Huberized support vector machine (HHSVM) (Wang et al., 2008). It takes advantage of a majorization–minimization trick to make each coordinate-wise update simple and efficient. Due to highly efficient updates and techniques such as warm starts and active-set convergence, this algorithms can compute the solution path very fast.

The core of package `llagcdnet` is a set of fortran subroutines, which make for very fast execution.

The package also includes methods for prediction and plotting of `gcdnet` object, and a function that performs K-fold cross-validation for `gcdnet`.

1.2 Folded Concave (SCAD) Penalty

Function `lla.gcdnet` solves the following problem

$$\min_{\beta} \frac{1}{N} \sum_{i=1}^{N} \mathrm{Loss}(y_i, \beta_0 + \mathbf{x}_i^{\intercal}\beta) + P_{\lambda}(|\beta|),$$

where $P_{\lambda}(|\beta|) = \sum_j p_{\lambda}(|\beta_j|)$ is a *folded concave penalty (SCAD)* (Fan and Li, 2001) with the derivative

$$P_{\lambda}^{'}(t) = \lambda I_{\{t \le \lambda\}} + \frac{(a\lambda - 1)_{+}}{a-1} I_{\{t > \lambda\}},$$

for some $a > 2$, where λ is the regularization parameter.

The local linear approximation (LLA) algorithm (Zou and Li, 2008; Fan et al., 2014) takes advantage of the special folded concave structure and utilizes the majorization-minimization (MM) principle to turn a concave regularization problem into a sequence of weighted ℓ_1 penalized problems. Within each LLA iteration, the local linear approximation is the best convex majorization of the concave penalty function (see Theorem 2 of Zou and Li (2008)). Moreover, Fan et al. (2014) showed that for the SCAD penalized linear regression, logistic regression, precision matrix estimation and quantile regression, the local linear approximation (LLA) algorithm initialized by zero converges to the oracle estimator after three iterations with high probability.

The `lla.gcdnet` algorithms use the LLA algorithm as the outer loop to turn the folded concave regularization problem into a sequence of weighted ℓ_1 penalized problems, and the GCD algorithm as the inner loop for solving those weighted ℓ_1 penalized problems.

The package also includes methods for prediction and plotting of `lla.gcdnet` object, and a function that performs K-fold cross-validation for `lla.gcdnet`.

2 Installation

The way to obtain `llgcdnet` is to clone it from the GitHub, generate the "*.tar.gz" file and install it to R. Type the following command in Unix command to clone the project

```
git clone https://github.umn.edu/zhou1354/llagcdnet.git
```

go to the folder containing file "notes" and folder "gcdnet" and type the following command in Unix command to generate the "*.tar.gz" file

```
R CMD build gcdnet2
```

Type the following command in R console to install the package

```r
install.packages("~/llagcdnet/package/llagcdnet_1.0.0.tar.gz",
                 repos = NULL, type = "source")
```

Users may change the `pkgs` options depending on their locations. Other options such as the directories where to install the packages can be altered in the command. For more details, see `help(install.packages)`.

Here the R package has been downloaded and installed to the default directories.

3 Quick Start

The purpose of this section is to give users a general sense of the package, including the components, what they do and some basic usage. We will briefly go over the main functions, see the basic operations and have a look at the outputs. Users may have a better idea after this section what functions are available, which one to choose, or at least where to seek help. More details are given in later sections.

First, we load the `llagcdnet` package:

```
library(llagcdnet)

## Loading required package:  Matrix
```

In this section, we will demonstrate the usage of the package under the Gaussian linear model or "least squares". We load a set of data created beforehand for illustration. Users can either load their own data or use those saved in the workspace.

```
data(FHT)
```

3.1 Basics of `gcdnet` and Its Related

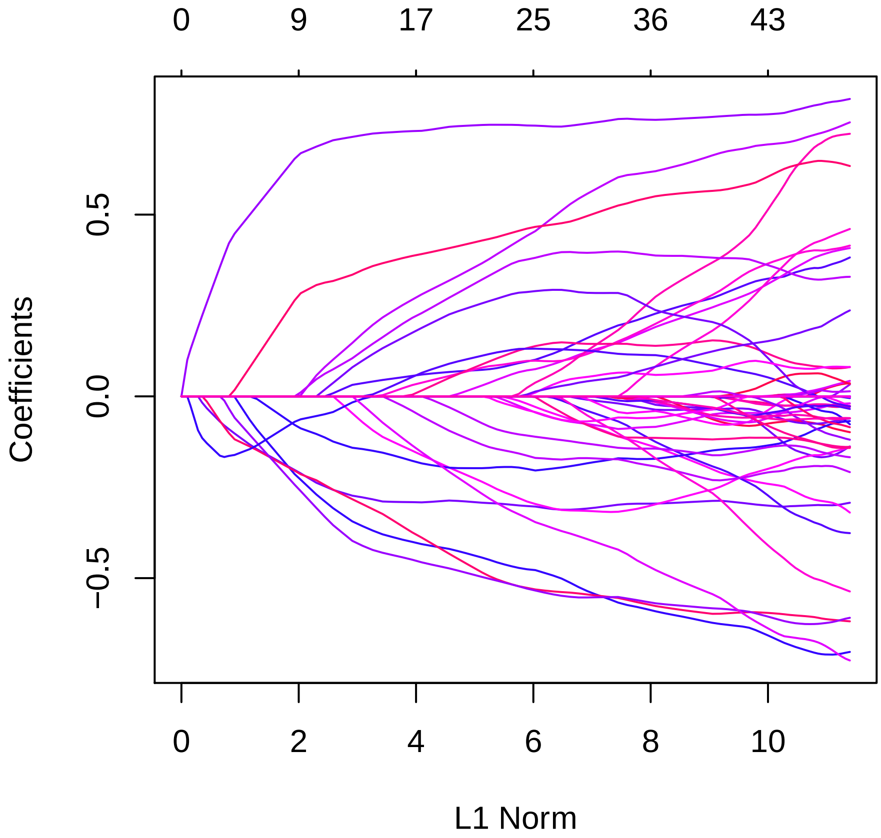
For the elastic net penalized least squared problem, we fit the model using the most basic call to `gcdnet`.

```
fit = gcdnet(x=FHT$x, y=FHT$y_reg, method="ls")
```

"fit" is an object of class `gcdnet` that contains all the relevant information of the fitted model for further use. We do not encourage users to extract the components directly. Instead, various methods are provided for the object such as `plot`, `print`, `coef` and `predict` that enable us to execute those tasks more elegantly.

We can visualize the coefficients by executing the `plot` function:

```
plot(fit)
```



Each curve corresponds to a variable. It shows the path of its coefficient against the ℓ_1-norm of the whole coefficient vector at as λ varies. The axis above indicates the number of nonzero coefficients at the current λ, which is the effective degrees of freedom (df) for the lasso. Users may also wish to annotate the curves; this can be done by setting `label = TRUE` in the plot command.

A summary of the `gcdnet` path at each step is displayed if we just enter the object name or use the `print` function:

```
print(fit)

##
## Call:  gcdnet(x = FHT$x, y = FHT$y_reg, method = "ls")
##
##         Df  Lambda
##    [1,]  0 1.46100
##    [2,]  1 1.39500
##    [3,]  1 1.33100
##    [4,]  1 1.27100
##    [5,]  1 1.21300
##    [6,]  2 1.15800
##    [7,]  2 1.10500
##    [8,]  2 1.05500
```

```
##   [9,]  3 1.00700
##  [10,]  4 0.96140
##  [11,]  4 0.91770
##  [12,]  4 0.87600
##  [13,]  4 0.83620
##  [14,]  5 0.79820
##  [15,]  5 0.76190
##  [16,]  7 0.72730
##  [17,]  7 0.69420
##  [18,]  7 0.66270
##  [19,]  8 0.63260
##  [20,]  8 0.60380
##  [21,]  8 0.57640
##  [22,]  8 0.55020
##  [23,]  8 0.52520
##  [24,]  8 0.50130
##  [25,]  8 0.47850
##  [26,] 10 0.45680
##  [27,] 10 0.43600
##  [28,] 10 0.41620
##  [29,] 11 0.39730
##  [30,] 12 0.37920
##  [31,] 13 0.36200
##  [32,] 13 0.34550
##  [33,] 14 0.32980
##  [34,] 14 0.31480
##  [35,] 15 0.30050
##  [36,] 16 0.28690
##  [37,] 16 0.27380
##  [38,] 17 0.26140
##  [39,] 17 0.24950
##  [40,] 18 0.23820
##  [41,] 18 0.22730
##  [42,] 18 0.21700
##  [43,] 19 0.20710
##  [44,] 19 0.19770
##  [45,] 19 0.18870
##  [46,] 19 0.18020
##  [47,] 20 0.17200
##  [48,] 21 0.16410
##  [49,] 22 0.15670
##  [50,] 22 0.14960
##  [51,] 23 0.14280
##  [52,] 25 0.13630
##  [53,] 26 0.13010
##  [54,] 26 0.12420
```

```
##  [55,] 27 0.11850
##  [56,] 28 0.11310
##  [57,] 29 0.10800
##  [58,] 29 0.10310
##  [59,] 30 0.09840
##  [60,] 31 0.09393
##  [61,] 31 0.08966
##  [62,] 32 0.08559
##  [63,] 33 0.08170
##  [64,] 35 0.07798
##  [65,] 37 0.07444
##  [66,] 36 0.07106
##  [67,] 36 0.06783
##  [68,] 37 0.06474
##  [69,] 37 0.06180
##  [70,] 37 0.05899
##  [71,] 38 0.05631
##  [72,] 38 0.05375
##  [73,] 38 0.05131
##  [74,] 40 0.04898
##  [75,] 43 0.04675
##  [76,] 43 0.04463
##  [77,] 42 0.04260
##  [78,] 42 0.04066
##  [79,] 40 0.03881
##  [80,] 41 0.03705
##  [81,] 42 0.03536
##  [82,] 43 0.03376
##  [83,] 44 0.03222
##  [84,] 44 0.03076
##  [85,] 46 0.02936
##  [86,] 46 0.02803
##  [87,] 44 0.02675
##  [88,] 44 0.02554
##  [89,] 44 0.02438
##  [90,] 45 0.02327
##  [91,] 44 0.02221
##  [92,] 43 0.02120
##  [93,] 44 0.02024
##  [94,] 44 0.01932
##  [95,] 44 0.01844
##  [96,] 45 0.01760
##  [97,] 46 0.01680
##  [98,] 47 0.01604
##  [99,] 47 0.01531
## [100,] 47 0.01461
```

It shows the number of nonzero coefficients (`Df`) and the value of λ (`Lambda`). By default `gcdnet` calls for 100 values of `lambda`.

We can obtain the actual coefficients at one or more λ's within the range of the sequence:

```
coef(fit, s=0.1)

## 101 x 1 sparse Matrix of class "dgCMatrix"
##                        1
## (Intercept) -0.498636324
## V1          -0.531820125
## V2           0.156582213
## V3          -0.309231562
## V4           0.749517675
## V5           .
## V6           0.552706045
## V7           .
## V8           0.119473932
## V9          -0.074477508
## V10          0.113834032
## V11          .
## V12          0.119769945
## V13          .
## V14          .
## V15          .
## V16          0.144830556
## V17         -0.544010138
## V18          0.000000000
## V19          0.000000000
## V20          .
## V21          .
## V22          0.126320982
## V23          .
## V24          0.000000000
## V25          .
## V26          .
## V27          0.000000000
## V28          0.395087887
## V29          .
## V30          0.000000000
## V31         -0.314760919
## V32          0.000000000
## V33          0.000000000
## V34          .
## V35          0.000000000
## V36          0.493332547
## V37          0.000000000
## V38          .
```

```
## V39         -0.185523821
## V40         -0.036822627
## V41          .
## V42          .
## V43         -0.007947995
## V44          .
## V45          .
## V46          .
## V47         -0.553217434
## V48          .
## V49          .
## V50         -0.169914579
## V51          .
## V52         -0.074966679
## V53         -0.004745872
## V54          0.000000000
## V55          .
## V56          .
## V57          0.000000000
## V58          .
## V59          .
## V60          .
## V61         -0.075408944
## V62          0.000000000
## V63          .
## V64          0.000000000
## V65          .
## V66          .
## V67          .
## V68          0.000000000
## V69          0.000000000
## V70          0.285508509
## V71          0.000000000
## V72         -0.128964047
## V73          0.000000000
## V74          .
## V75          .
## V76          .
## V77          0.052581561
## V78         -0.047025633
## V79          0.000000000
## V80          0.000000000
## V81          .
## V82          0.000000000
## V83          0.000000000
## V84          .
```

```
## V85           .
## V86           0.000000000
## V87           0.000000000
## V88           0.040179355
## V89           .
## V90           .
## V91           .
## V92           0.000000000
## V93           .
## V94           0.000000000
## V95          -0.390008556
## V96           .
## V97          -0.068960432
## V98           0.000000000
## V99           0.000000000
## V100          .
```

(why `s` and not `lambda`? In case later we want to allow one to specify the model size in other ways.)

Users can also make predictions at specific λ's with new input data:

```
nx = matrix(rnorm(10*100),10,100)
predict(fit, newx=nx, s=c(0.1,0.05))
```

```
##                1           2
##  [1,] -1.6155353 -1.81476983
##  [2,] -1.9633857 -2.77673849
##  [3,] -1.7935654 -2.18398345
##  [4,]  1.7485133  1.56097773
##  [5,] -0.8411666 -0.10443267
##  [6,] -0.6077084 -0.09659541
##  [7,]  0.5644976  0.63919039
##  [8,]  2.5337119  2.84256304
##  [9,] -0.3279143 -0.27472291
## [10,]  0.7564969  0.05499798
```

The function `gcdnet` returns a sequence of models for the users to choose from. In many cases, users may prefer the software to select one of them. Cross-validation is perhaps the simplest and most widely used method for that task.

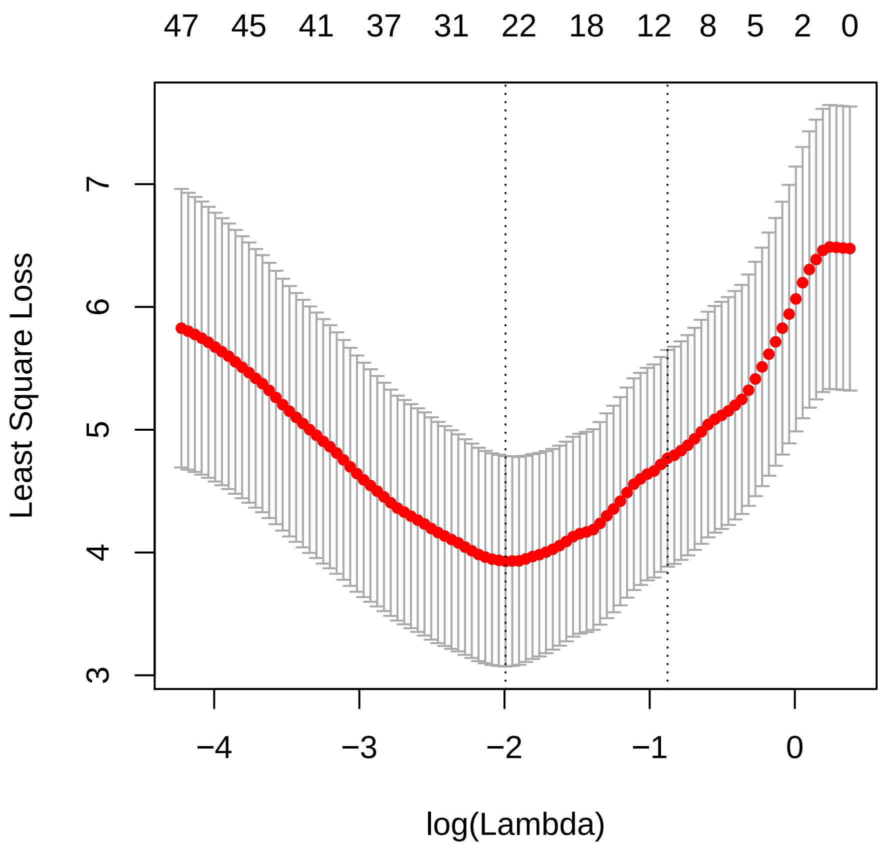
`cv.gcdnet` is the main function to do cross-validation here, along with various supporting methods such as plotting and prediction. We still act on the sample data loaded before.

```
cvfit = cv.gcdnet(FHT$x, FHT$y_reg, method="ls")
```

`cv.gcdnet` returns a `cv.gcdnet` object, which is "cvfit" here, a list with all the ingredients of the cross-validation fit. As for `gcdnet`, we do not encourage users to extract the components directly except for viewing the selected values of λ. The package provides well-designed functions for potential tasks.

We can plot the object.

```r
plot(cvfit)
```



It includes the cross-validation curve (red dotted line), and upper and lower standard deviation curves along the λ sequence (error bars). Two selected λ's are indicated by the vertical dotted lines (see below).

We can view the selected λ's and the corresponding coefficients. For example,

```r
cvfit$lambda.min
```

```
## [1] 0.1362791
```

`lambda.min` is the value of λ that gives minimum mean cross-validated error. The other λ saved is `lambda.1se`, which gives the most regularized model such that error is within one standard error of the minimum. To use that, we only need to replace `lambda.min` with `lambda.1se` above.

```r
coef(cvfit, s = "lambda.min")
```

```
## 101 x 1 sparse Matrix of class "dgCMatrix"
##                       1
## (Intercept) -0.474963839
## V1          -0.475604936
## V2           0.095916703
## V3          -0.301703049
## V4           0.745191104
## V5           .
## V6           0.441587362
## V7           .
## V8           0.071401294
## V9          -0.020102084
## V10          0.097541024
## V11          .
## V12          0.023546651
## V13          .
## V14          .
## V15          .
## V16          0.132555807
## V17         -0.526707397
## V18          0.000000000
## V19          0.000000000
## V20          .
## V21          .
## V22          0.131022339
## V23          .
## V24          0.000000000
## V25          .
## V26          .
## V27          0.000000000
## V28          0.376393789
## V29          .
## V30          0.000000000
## V31         -0.289050386
## V32          0.000000000
## V33          0.000000000
## V34          .
## V35          0.000000000
## V36          0.461426906
## V37          0.000000000
## V38          .
## V39         -0.199423192
## V40          0.000000000
## V41          .
```

```
## V42            .
## V43            0.000000000
## V44            .
## V45            .
## V46            .
## V47           -0.528795010
## V48            .
## V49            .
## V50           -0.162940776
## V51            .
## V52           -0.034717238
## V53            0.000000000
## V54            0.000000000
## V55            .
## V56            .
## V57            0.000000000
## V58            .
## V59            .
## V60            .
## V61            0.000000000
## V62            0.000000000
## V63            .
## V64            0.000000000
## V65            .
## V66            .
## V67            .
## V68            0.000000000
## V69            0.000000000
## V70            0.287027027
## V71            0.000000000
## V72           -0.107617487
## V73            0.000000000
## V74            .
## V75            .
## V76            .
## V77            0.001629251
## V78            0.000000000
## V79            0.000000000
## V80            0.000000000
## V81            .
## V82            0.000000000
## V83            0.000000000
## V84            .
## V85            .
## V86            0.000000000
## V87            0.000000000
```

```
## V88          0.005076475
## V89          .
## V90          .
## V91          .
## V92          0.000000000
## V93          .
## V94          0.000000000
## V95         -0.334506613
## V96          .
## V97         -0.037746174
## V98          0.000000000
## V99          0.000000000
## V100         .
```

Note that the coefficients are represented in the sparse matrix format. The reason is that the solutions along the regularization path are often sparse, and hence it is more efficient in time and space to use a sparse format. If you prefer non-sparse format, pipe the output through `as.matrix()`.

Predictions can be made based on the fitted `cv.gcdnet` object. Let's see a toy example.

```
predict(cvfit, newx = FHT$x[1:5,], s = "lambda.min")

##               1
## [1,] -1.1050013
## [2,] -0.7315396
## [3,] -3.4118931
## [4,] -0.8358573
## [5,] -0.3163643
```

`newx` is for the new input matrix and `s`, as before, is the value(s) of λ at which predictions are made.

That is the end of quick start for `gcdnet`. With the tools introduced so far, users are able to fit the entire elastic net family, including ridge regression, using squared-error loss. In the package, there are many more options that give users a great deal of flexibility. To learn more, move on to later sections.

3.2 Basics of `lla.gcdnet` and Its Related

For the folded concave penalized least squared problem, we fit the model using the most basic call to `lla.gcdnet`.

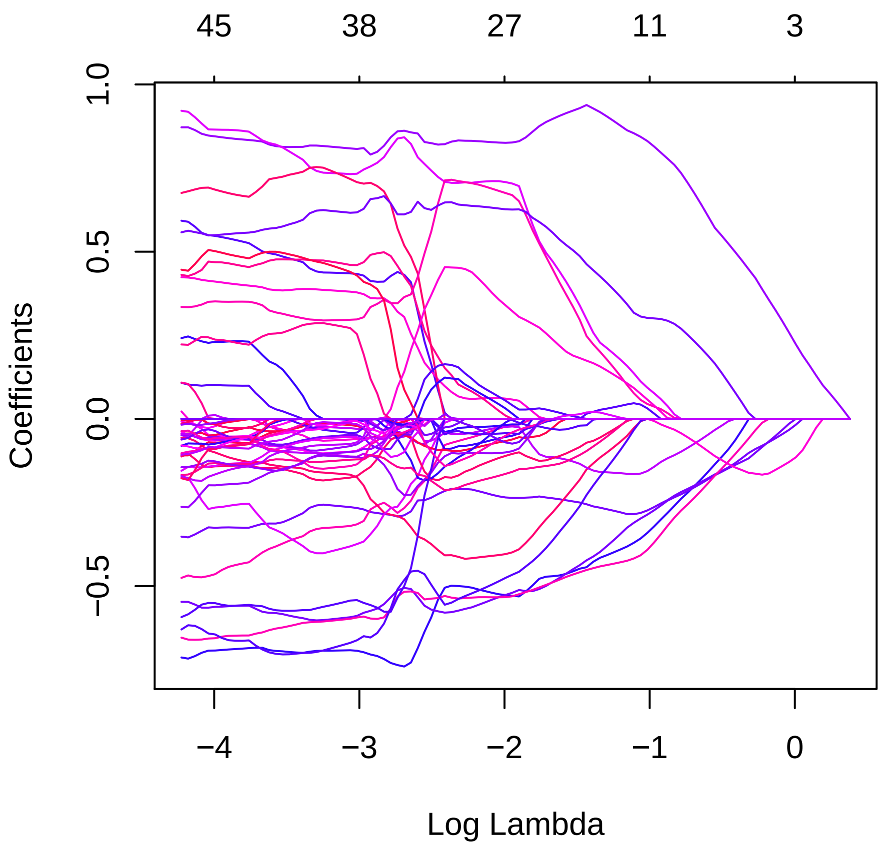
```
lla.fit = lla.gcdnet(x=FHT$x, y=FHT$y_reg, method="ls")
```

"fit" is an object of class `lla.gcdnet` that contains all the relevant information of the fitted model for further use. Again, we do not encourage users to extract the components directly.

Instead, various methods are provided for the object such as `plot`, `print`, `coef` and `predict` that enable us to execute those tasks more elegantly.

We can also visualize the coefficients by executing the `plot` function:

```
plot(lla.fit, xvar="lambda")
```



ach curve corresponds to a variable. It shows the path of its coefficient as regularization parameter λ varies. The axis above indicates the number of nonzero coefficients at the current λ, which is the effective degrees of freedom (df) for the lasso. Users may also wish to annotate the curves; this can be done by setting `label = TRUE` in the plot command.

A summary of the `lla.gcdnet` path at each step is displayed if we just enter the object name or use the `print` function:

```
print(lla.fit)

##
## Call:  lla.gcdnet(x = FHT$x, y = FHT$y_reg, method = "ls")
##
##      Df   Lambda
## s0    0 1.46100
## s1    1 1.39500
```

```
## s2   1 1.33100
## s3   1 1.27100
## s4   1 1.21300
## s5   2 1.15800
## s6   2 1.10500
## s7   2 1.05500
## s8   3 1.00700
## s9   4 0.96140
## s10  4 0.91770
## s11  4 0.87600
## s12  4 0.83620
## s13  5 0.79820
## s14  5 0.76190
## s15  7 0.72730
## s16  7 0.69420
## s17  7 0.66270
## s18  8 0.63260
## s19  8 0.60380
## s20  8 0.57640
## s21  8 0.55020
## s22  8 0.52520
## s23  8 0.50130
## s24  8 0.47850
## s25  8 0.45680
## s26 10 0.43600
## s27 11 0.41620
## s28 11 0.39730
## s29 12 0.37920
## s30 11 0.36200
## s31 13 0.34550
## s32 13 0.32980
## s33 15 0.31480
## s34 16 0.30050
## s35 16 0.28690
## s36 16 0.27380
## s37 16 0.26140
## s38 17 0.24950
## s39 17 0.23820
## s40 17 0.22730
## s41 17 0.21700
## s42 17 0.20710
## s43 18 0.19770
## s44 18 0.18870
## s45 19 0.18020
## s46 23 0.17200
## s47 23 0.16410
```

```
## s48 24 0.15670
## s49 25 0.14960
## s50 26 0.14280
## s51 27 0.13630
## s52 27 0.13010
## s53 27 0.12420
## s54 27 0.11850
## s55 27 0.11310
## s56 27 0.10800
## s57 27 0.10310
## s58 28 0.09840
## s59 29 0.09393
## s60 31 0.08966
## s61 32 0.08559
## s62 34 0.08170
## s63 33 0.07798
## s64 39 0.07444
## s65 40 0.07106
## s66 39 0.06783
## s67 38 0.06474
## s68 41 0.06180
## s69 40 0.05899
## s70 38 0.05631
## s71 39 0.05375
## s72 38 0.05131
## s73 38 0.04898
## s74 38 0.04675
## s75 38 0.04463
## s76 38 0.04260
## s77 38 0.04066
## s78 37 0.03881
## s79 38 0.03705
## s80 40 0.03536
## s81 41 0.03376
## s82 41 0.03222
## s83 41 0.03076
## s84 41 0.02936
## s85 42 0.02803
## s86 44 0.02675
## s87 44 0.02554
## s88 43 0.02438
## s89 43 0.02327
## s90 44 0.02221
## s91 44 0.02120
## s92 44 0.02024
## s93 45 0.01932
```

```
## s94 45 0.01844
## s95 46 0.01760
## s96 45 0.01680
## s97 46 0.01604
## s98 47 0.01531
## s99 47 0.01461
```

It shows the nonzero coefficients (`Df`) and the value of λ (`Lambda`). By default `lla.gcdnet` calls for 100 values of `lambda`.

We can obtain the actual coefficients at one or more λ's within the range of the sequence:

```
coef(lla.fit, s=0.1)
```

```
## 101 x 1 sparse Matrix of class "dgCMatrix"
##                       1
## (Intercept) -0.577289898
## V1          -0.500514616
## V2           .
## V3          -0.209522843
## V4           0.832328307
## V5           .
## V6           0.706339996
## V7           .
## V8           0.065266695
## V9          -0.064119772
## V10          0.099384263
## V11          .
## V12          .
## V13          .
## V14         -0.081565397
## V15          .
## V16          0.148095427
## V17         -0.571715092
## V18          .
## V19          .
## V20          .
## V21          .
## V22          .
## V23          .
## V24          .
## V25          .
## V26          .
## V27          .
## V28          0.710056677
## V29          .
## V30          .
```

```
## V31          -0.414539057
## V32            .
## V33          -0.026592713
## V34            .
## V35            .
## V36           0.640229196
## V37            .
## V38            .
## V39          -0.045222842
## V40            .
## V41            .
## V42            .
## V43            .
## V44            .
## V45            .
## V46            .
## V47          -0.536698760
## V48            .
## V49            .
## V50          -0.165542079
## V51            .
## V52          -0.095419141
## V53          -0.111175349
## V54            .
## V55          -0.038288801
## V56            .
## V57            .
## V58            .
## V59            .
## V60            .
## V61          -0.103276615
## V62            .
## V63            .
## V64            .
## V65            .
## V66            .
## V67            .
## V68            .
## V69            .
## V70           0.450645230
## V71            .
## V72          -0.202919553
## V73            .
## V74            .
## V75            .
## V76            .
```

```
## V77          0.113615635
## V78          .
## V79         -0.008622049
## V80          .
## V81          .
## V82          .
## V83          .
## V84          .
## V85          .
## V86          .
## V87         -0.124030925
## V88          .
## V89          .
## V90          .
## V91          .
## V92          .
## V93          .
## V94          .
## V95         -0.535793913
## V96          .
## V97         -0.010060243
## V98          .
## V99          .
## V100         .
```

Users can also make predictions at specific λ's with new input data:

```
nx = matrix(rnorm(10*100),10,100)
predict(lla.fit,newx=nx,s=c(0.1,0.05))

##                1          2
##  [1,]  1.3591719 -0.4287237
##  [2,]  2.1255348  1.2909193
##  [3,]  0.7825226 -1.8235604
##  [4,] -1.4879999 -1.2621785
##  [5,] -0.6526655 -3.2507980
##  [6,] -0.5972297  1.8751093
##  [7,]  0.1776675 -1.2788363
##  [8,]  3.1372759  1.6169057
##  [9,] -7.3153544 -4.3328267
## [10,]  1.6685015  2.9872893
```

The function `lla.gcdnet` returns a sequence of models for the users to choose from. In many cases, users may prefer the software to select one of them. Cross-validation is perhaps the simplest and most widely used method for that task.

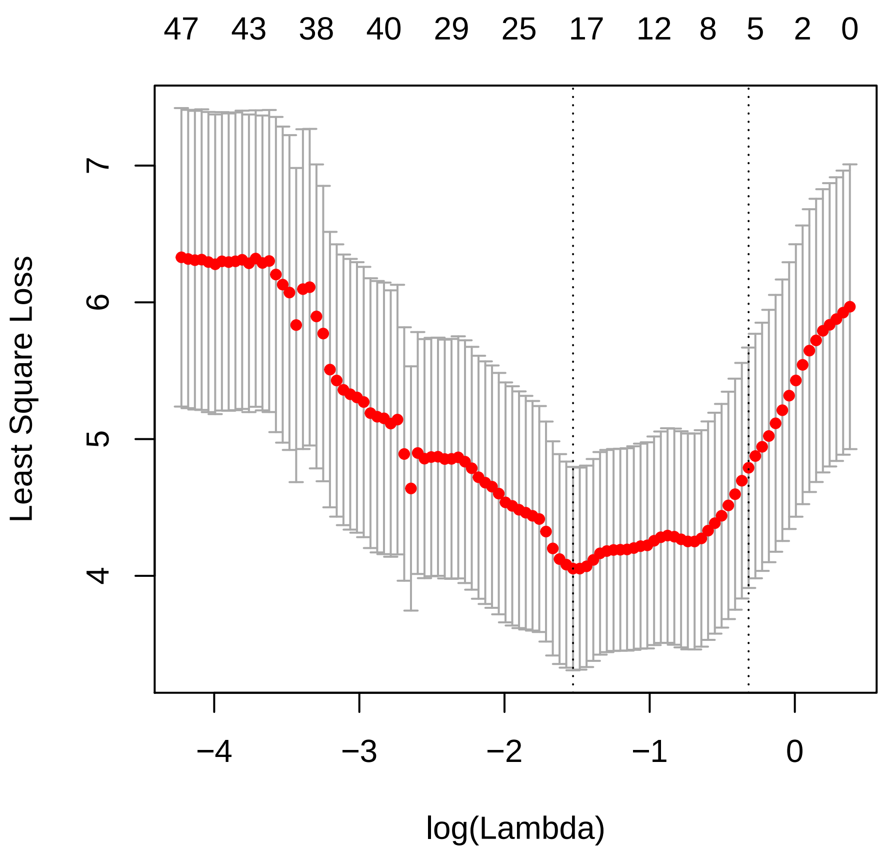
`cv.lla.gcdnet` is the main function to do cross-validation here, along with various supporting methods such as plotting and prediction. We still act on the sample data loaded before.

```
lla.cvfit = cv.lla.gcdnet(FHT$x, FHT$y_reg, method="ls")
```

Function `cv.lla.gcdnet`‘ returns a `cv.lla.gcdnet` object, which is "lla.cvfit" here, a list with all the ingredients of the cross-validation fit.

We can plot the `cv.lla.gcdnet` object.

```
plot(lla.cvfit)
```



It includes the cross-validation curve (red dotted line), and upper and lower standard deviation curves along the λ sequence (error bars). Two selected λ's are indicated by the vertical dotted lines (see below).

We can view the selected λ's and the corresponding coefficients. For example,

```
lla.cvfit$lambda.min

## [1] 0.2169948
```

`lambda.min` is the value of λ that gives minimum mean cross-validated error. The other λ saved is `lambda.1se`, which gives the most regularized model such that error is within one standard error of the minimum. To use that, we only need to replace `lambda.min` with `lambda.1se` above.

```
coef(lla.cvfit, s = "lambda.min")
```

```
## 101 x 1 sparse Matrix of class "dgCMatrix"
##                        1
## (Intercept) -0.526873908
## V1          -0.455882931
## V2           .
## V3          -0.246879989
## V4           0.921852268
## V5           .
## V6           0.375356440
## V7           .
## V8           .
## V9           .
## V10          .
## V11          .
## V12          .
## V13          .
## V14          .
## V15          .
## V16          0.008198094
## V17         -0.449591191
## V18          .
## V19          .
## V20          .
## V21          .
## V22          0.011580853
## V23          .
## V24          .
## V25          .
## V26          .
## V27          .
## V28          0.334977302
## V29          .
## V30          .
## V31         -0.206786529
## V32          .
## V33          .
## V34          .
## V35          .
## V36          0.504081271
## V37          .
## V38          .
## V39         -0.127517281
```

```
## V40          .
## V41          .
## V42          .
## V43          .
## V44          .
## V45          .
## V46          .
## V47         -0.465274142
## V48          .
## V49          .
## V50         -0.093709099
## V51          .
## V52          .
## V53          .
## V54          .
## V55          .
## V56          .
## V57          .
## V58          .
## V59          .
## V60          .
## V61          .
## V62          .
## V63          .
## V64          .
## V65          .
## V66          .
## V67          .
## V68          .
## V69          .
## V70          0.189210291
## V71          .
## V72         -0.117790544
## V73          .
## V74          .
## V75          .
## V76          .
## V77          .
## V78         -0.026234272
## V79          .
## V80          .
## V81          .
## V82          .
## V83          .
## V84          .
## V85          .
```

```
## V86          .
## V87          .
## V88          .
## V89          .
## V90          .
## V91          .
## V92          .
## V93          .
## V94          .
## V95         -0.287884508
## V96          .
## V97          .
## V98          .
## V99          .
## V100         .
```

Predictions can be made based on the fitted `cv.lla.gcdnet` object. Let's see a toy example.

```
predict(lla.cvfit, newx = FHT$x[1:5,], s = "lambda.min")

##              1
## [1,] -1.3238093
## [2,] -0.7275523
## [3,] -2.9291119
## [4,] -0.2511574
## [5,] -1.0677517
```

`newx` is for the new input matrix and `s`, as before, is the value(s) of λ at which predictions are made.

That is the end of quick start for `lla.gcdnet` and its related functions. With the tools introduced so far, users are able to fit the folded concave (SCAD) penalized regression or classification problems. In the package, there are many more options that give users a great deal of flexibility. To learn more, move on to later sections.

4 Hybrid Huberized Support Vector Machine (HHSVM)

Hybrid Huberized support vector machine (HHSVM) is proposed in Wang et al. (2008). It uses the elastic net penalty for regularization and variable selection and uses the Huberized squared hinge loss for efficient computation. The HHSVM poses a major challenge for applying the coordinate descent algorithm, because the Huberized hinge loss function does not have a smooth first derivative everywhere. As a result, the coordinate descent algorithm for the elastic net penalized logistic regression (Friedman et al., 2010) cannot be used for solving the HHSVM. To overcome the computational difficulty, Yang and Zou (2013) propose a new generalized coordinate descent (GCD) algorithm for solving the solution paths of the HHSVM.

All the functions and methods discussed in the following part of this section can be applied to penalized least squares, logistic regression, probit regression, squared hinge SVM and expectile regression. We just use HHSVM as an detailed example to show respect to the work done by Yang and Zou (2013). And HHSVM is the motivation of the GCD algorithm.

4.1 LASSO and Elastic Net (Adaptive) Penalized HHSVM

4.1.1 Model and Algorithm

`hhsvm` is the default family option in the function `gcdnet`. Suppose we have observations $\mathbf{x}_i \in \mathbb{R}^p$ and the responses $y_i \in \{-1, 1\}, i = 1, \ldots, N$. The objective function for the elastic net penalized HHSVM is

$$\min_{\beta_0,\beta} \frac{1}{N} \sum_{i=1}^{N} \phi_c \left(y_i(\beta_0 + \mathbf{x}_i^\intercal \beta) \right) + P_{\lambda,\lambda_2}(\beta),$$

where $P_{\lambda,\lambda_2}(\beta) = \lambda\|\beta\|_1 + \frac{\lambda_2}{2}\|\beta\|_2^2$, $\phi_c(\cdot)$ is the Huberized hinge loss

$$\phi_c(t) = \begin{cases} 0, & t > 1 \\ (1-t)^2/2\delta, & 1-\delta < t \le 1 \\ 1 - t - \delta/2, & t \le 1 - \delta \end{cases}$$

Generalized coordinate descent can be applied to solve the problem. Specifically, suppose we have current estimates $\tilde{\beta}_0$ and $\tilde{\beta}$. We want to update the j-th coordinate of β. The current penalized HHSVM objective function can be majorized by a penalized quadratic function defined as

$$Q(\beta_j|\tilde{\beta}_0, \tilde{\beta}) = \frac{1}{N}\sum_{i=1}^{N} \phi_c(r_i) + \left(\frac{1}{N}\sum_{i=1}^{N} \phi_c'(r_i) y_i x_{ij} \right)(\beta_j - \tilde{\beta}_j) + \frac{1}{\delta}(\beta_j - \tilde{\beta}_j)^2 + p_{\lambda,\lambda_2}(\beta_j)$$

where $\phi_c'(t)$ is the first derivative of $\phi_c(t)$ and $r_i = y_i(\tilde{\beta}_0 + \mathbf{x}_i^\intercal \tilde{\beta})$ is the current margin. We can easily solve the minimizer of the above penalized quadratic function by a simple soft-thresholding rule (Zou and Hastie, 2005):

$$\tilde{\beta}_j \leftarrow \frac{S(\frac{2}{\delta}\tilde{\beta}_j - \frac{1}{N}\sum_{i=1}^{N} \phi_c'(r_i) y_i x_{ij}, \lambda)}{\frac{2}{\delta} + \lambda_2},$$

where $S(z, t) = (|z| - t)_+ \mathrm{sgn}(z)$. This formula above applies when the `x` variables are standardized to have zero mean and unit variance; it is slightly more complicated when they are not.

The same trick is used to update intercept β_0:

$$\tilde{\beta}_0 \leftarrow \tilde{\beta}_0 - \frac{\delta}{2}\frac{1}{N}\sum_{i=1}^{N} \phi_c'(r_i) y_i.$$

4.1.2 Function `gcdnet`: Augments and Example

`gcdnet` provides various options for users to customize the fit. We introduce some commonly used options here and they can be specified in the `gcdnet` function.

- `nlambda`: the number of λ values in the sequence. Default is 100.

- `lambda.factor`: the factor for getting the minimal lambda in `lambda` sequence, where `min(lambda) = lambda.factor * max(lambda)`. `max(lambda)` is the smallest value of lambda for which all coefficients are zero. The default depends on the relationship between N (the number of rows in the matrix of predictors) and p (the number of predictors). If $N > p$, the default is 0.0001, close to zero. If $N < p$, the default is 0.01. A very small value of lambda.factor will lead to a saturated fit. It takes no effect if there is user-defined lambda sequence.
- `lambda`: a user supplied `lambda` sequence. Typically, by leaving this option unspecified users can have the program compute its own lambda sequence based on `nlambda` and `lambda.factor`. Supplying a value of `lambda` overrides this. It is better to supply a decreasing sequence of `lambda` values than a single (small) value, if not, the program will sort user-defined `lambda` sequence in decreasing order automatically.
- `lambda2`: regularization parameter for the quadratic penalty of the coefficients.
- `pf`: the ℓ_1 penalty factor of length p used for adaptive LASSO or adaptive elastic net. Separate ℓ_1 penalty weights can be applied to each coefficient of beta to allow different ℓ_1 shrinkage. Can be 0 for some variables, which implies no ℓ_1 shrinkage, and results in that variable always being included in the model. Default is 1 for all variables (and implicitly infinity for variables listed in exclude).
- `pf2`: the ℓ_2 penalty factor of length p used for adaptive LASSO or adaptive elastic net. Separate ℓ_2 penalty weights can be applied to each coefficient of beta to allow different ℓ_2 shrinkage. Can be 0 for some variables, which implies no ℓ_2 shrinkage. Default is 1 for all variables.
- `exclude`: indices of variables to be excluded from the model. Default is none. Equivalent to an infinite penalty factor.
- `standardize`: a logical flag for variable standardization, prior to fitting the model sequence. If `TRUE`, `x` matrix is normalized such that `x` is centered (i.e. $\sum_i x_{ij} = 0$), and sum squares of each column $\frac{1}{N}\sum_i x_{ij}^2 = 1$. If `x` matrix is standardized, the ending coefficients will be transformed back to the original scale. Default is `FALSE`.
- `delta`: the parameter δ in the HHSVM model. The value must be greater than 0. Default is 2.
- `omega`: the parameter ω in the expectile regression model. The value must be in (0,1). Default is 0.5.

For more information, type `help(gcdnet)` or simply `?gcdnet`.

As an example of adaptive elastic net penalized HHSVM, we set $\lambda_2 = 0.01$, and different ℓ_1 and ℓ_2 penalty weights to different coefficients of β. To avoid too long a display here, we set `nlambda` to 20. In practice, however, the number of values of λ is recommended to be 100 (default) or more. In most cases, it does not come with extra cost because of the warm-starts used in the algorithm, and for nonlinear models leads to better convergence properties.

```
p <- ncol(FHT$x)
pf <- c(10,10,10,rep(1,p-3))
pf2 <- c(rep(1,p-3),0.1,0.1,0.1)
fit <- gcdnet(x=FHT$x, y=FHT$y, pf=pf, pf2=pf2, delta=1.5,
```

```
                 lambda2=0.01, nlambda=20)
```

4.1.3 Function `print`

We can then print the `gcdnet` object.

```
print(fit)

##
## Call:  gcdnet(x = FHT$x, y = FHT$y, nlambda = 20, lambda2 = 0.01, pf = pf,      pf2 = pf2
##
##        Df   Lambda
##  [1,]   0 0.316900
##  [2,]   4 0.248700
##  [3,]   6 0.195200
##  [4,]   8 0.153100
##  [5,]  13 0.120200
##  [6,]  21 0.094320
##  [7,]  25 0.074010
##  [8,]  28 0.058080
##  [9,]  31 0.045580
## [10,]  33 0.035770
## [11,]  33 0.028070
## [12,]  35 0.022030
## [13,]  38 0.017290
## [14,]  38 0.013570
## [15,]  39 0.010650
## [16,]  45 0.008355
## [17,]  47 0.006557
## [18,]  51 0.005145
## [19,]  52 0.004038
## [20,]  54 0.003169
```

This displays the call that produced the object "fit" and a two-column matrix with columns `Df` (the number of nonzero coefficients) and `Lambda` (the corresponding value of λ).

4.1.4 Function `plot`

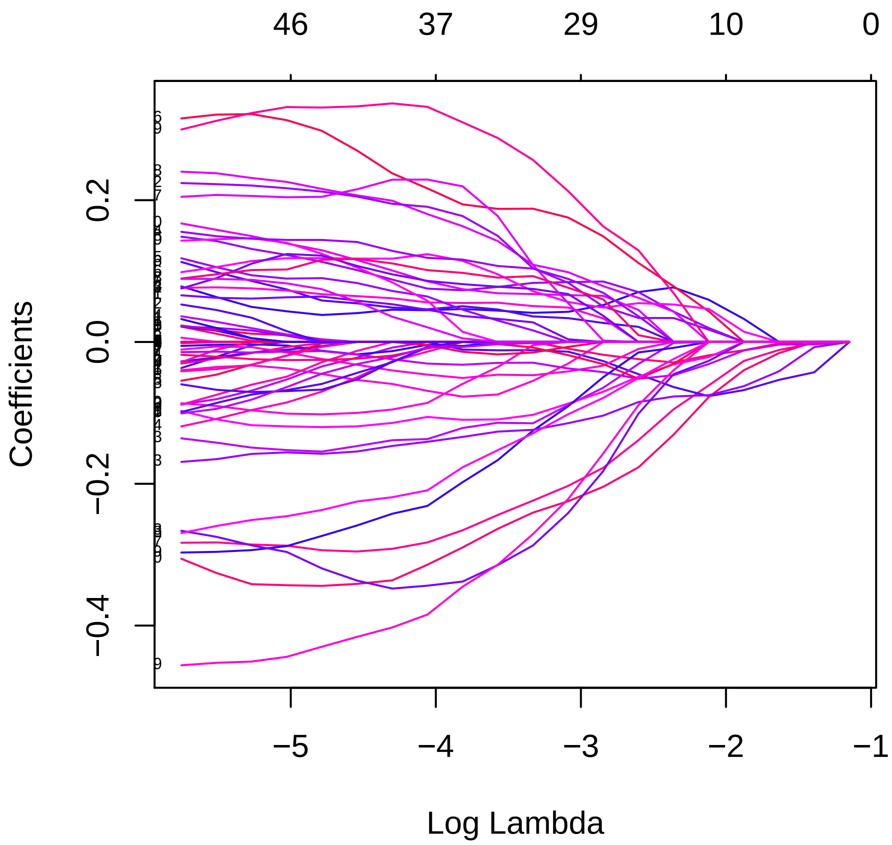
We can plot the fitted object as in the previous section. There are more options in the `plot` function.

Users can decide what is on the X-axis. `xvar` allows two measures: "norm" for the ℓ_1-norm of the coefficients (default) and "lambda" for the log-lambda value.

Users can also label the curves with variable sequence numbers simply by setting `label = TRUE`.

Let's plot "fit" against the log-lambda value and with each curve labeled.

```r
plot(fit, xvar = "lambda", label = TRUE)
```



4.1.5 Function `coef`

We can extract the coefficients and make predictions at certain values of λ.

- `s`: the value(s) of λ at which extraction is made. If `s` is not in the lambda sequence used for fitting the model, the coef function will use linear interpolation to make predictions. The new values are interpolated using a fraction of coefficients from both left and right `lambda` indices.
- `type`: two options, "coefficients" (default), and "nonzero".
  - Type "coefficients" computes the coefficients at the requested values for `s`.
  - Type "nonzero" returns a list of the indices of the nonzero coefficients for each value of `s`.

A simple example is:

```r
coef(fit, type="coef", s = 0.03)

## 101 x 1 sparse Matrix of class "dgCMatrix"
```

```
##                       1
## (Intercept) -0.129284352
## V1           .
## V2           .
## V3           .
## V4           0.043465612
## V5           .
## V6           0.043394236
## V7           0.000000000
## V8           0.079081053
## V9           0.000000000
## V10          0.068429662
## V11          .
## V12          0.000000000
## V13          0.054229959
## V14          0.000000000
## V15          .
## V16          .
## V17         -0.238959818
## V18          0.000000000
## V19          0.000000000
## V20          0.000000000
## V21         -0.011709784
## V22          .
## V23          .
## V24          0.000000000
## V25          0.026762754
## V26          .
## V27          0.000000000
## V28          0.133982587
## V29          .
## V30          0.000000000
## V31         -0.069257583
## V32          .
## V33          0.000000000
## V34          0.000000000
## V35          0.000000000
## V36          0.187810008
## V37          .
## V38          .
## V39          0.000000000
## V40          0.000000000
## V41          0.031392605
## V42          0.138176200
## V43         -0.114133152
## V44          .
```

```
## V45          0.000000000
## V46          0.089492458
## V47         -0.046342520
## V48          0.000000000
## V49          0.279951529
## V50         -0.257790376
## V51          0.000000000
## V52          0.000000000
## V53          0.000000000
## V54          0.000000000
## V55          .
## V56          0.000000000
## V57         -0.029431261
## V58          0.000000000
## V59         -0.107657214
## V60         -0.027639020
## V61          .
## V62          0.000000000
## V63          0.000000000
## V64          0.000000000
## V65          .
## V66          .
## V67          .
## V68         -0.004127351
## V69         -0.156234732
## V70          .
## V71          0.000000000
## V72          0.077176467
## V73         -0.125791660
## V74          0.000000000
## V75          .
## V76          .
## V77          0.160258842
## V78          .
## V79          .
## V80          0.000000000
## V81          0.000000000
## V82          .
## V83          .
## V84          .
## V85          .
## V86          0.000000000
## V87          0.000000000
## V88          0.091303454
## V89          .
## V90         -0.016834277
```

```
## V91          0.000000000
## V92          0.000000000
## V93         -0.307727144
## V94          0.106263880
## V95          .
## V96         -0.002744730
## V97          0.000000000
## V98         -0.146669818
## V99         -0.303358024
## V100         .
```

```
coef(fit, type="nonzero", s = 0.03)
```

```
##        [,1]
##  [1,]     4
##  [2,]     6
##  [3,]     8
##  [4,]    10
##  [5,]    13
##  [6,]    17
##  [7,]    21
##  [8,]    25
##  [9,]    28
## [10,]    31
## [11,]    36
## [12,]    41
## [13,]    42
## [14,]    43
## [15,]    46
## [16,]    47
## [17,]    49
## [18,]    50
## [19,]    57
## [20,]    59
## [21,]    60
## [22,]    68
## [23,]    69
## [24,]    72
## [25,]    73
## [26,]    77
## [27,]    88
## [28,]    90
## [29,]    93
## [30,]    94
## [31,]    96
## [32,]    98
## [33,]    99
```

4.1.6 Function `predict`

Users can make predictions from the fitted object.

- `newx`: a matrix of new values for `x`.
- `s`: value(s) of the penalty parameter lambda at which predictions are required.
- `type`: the type of prediction required:
  - Type `"link"` gives the linear predictors for classification problems and gives predicted response for regression problems.
  - Type `"class"` produces the class label corresponding to the maximum probability. Only available for classification problems.

For example,

```
predict(fit, newx = FHT$x[1:5,], type = "class", s = 0.05)

##        1
## [1,]  1
## [2,] -1
## [3,] -1
## [4,] -1
## [5,] -1
```

gives the fitted values for the first 5 observations at $\lambda = 0.05$. If multiple values of `s` are supplied, a matrix of predictions is produced.

4.1.7 Function `cv.gcdnet`

Users can customize K-fold cross-validation. In addition to all the `gcdnet` parameters, `cv.gcdnet` has its special parameters including

- `nfolds`: number of folds - default is 5. Although `nfolds` can be as large as the sample size (leave-one-out CV), it is not recommended for large datasets. Smallest value allowable is `nfolds`=3.
- `foldid`: an optional vector of values between 1 and `nfold` identifying what fold each observation is in. If supplied, `nfold` can be missing.
- `pred.loss`: loss function to use for cross-validation error. Valid options are:
  - `"loss"`: Margin based loss function, which is the default option. When use least square loss `method="ls"`, it gives mean square error (MSE). When use expectile regression loss `method="er"`, it gives asymmetric mean square error (AMSE).
  - `"misclass"`: Misclassification error. Only available for classification.
- `delta`: parameter only used in HHSVM for computing margin based loss function, only available for `pred.loss = "loss"`.

As an example,

```r
cvfit = cv.gcdnet(FHT$x, FHT$y, lambda2=0.1, pred.loss="misclass",
                  nfolds = 5, delta=1.5)
```

does 5-fold cross-validation, based on misclassification criterion.

Functions `coef` and `predict` on `cv.gcdnet` object are similar to those for a `gcdnet` object, except that two special strings are also supported by `s` (the values of λ requested):

- `"lambda.1se"`: the largest value of `lambda` such that error is within 1 standard error of the minimum.
- `"lambda.min"`: the optimal value of `lambda` that gives minimum cross validation error.

```r
cvfit$lambda.min

## [1] 0.005586467

coef(cvfit, s = "lambda.min")

## 101 x 1 sparse Matrix of class "dgCMatrix"
##                        1
## (Intercept) -0.032991833
## V1          -0.117735879
## V2           0.086141037
## V3          -0.115433232
## V4           0.094018082
## V5           .
## V6           0.141474462
## V7           0.000000000
## V8           0.130436524
## V9          -0.044281800
## V10          0.130416465
## V11          0.000000000
## V12          0.029903580
## V13          0.056014601
## V14         -0.082328323
## V15          .
## V16          0.000000000
## V17         -0.144165901
## V18          0.051762703
## V19         -0.021475874
## V20         -0.036124474
## V21         -0.112682577
## V22         -0.033107136
## V23          .
## V24         -0.026460515
## V25          0.045576506
## V26          0.002418036
```

```
## V27         -0.033028012
## V28          0.126214325
## V29          0.000000000
## V30          0.030085167
## V31         -0.037975444
## V32         -0.003233621
## V33         -0.085039775
## V34         -0.037089260
## V35         -0.044954359
## V36          0.192489505
## V37         -0.047168181
## V38         -0.004446177
## V39          0.026051107
## V40         -0.033730938
## V41          0.075395070
## V42          0.135027477
## V43         -0.067122322
## V44          .
## V45          0.071717050
## V46          0.082511211
## V47         -0.059228314
## V48          0.000000000
## V49          0.147858489
## V50         -0.163273386
## V51          0.007368645
## V52          0.027743128
## V53         -0.022887131
## V54          0.027056217
## V55          .
## V56          0.030321061
## V57         -0.078401294
## V58          0.055861163
## V59         -0.059163282
## V60         -0.052878228
## V61          .
## V62          0.028126184
## V63          .
## V64         -0.007990363
## V65          0.000000000
## V66         -0.055909640
## V67         -0.011303168
## V68         -0.061856434
## V69         -0.156796138
## V70          .
## V71         -0.080120622
## V72          0.057754040
```

```
## V73          -0.132664334
## V74           0.034712077
## V75           0.007021378
## V76           0.000000000
## V77           0.109431414
## V78          -0.008901890
## V79           0.022948146
## V80           .
## V81           .
## V82           .
## V83           .
## V84           .
## V85           0.000000000
## V86          -0.001490084
## V87          -0.021931911
## V88           0.104326522
## V89           0.000000000
## V90          -0.034226202
## V91           0.073799256
## V92           0.023557373
## V93          -0.139482119
## V94           0.148035557
## V95           .
## V96          -0.079435384
## V97           0.046256675
## V98          -0.145423296
## V99          -0.201639188
## V100          .
```

```
predict(cvfit, newx = FHT$x[1:5,], s = "lambda.min")
```

```
##       1
## [1,]  1
## [2,] -1
## [3,] -1
## [4,] -1
## [5,] -1
```

4.2 Folded Concave (SCAD) Penalized HHSVM

4.2.1 Model and Algorithm

`hhsvm` is also the default family option in the function `lla.gcdnet`. The objective function for the folded concave (SCAD) penalized HHSVM is

$$\min_{\beta_0,\beta} \frac{1}{N}\sum_{i=1}^{N}\phi_c\left(y_i(\beta_0 + \mathbf{x}_i^\intercal\beta)\right) + P_\lambda(|\beta|),$$

where $P_\lambda(|\beta|) = \sum_j p_\lambda(|\beta_j|)$ is the folded concave (SCAD) penalty. The local linear approximation (LLA) algorithm along with the GCD algorithm can be applied to solve this problem. Specifically, suppose the current estimates are $\tilde{\beta}_0$ and $\tilde{\beta}$, then the updated estimates are the solution to the following weighted LASSO penalized problem:

$$\min_{\beta_0,\beta} \frac{1}{N}\sum_{i=1}^{N}\phi_c\left(y_i(\beta_0 + \mathbf{x}_i^\intercal\beta)\right) + \sum_{j=1}^{p} p_\lambda'(|\tilde{\beta}_j|)|\beta_j|.$$

This problem can be easily solved by the generalized coordinate descent (GCD) algorithm discussed in the previous section.

4.2.2 Function `lla.gcdnet` and Its Related

The usage of function `lla.gcdnet` is similar to function `gcdnet`. Here's an example of folded concave (SCAD) penalized HHSVM:

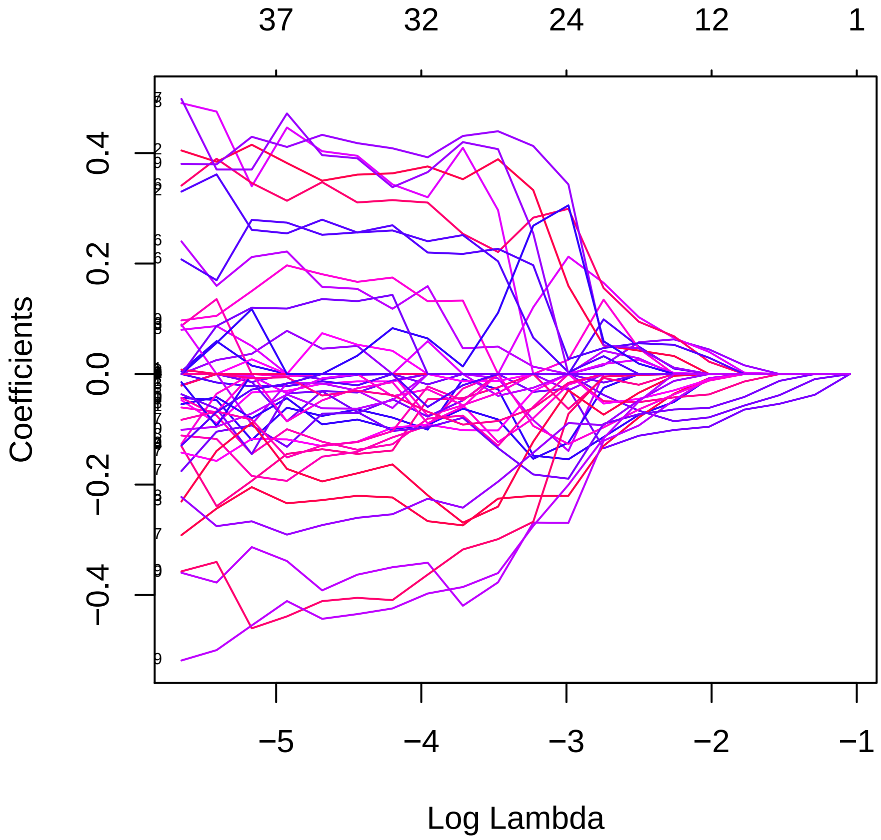
```
lla.fit = lla.gcdnet(FHT$x, FHT$y, nlambda=20, lambda.factor=0.01, delta=1.5)
```

The λ sequence is unspecified, so the program computes its own `lambda` sequence based on `nlambda` and `lambda.factor`.

All those functions used to deal with the `gcdnet` object can also be applied similarly to the `lla.gcdnet`. Those include `print`, `coef`, `predict`, `cv.lla.gcdnet`.

To visualize the coefficients, we use the `plot` function.

```
plot(lla.fit, xvar = "lambda", label = TRUE)
```



We can extract the coefficients at requested values of λ by using the function `coef` and make predictions by `predict`. The usage is similar and we only provide an example of `predict` here.

```
predict(lla.fit, newx = FHT$x[1:5,], s = c(0.1, 0.01))
```

```
##       1  2
## [1,] -1  1
## [2,] -1 -1
## [3,] -1 -1
## [4,] -1 -1
## [5,] -1 -1
```

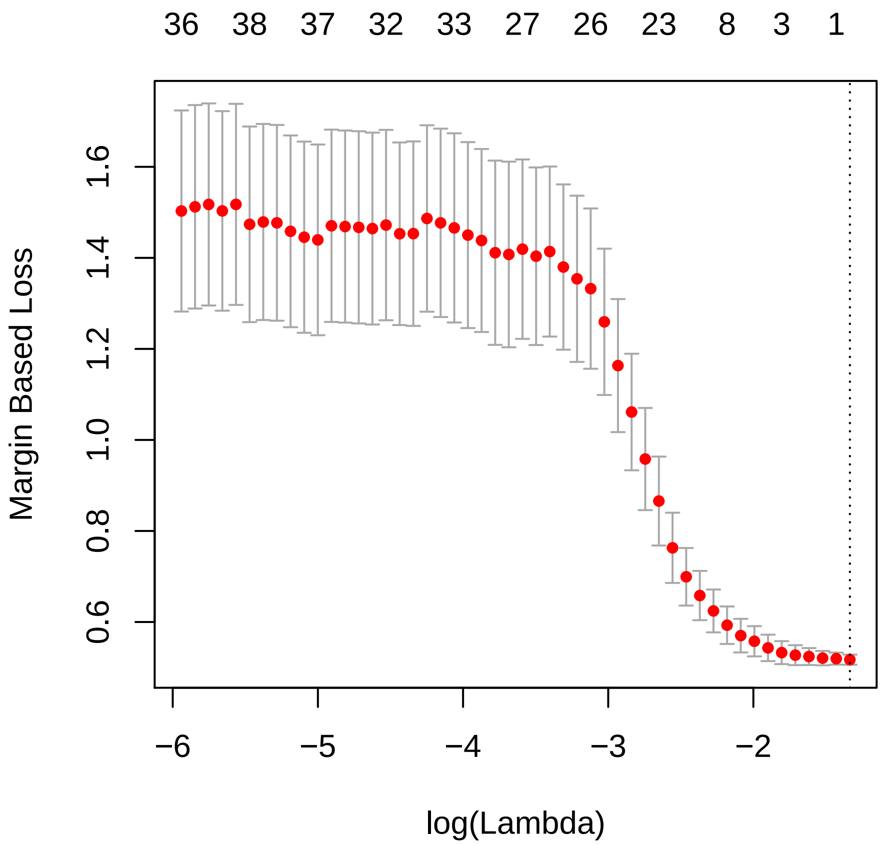
The prediction result is saved in a two column matrix containing the prediction for each new observation (row) and each λ (column).

We can also do K-fold cross-validation for `lla.gcdnet`. The options are almost the same as the function `cv.gcdnet`.

```
lla.cvmfit = cv.lla.gcdnet(FHT$x, FHT$y, pred.loss="loss", nlambda=50)
```

We plot the resulting `cv.lla.gcdnet` object "lla.cvmfit".

```r
plot(lla.cvmfit)
```



To show explicitly the selected optimal values of λ, type

```r
lla.cvmfit$lambda.min

## [1] 0.2631348

lla.cvmfit$lambda.1se

## [1] 0.2631348
```

As before, the first one is the value at which the minimal mean squared error is achieved and the second is for the most regularized model whose mean squared error is within one standard error of the minimal.

5 Penalized Large Margin Classifier

The GCD algorithm can be generalized for solving a class of large margin classifiers (Yang and Zou, 2013), including Huberized SVM, squared SVM, logistic regression and probit regression.

Suppose we are given N pairs of training data $\{\mathbf{x}_i, y_i\}$ for $i = 1, \cdots, n$ where $\mathbf{x}_i \in \mathbb{R}^p$ are predictors and $y_i \in \{-1, 1\}$ denotes class labels. Without loss of generality assume that the input data are standardized and centerized: $\frac{1}{N}\sum_{i=1}^N x_{ij} = 0$, $\frac{1}{N}\sum_{i=1}^N x_{ij}^2 = 1$ for $j = 1, \cdots, p$. Define a penalized large margin classifier as follows:

$$\min_{\beta_0,\beta} \frac{1}{N}\sum_{i=1}^N L\left(y_i(\beta_0 + \mathbf{x}_i^\intercal \beta)\right) + P(|\beta|)$$

where $L(\cdot)$ is a convex loss function and $P(|\beta|)$ is the penalty funciton. For elastic net penalty, $P_{\lambda,\lambda_2}(|\beta|) = \sum_{j=1}^p \lambda|\beta_j| + \frac{\lambda_2}{2}\beta_j^2$; for folded concave (SCAD) penalty, $P_\lambda(|\beta|) = \sum_{j=1}^p p_\lambda(|\beta_j|)$, where $p_\lambda(t)$ is the folded concave (SCAD) function defined in the section 1

To generalized the GCD algorithm, we assume that the loss function L satisfies the following *quadratic majorization condition* with coefficient M:

$$L(t + a) \le L(t) + L^{'}(t)a + \frac{M}{2}a^2, \quad \forall t, a.$$

Given the current estimates $\tilde{\beta}_0$ and $\tilde{\beta}$. Suppose we want to update the j-th coordinate, $j \in \{0, 1, \cdots, p\}$. The objective function could be majorized by a penalized quadratic funcion:

$$Q(\beta_j|\tilde{\beta}_0, \tilde{\beta}) := \frac{1}{N}\sum_{i=1}^N \left[L(r_i) + L^{'}(r_i)y_i x_{ij}(\beta_j - \tilde{\beta}_j) + \frac{M}{2}x_{ij}^2(\beta_j - \tilde{\beta}_j)^2\right] + P(|\beta|),$$

where $r_i = y_i(\tilde{\beta}_0 + \mathbf{x}_i^\intercal \tilde{\beta})$ is the current margin for the i-th pair of data. The new update of j-th coordinate is given by solving

$$\min_{\beta_0,\beta} Q(\beta_j|\tilde{\beta}_0, \tilde{\beta})$$

- For elastic net penalty, the updates have the following closed-form solution:

$$\tilde{\beta}_j \longleftarrow \frac{S\left(M\tilde{\beta}_j - \frac{1}{N}\sum_{i=1}^N L^{'}(r_i)y_i x_{ij}, \lambda\right)}{M + \lambda_2}, \quad \text{if } j \in \{1, \cdots, p\};$$

  and

$$\tilde{\beta}_0 \longleftarrow \tilde{\beta}_0 - \frac{1}{M}\frac{1}{N}\sum_{i=1}^N L^{'}(r_i)y_i.$$

- For folded concave (SCAD) penalty, the problem can be solved by using the local linear approximation (LLA) algorithm along with the GCD algorithm, as discussed in section 4.2.

5.1 Hybrid Huberized SVM (HHSVM)

In section **??**, we have shown that the Huberized hinge loss has $M = 2/\delta$. Examples are also illustrated in that section.

5.2 Probit Regression

The probit regression model is

$$\mathbb{P}(y_i = 1|\mathbf{x}_i) = \Phi(\beta_0 + \mathbf{x}_i^\intercal \beta), \text{ and } \quad \mathbb{P}(y_i = -1|\mathbf{x}_i) = \Phi(-(\beta_0 + \mathbf{x}_i^\intercal \beta)),$$

where $\Phi(\cdot)$ is the cumulative distribution function of the standard normal. Then the negative log-likelihood function (scaled by $1/N$) is

$$\frac{1}{N}\sum_{i=1}^{N} I\{y_i = 1\} - \log(\Phi(\beta_0 + \mathbf{x}_i^\intercal \beta)) - I\{y_i = -1\}\log(\Phi(-(\beta_0 + \mathbf{x}_i^\intercal \beta))),$$

or equivalently,

$$\frac{1}{N}\sum_{i=1}^{N} -\log(\Phi(y_i(\beta_0 + \mathbf{x}_i^\intercal \beta))).$$

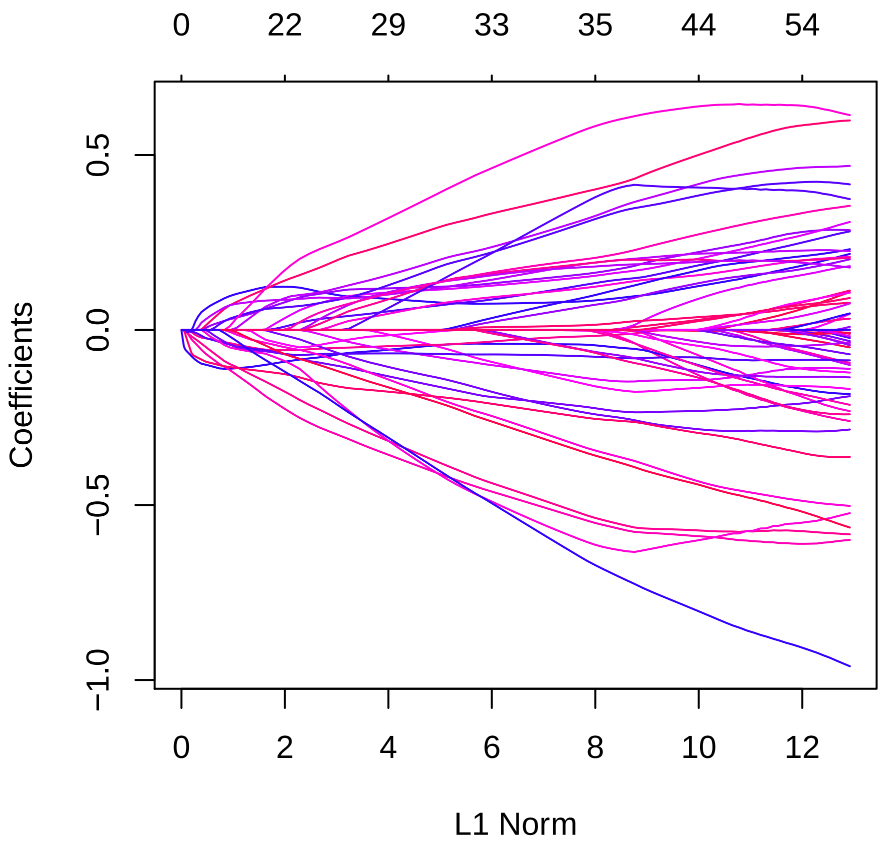
Thus, we notice that the probit regression has the probit regression loss with the expression $L(t) = -\log(\Phi(t))$ and its derivative is $L^{'}(t) = -\varphi(t)/\Phi(t)$, with $\varphi(\cdots)$ being the probability density function of the standard normal. We proved that its second derivative is bounded by 1. So it also satisfies the quadratic majorization condition with $M = 1$.

5.2.1 Examples

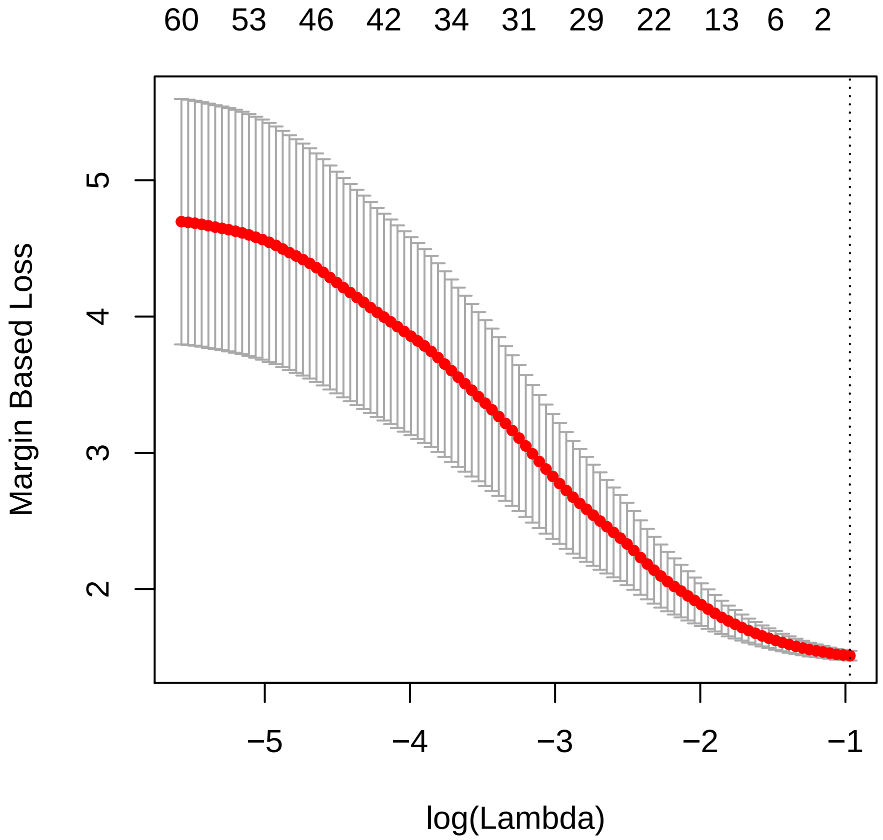
We only need to specify the method as `method="probit"` in function `gcdnet`, `cv.gcdnet`, `lla.gcdnet` and `cv.lla.gcdnet`.

(Adaptive) Elastic Net Penalized Probit Regression: set lambda2 = 0.01; set the first three ℓ_1 penalty weights as 10 and the rest as 1; set the last three ℓ_2 penalty weights as 0.1 and the rest as 1.

```
p <- ncol(FHT$x)
# set the first three L1 penalty weights as 10 and the rest are 1.
pf = c(10,10,10,rep(1,p-3))
# set the last three L2 penalty weights as 0.1 and the rest are 1.
pf2 = c(rep(1,p-3),0.1,0.1,0.1)
# set the L2 penalty parameter lambda2=0.01.
m.probit <- gcdnet(x=FHT$x, y=FHT$y, pf=pf, pf2=pf2,
                   lambda2=0.01, method="probit")
plot(m.probit)
```

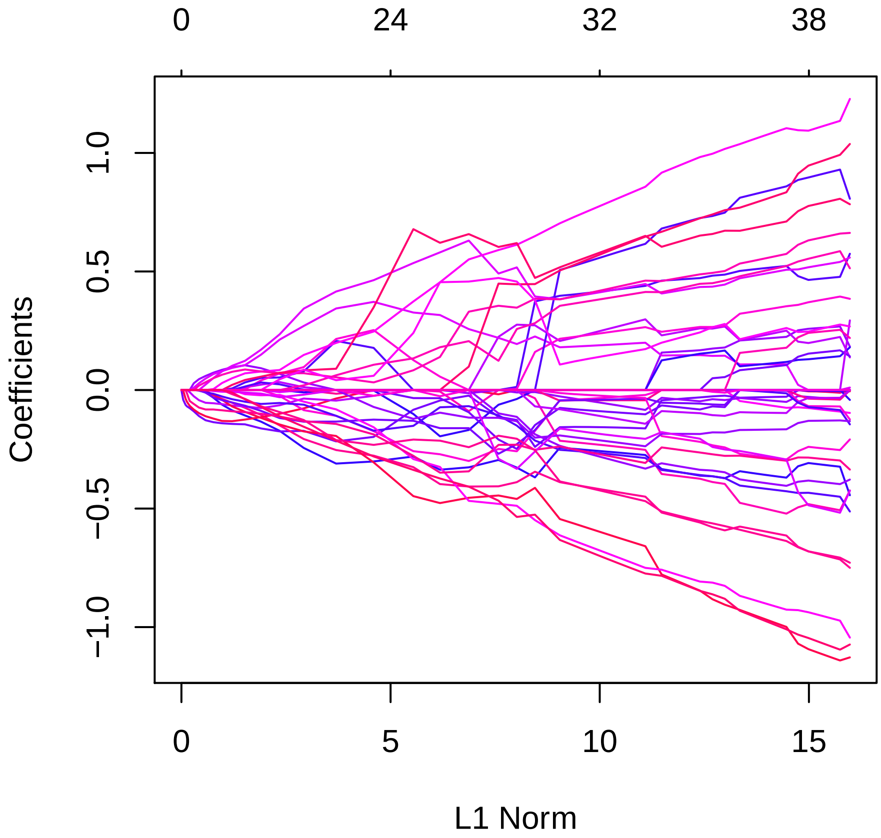



```
cv.probit <- cv.gcdnet(FHT$x, FHT$y, method="probit", pf=pf, pf2=pf2,
                       lambda2=0.01, pred.loss="loss", nfolds=5)
plot(cv.probit)
```

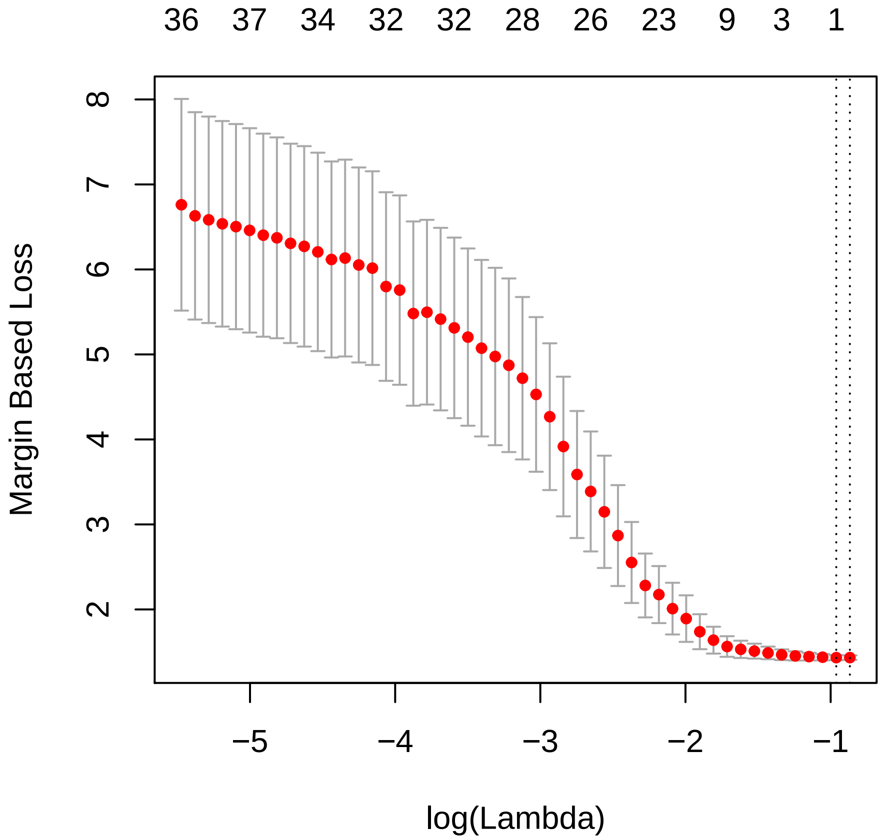


Folded Concave (SCAD) Penalized Probit Regression:

```
lla.probit <- lla.gcdnet(FHT$x, FHT$y, nlambda=50, method="probit")
plot(lla.probit)
```




```
cv.lla.probit <- cv.lla.gcdnet(FHT$x, FHT$y, method="probit",
                               nlambda=50, pred.loss="loss", nfolds=5)
plot(cv.lla.probit)
```



5.3 Logistic Regression

The logistic regression has the logistic regression loss with the expression $L(t) = \log(1+e^{-t})$ and its derivative is $L^{'}(t) = -(1+e^{t})^{-1}$. Its second derivative is bounded by 1/4. So it also satisfies the quadratic majorization condition with $M = 1/4$.

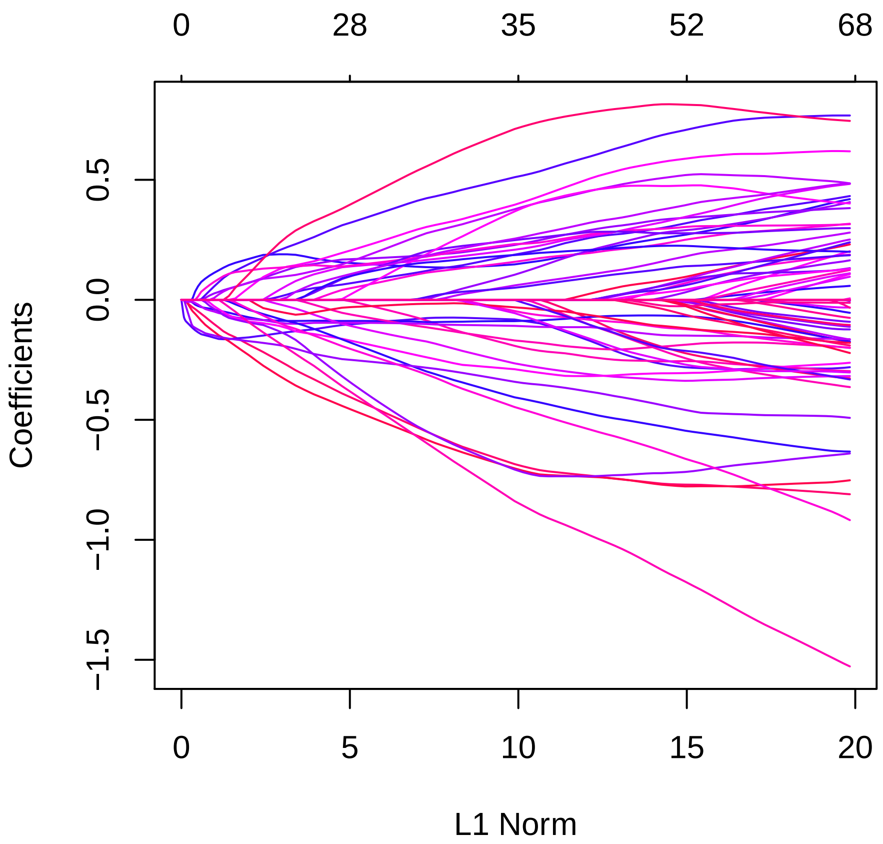
5.3.1 Examples

We only need to specify the method as `method="logit"` in function `gcdnet`, `cv.gcdnet`, `lla.gcdnet` and `cv.lla.gcdnet`.

(Adaptive) Elastic Net Penalized Logistic Regression: set the first three ℓ_1 penalty weights as 10 and the rest are 1; set the last three ℓ_2 penalty weights as 0.1 and the rest are 1; set the ℓ_2 penalty parameter lambda2=0.01.

```
p <- ncol(FHT$x)
# set the first three L1 penalty weights as 10 and the rest are 1.
pf = c(10,10,10,rep(1,p-3))
# set the last three L2 penalty weights as 0.1 and the rest are 1.
pf2 = c(rep(1,p-3),0.1,0.1,0.1)
# set the L2 penalty parameter lambda2=0.01.
m.logit <- gcdnet(x=FHT$x, y=FHT$y, pf=pf, pf2=pf2,
```

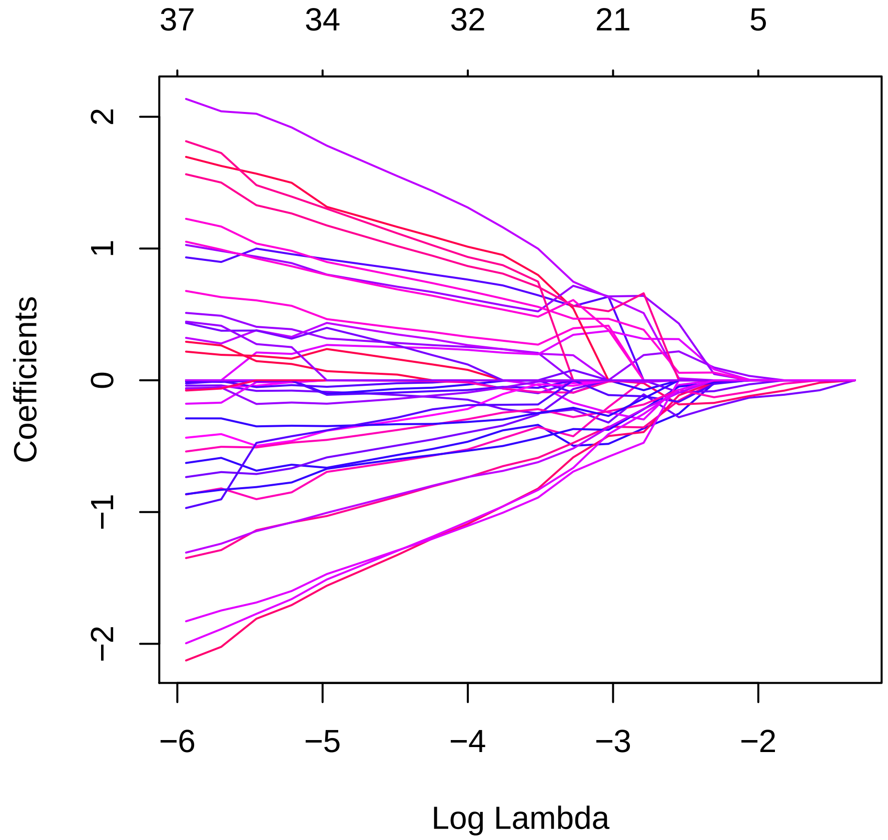
```r
                     lambda2=0.01, method="logit")
plot(m.logit)
```




```r
cv.logit <- cv.gcdnet(FHT$x, FHT$y, method="logit", pf=pf, pf2=pf2,
                     lambda2=0.01, pred.loss="loss", nfolds=5)
```

Folded Concave (SCAD) Penalized Logistic Regression:

```r
lla.logit <- lla.gcdnet(FHT$x, FHT$y, nlambda=20, method="logit")
plot(lla.logit, xvar="lambda")
```




```r
cv.lla.logit <- cv.lla.gcdnet(FHT$x, FHT$y, method="logit",
                              nlambda=20, pred.loss="loss", nfolds=5)
```

5.4 Squared SVM

The squared SVM has a squared hinge loss function with the expression $L(t) = [(1-t)+]^2$ and its derivative $L^{'}(t) = -2(1-t)_+$. Yang and Zou (2013) shows that it satisfies the *quadratic majorization condition* with $M = 4$.

5.4.1 Examples

We only need to specify the method as `method="sqsvm"` in function `gcdnet`, `cv.gcdnet`, `lla.gcdnet` and `cv.lla.gcdnet`.

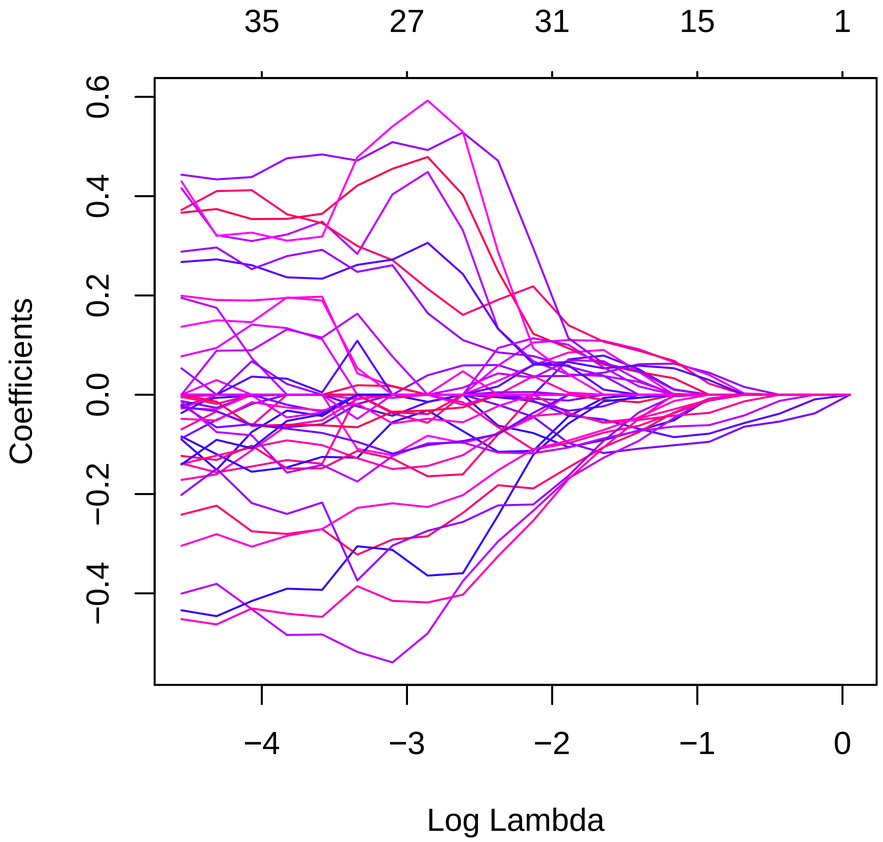
(Adaptive) Elastic Net Penalized Squared Hinge SVM:

```r
# set lambda2 = 0 and meanwhile specify the L1 penalty weights.
p <- ncol(FHT$x)
# set the first three L1 penalty weights as 0.1 and the rest are 1
pf = c(0.1,0.1,0.1,rep(1,p-3))
m.sqsvm <- gcdnet(x=FHT$x, y=FHT$y, pf=pf, lambda2=0.1, method="sqsvm")
```

```
cvm.sqsvm <- cv.gcdnet(x=FHT$x, y=FHT$y, pf=pf, lambda2=0.1, method="sqsvm",
                       pred.loss="loss", nfolds=5)
```

Folded Concave (SCAD) Penalized Squared Hinge SVM:

```
lla.sqsvm <- lla.gcdnet(x=FHT$x, y=FHT$y, nlambda=20, method="sqsvm")
plot(lla.sqsvm, xvar="lambda")
```




```
cv.lla.sqsvm <- cv.lla.gcdnet(FHT$x, FHT$y, method="sqsvm",
                              nlambda=50, pred.loss="loss", nfolds=5)
```

References

Y. Yang and H. Zou, "An efficient algorithm for computing the hhsvm and its generalizations," *Journal of Computational and Graphical Statistics*, vol. 22, no. 2, pp. 396–415, 2013.

J. Fan, L. Xue, and H. Zou, "Strong oracle optimality of folded concave penalized estimation," *Annals of statistics*, vol. 42, no. 3, p. 819, 2014.

L. Wang, J. Zhu, and H. Zou, "Hybrid huberized support vector machines for microarray classification and gene selection," *Bioinformatics*, vol. 24, no. 3, pp. 412–419, 2008.

J. Fan and R. Li, “Variable selection via nonconcave penalized likelihood and its oracle properties,” *Journal of the American statistical Association*, vol. 96, no. 456, pp. 1348–1360, 2001.

H. Zou and R. Li, “One-step sparse estimates in nonconcave penalized likelihood models,” *Annals of statistics*, vol. 36, no. 4, p. 1509, 2008.

J. Friedman, T. Hastie, and R. Tibshirani, “Regularization paths for generalized linear models via coordinate descent,” *Journal of statistical software*, vol. 33, no. 1, p. 1, 2010.

H. Zou and T. Hastie, “Regularization and variable selection via the elastic net,” *Journal of the royal statistical society: series B (statistical methodology)*, vol. 67, no. 2, pp. 301–320, 2005.