# Penalized Regression with R Package

Flexible Penalized Regression with Multiple Penalties and Solvers Algorithm

JuHyun Kang

June 6, 2025

The Three Sisters of Newton
School of Mathematics, Statistics and Data Science
Sungshin Women's University

# Outline

- Penalized regression is a statistical technique that adds a regularization term to the loss function to prevent overfitting and enable variable selection by shrinking regression coefficients

$$\arg\min_{\beta} \left\{ \frac{1}{2n} \sum_{i=1}^{n} \left(y_i - x_i^\top \beta\right)^2 + \sum_{j=1}^{p} P_\lambda(|\beta_j|) \right\}$$

- $y_i$ represents the response variable for observation $i$
- $x_i$ is the vector of predictor variables for observation $i$
- $\beta$ is the vector of regression coefficients
- $P_\lambda(\cdot)$ is a penalty function parameterized by a regularization parameter $\lambda$

# Outline

## Penalized Regression Method

- Ridge

- Lasso

- Elastic Net

- MCP (Minimax Concave Penalty)

- SCAD (Smoothly Clipped Absolute Deviation)

## Ridge Regression

- Uses an L2 norm penalty to shrink coefficients and reduce multicollinearity, though it does not perform variable selection

$$P_\lambda(\beta_j) = \frac{\lambda}{2}\beta_j^2$$

- The ridge objective function is strictly convex, which guarantees a unique global minimum and makes it well-suited for convex optimization methods

- Ridge regression has a closed-form solution given by:

$$\hat{\beta}_{\text{ridge}} = (X^\top X + \lambda I)^{-1}X^\top Y$$

# Lasso Regression

- Lasso regression uses an L1 norm penalty, which encourages sparsity by driving some coefficients exactly to zero, thereby performing effective variable selection

$$P_\lambda(\beta_j) = \lambda|\beta_j|$$

- The lasso objective function is convex but not strictly convex, which means it can have multiple solutions, especially when predictors are highly correlated

- The penalty term is not differentiable at $\beta_j = 0$, which requires specialized optimization algorithms such as coordinate descent or subgradient methods

- Elastic net combines the L1 and L2 penalties from lasso and ridge regression

$$P_\lambda(\beta_j) = \lambda \left( \alpha|\beta_j| + \frac{1-\alpha}{2}\beta_j^2 \right)$$

- The elastic net objective function is convex. Under certain conditions, it guarantees a unique global minimum

- Elastic net is particularly effective when predictors are correlated, as it encourages a grouping effect while maintaining sparsity

  - $\alpha = 1$: equivalent to lasso

  - $\alpha = 0$: equivalent to ridge

  - $0 < \alpha < 1$: elastic net (a mixture of both)

## Minimax Concave Penalty Regression (MCP)

- The minimax concave penalty (MCP) is a non-convex penalty function that aims to reduce estimation bias for large coefficients while maintaining sparsity

$$P_\lambda(\beta_j) = \begin{cases} \lambda|\beta_j| - \dfrac{\beta_j^2}{2\gamma}, & \text{if } |\beta_j| \leq \gamma\lambda, \\ \dfrac{\gamma\lambda^2}{2}, & \text{if } |\beta_j| > \gamma\lambda. \end{cases}$$

- The parameter $\gamma$ controls the degree of concavity and non-linearity: As $\gamma$ increases, the MCP penalty approaches the lasso penalty

# Smoothing Clipped Absolute Deviation Regression (SCAD)

- The smoothly clipped absolute deviation (SCAD) penalty is a non-convex function designed to reduce the estimation bias of large coefficients while preserving sparsity, addressing the limitations of the lasso

$$P_\lambda(\beta_j) = \begin{cases} \lambda|\beta_j|, & \text{if } |\beta_j| \leq \lambda, \\ \dfrac{-|\beta_j|^2 + 2a\lambda|\beta_j| - \lambda^2}{2(a-1)}, & \text{if } \lambda < |\beta_j| \leq a\lambda, \\ \dfrac{(a+1)\lambda^2}{2}, & \text{if } |\beta_j| > a\lambda. \end{cases}$$

- The parameter $a$ controls the concavity of the penalty: commonly, $a = 3.7$ is recommended

# Outline

## Coordinate Descent Algorithm (CDA)

- Coordinate Descent is an iterative optimization algorithm that updates one parameter at a time while keeping the others fixed

- It is particularly efficient for problems where each coordinate update has a closed-form solution, such as in Lasso and Elastic Net regressions

- For the following objective,

$$\arg\min_{\beta} \left\{ \frac{1}{2n} \sum_{i=1}^{n} \left(y_i - x_i^\top \beta\right)^2 + \sum_{j=1}^{p} P_\lambda(|\beta_j|) \right\}$$

the coordinate-wise update step is given by:

$$\beta_j^{(k+1)} \leftarrow \arg\min_{\beta_j} \left\{ \frac{1}{2n} \sum_{i=1}^{n} \left(y_i - x_{ij}\beta_j - \sum_{k \neq j} x_{ik}\beta_k^{(k+1)}\right)^2 + P_\lambda(|\beta_j|) \right\}$$

## Fast Iterative Soft-Thresholding Algorithm (FISTA)

- Fast iterative soft-thresholding algorithm (FISTA) is an accelerated version of the proximal gradient descent method and is designed to solve optimization problems with non-smooth penalties such as the Lasso

- It achieves faster convergence compared to the standard iterative soft-thresholding algorithm

- FISTA is widely used in sparse regression models, including Lasso and Elastic Net, due to its efficiency and simplicity

# FISTA Algorithm with Objective function

- Consider the following objective function:

$$\min_{\beta}\{f(\beta) + P_\lambda(\beta)\}$$

  where $f(\beta)$ is convex and differentiable, and $P_\lambda(\beta)$ is convex but possibly non-smooth

- FISTA updates proceed as follows:

$$\beta_{k+1} = \text{prox}_{\eta P_\lambda}\left(y^k - \eta\nabla f(y^k)\right),$$

$$t_{k+1} = \frac{1 + \sqrt{1 + 4t_k^2}}{2},$$

$$y^{k+1} = \beta_{k+1} + \frac{t_k - 1}{t_{k+1}}\left(\beta_{k+1} - \beta_k\right),$$

  where $\text{prox}_{\eta P_\lambda}$ is the proximal operator (often implemented as a soft-thresholding function when $P_\lambda$ is the $\ell_1$ norm as in the Lasso)

## Local Linear Approximation Algorithm (LLA)

- The Local Linear Approximation (LLA) algorithm is used to handle non-convex penalties, such as SCAD and MCP, by approximating the penalty function locally with a linear function

- This transforms the original non-convex optimization problem into a series of convex problems, which are easier to solve

- LLA is known to achieve desirable statistical properties, including the oracle property

- For example, the SCAD penalty can be locally approximated at the $k$-th iteration as follows:

$$P_\lambda(|\beta_j|) \approx P_\lambda(|\beta_j^{(k)}|) + P_\lambda'(|\beta_j^{(k)}|)(|\beta_j| - |\beta_j^{(k)}|)$$

- Hence, the optimization problem at iteration $k+1$ becomes:

$$\min_\beta \left\{ \frac{1}{2n} \sum_{i=1}^n \left( y_i - x_i^\top \beta \right)^2 + \sum_{j=1}^p w_j^{(k)} |\beta_j| \right\}$$

where

$$w_j^{(k)} = P_\lambda'(|\beta_j^{(k)}|)$$

## Outline

```r
penalized_regression <- function(X, y,
                                 method = c("lasso", "ridge",
                                            "scad", "mcp", "elasticnet"),
                                 algorithm = c("cda", "fista", "lla"),
                                 lambda = 1, learning_rate = 0.01,
                                 max_iter = 1000, alpha = 0.5, gamma = 3.7) {
  method <- match.arg(method)
  algorithm <- match.arg(algorithm)

  check_algorithm_compatibility(method, algorithm)
```

```r
  if (algorithm == "cda") {
    return(perform_CDA(X, y, method, lambda, learning_rate,
                       max_iter, alpha, gamma))
  } else if (algorithm == "fista") {
    return(perform_FISTA(X, y, method, lambda, learning_rate,
                         max_iter, alpha, gamma))
  } else if (algorithm == "lla") {
    return(perform_LLA(X, y, method, lambda, learning_rate,
                       max_iter, gamma))
  } else {
    stop("Unknown algorithm selected.")
  }
}
```

# Check Algorithm Compatability Function

```r
# check algorithm
check_algorithm_compatibility <- function(method, algorithm) {
  method <- tolower(method)
  algorithm <- tolower(algorithm)

  if (method == "ridge" && algorithm == "cda") {
    warning("CDA may not be the most appropriate choice for Ridge penalty.")
  }
  if (method %in% c("scad", "mcp") && algorithm == "fista") {
    stop("FISTA is not recommended for non-convex penalties
          like SCAD or MCP.")
  }
  if (!method %in% c("scad", "mcp") && algorithm == "lla") {
    warning("LLA is primarily designed for SCAD or MCP penalties and
             may not be optimal for Ridge, Lasso, or Elastic Net.")
  }
}
```

## Coordinate Descent Algorithm (CDA) Function

```
perform_CDA <- function(X, y, method, lambda, learning_rate = 0.01,
                        max_iter = 1000, alpha = 0.5, gamma = 3.7) {
  n <- nrow(X)
  p <- ncol(X)
  beta <- rep(0, p)
  Xy <- t(X) %*% y
  XX <- colSums(X^2)

  soft_threshold <- function(z, t) {
    sign(z) * pmax(0, abs(z) - t)
  }

  for (iter in 1:max_iter) {
    for (j in 1:p) {
      r_j <- y - X %*% beta + X[, j] * beta[j]
      rho_j <- sum(X[, j] * r_j)
```

# CDA Function - Penalty Updates (Part 1)

```
if (method == "lasso") {
  beta[j] <- soft_threshold(rho_j / XX[j], lambda / XX[j])


} else if (method == "ridge") {
  beta[j] <- rho_j / (XX[j] + 2 * lambda)


} else if (method == "elasticnet") {
  z <- rho_j / XX[j]
  beta[j] <- soft_threshold(z, lambda * alpha / XX[j]) /
    (1 + lambda * (1 - alpha) / XX[j])


} else if (method == "scad") {
  z <- rho_j / XX[j]
  if (abs(z) <= lambda) {
    beta[j] <- soft_threshold(z, lambda / XX[j])
  } else if (abs(z) <= gamma * lambda) {
    beta[j] <- soft_threshold(z, gamma * lambda / (gamma - 1) / XX[j])
  } else {
    beta[j] <- z}
```

# CDA Function - Penalty Updates (Part 2)

```r
    } else if (method == "mcp") {
      z <- rho_j / XX[j]
      abj <- abs(z)

      if (abj <= gamma * lambda) {
        beta[j] <- soft_threshold(z, lambda / XX[j]) / (1 - 1 / gamma)
      } else {
        beta[j] <- z
      }

    } else {
      stop("Unsupported method in CDA.")
    }
  }
}
return(beta)
}
```

# Fast Iterative Shrinkage-Thresholding Algorithm (FISTA) Function

```r
perform_FISTA <- function(X, y, method, lambda, learning_rate = 1e-3,
                          max_iter = 1000, alpha = 0.5, gamma = 3.7) {
  n <- nrow(X)
  p <- ncol(X)

  beta <- rep(0, p)
  beta_old <- beta
  t <- 1

  soft_threshold <- function(z, t) {
    sign(z) * pmax(0, abs(z) - t)
  }

  grad <- function(beta) {
    -t(X) %*% (y - X %*% beta) / n
  }
```

```r
penalty_grad <- function(beta_j) {
  if (method == "lasso") {
    return(lambda * sign(beta_j))

  } else if (method == "ridge") {
    return(2 * lambda * beta_j)

  } else if (method == "elasticnet") {
    return(lambda * (alpha * sign(beta_j) + 2 * (1 - alpha) * beta_j))

  } else if (method == "scad") {
    abj <- abs(beta_j)
    if (abj <= lambda) {
      return(lambda * sign(beta_j))
    } else if (abj <= gamma * lambda) {
      return(((gamma * lambda - abj) / (gamma - 1)) * sign(beta_j))
    } else {
      return(0)
    }
```

```r
    } else if (method == "mcp") {
      abj <- abs(beta_j)
      if (abj <= gamma * lambda) {
        return(lambda * (1 - abj / (gamma * lambda)) * sign(beta_j))
      } else {
        return(0)
      }
    } else {
      stop("Unsupported method.")
    }
  }


  for (k in 1:max_iter) {
    z <- beta + ((t - 1) / (t + 2)) * (beta - beta_old)
    grad_z <- grad(z)

    beta_new <- numeric(p)
```

# FISTA Function - Updating Coefficients

```
for (j in 1:p) {
  if (method == "ridge") {
    beta_new[j] <- z[j] - learning_rate * (grad_z[j] + 2 * lambda * z[j])

  } else if (method %in% c("lasso", "elasticnet")) {
    # In elasticnet, threshold value is learning_rate * lambda * alpha
    thresh <- learning_rate * lambda * alpha

    # Soft thresholding Gradient step
    beta_new[j] <- soft_threshold(z[j] - learning_rate * grad_z[j],
                                  thresh)

  } else if (method %in% c("scad", "mcp")) {
    # Compute threshold: learning_rate * |penalty gradient
    pen_grad_val <- penalty_grad(z[j])
    thresh <- learning_rate * abs(pen_grad_val)

    # If the threshold is 0, soft-thresholding has no effect
    if (thresh < 1e-12) thresh <- 0
```

```r
        beta_new[j] <- soft_threshold(z[j] - learning_rate * grad_z[j],
                                      thresh)

    } else {
      stop("Unsupported method in FISTA.")
    }
  }

  beta_old <- beta
  beta <- beta_new
  t <- t + 1

  # Optional: Check for convergence - stop if changes are small
  if (sqrt(sum((beta - beta_old)^2)) < 1e-6) break
  }

  return(beta)
}
```

## Local Linear Approximation (LLA) Function

```r
perform_LLA <- function(X, y, method, lambda, learning_rate = 0.01,
                        max_iter = 100, alpha = 0.5, gamma = 3.7) {
  n <- nrow(X)
  p <- ncol(X)
  beta <- rep(0, p)
  tol <- 1e-4

  for (iter in 1:max_iter) {
    weights <- rep(1, p)

    for (j in 1:p) {
      bj <- beta[j]

      if (method == "lasso") {
        weights[j] <- 1
      } else if (method == "ridge") {
        weights[j] <- 2 * abs(bj)
      } else if (method == "elasticnet") {
        weights[j] <- alpha + 2 * (1 - alpha) * abs(bj)
```

```r
} else if (method == "scad") {
  abj <- abs(bj)
  if (abj <= lambda) {
    weights[j] <- 1
  } else if (abj <= gamma * lambda) {
    weights[j] <- (gamma * lambda - abj) / ((gamma - 1) * lambda)
  } else {
    weights[j] <- 0
  }
} else if (method == "mcp") {
  abj <- abs(bj)
  if (abj <= gamma * lambda) {
    weights[j] <- 1 - abj / (gamma * lambda)
  } else {
    weights[j] <- 0
  }
} else {
  stop("Unsupported method in LLA.")
```

```r
      }
    }

    for (j in 1:p) {
      r_j <- y - X %*% beta + X[, j] * beta[j]
      rho_j <- sum(X[, j] * r_j)
      XX_j <- sum(X[, j]^2)
      beta[j] <- sign(rho_j) * max(0, abs(rho_j) - lambda * weights[j]) /
        XX_j
    }
  }

  return(beta)
}
```

# Outline

# Boston Dataset for Penalize Regression

```r
library(MASS)
data("Boston")

# response variable (medv)
y <- scale(Boston[, ncol(Boston)])

# predict variables
X <- scale(Boston[, -ncol(Boston)])
head(Boston,3)

##      crim zn indus chas   nox    rm  age    dis rad tax ptratio  black lstat
## 1 0.00632 18  2.31    0 0.538 6.575 65.2 4.0900   1 296    15.3 396.90  4.98
## 2 0.02731  0  7.07    0 0.469 6.421 78.9 4.9671   2 242    17.8 396.90  9.14
## 3 0.02729  0  7.07    0 0.469 7.185 61.1 4.9671   2 242    17.8 392.83  4.03
##   medv
## 1 24.0
## 2 21.6
## 3 34.7
```

# Ridge Regression with Boston Dataset

```
penalized_regression(X, y, method="ridge", algorithm="cda")
```

```
## [1] -0.0992520661  0.1144912234  0.0103660543  0.0749141210 -0.2178957342
## [6]  0.2928666357  0.0007757563 -0.3320626430  0.2747593206 -0.2120429740
## [11] -0.2225045953  0.0923399477 -0.4046479170
```

```
penalized_regression(X, y, method="ridge", algorithm="fista")
```

```
## [1] -0.04852125  0.04015512 -0.05150618  0.04776459 -0.04236857  0.16056307
## [7] -0.03079502 -0.02038188 -0.02068400 -0.04793745 -0.09603274  0.04874783
## [13] -0.14758457
```

```
penalized_regression(X, y, method="ridge", algorithm="lla")
```

```
## Warning in check_algorithm_compatibility(method, algorithm): LLA is primarily
## designed for SCAD or MCP penalties and may not be optimal for Ridge, Lasso, or
## Elastic Net.
```

```
## [1] -0.0992520661  0.1144912234  0.0103660543  0.0749141210 -0.2178957342
## [6]  0.2928666357  0.0007757563 -0.3320626430  0.2747593206 -0.2120429740
## [11] -0.2225045953  0.0923399477 -0.4046479170
```

## Lasso Regression with Boston Dataset

```
penalized_regression(X, y, method="lasso", algorithm="cda")
```

```
## [1] -0.09547432  0.10913508  0.00000000  0.07444264 -0.21027610  0.29354573
## [7]  0.00000000 -0.32754924  0.25576045 -0.19334872 -0.22034164  0.09053869
## [13] -0.40569871
```

```
penalized_regression(X, y, method="lasso", algorithm="fista")
```

```
## [1]  0.00000000  0.00000000  0.00000000  0.00000000  0.00000000  0.07858555
## [7]  0.00000000  0.00000000  0.00000000  0.00000000  0.00000000  0.00000000
## [13] -0.18852509
```

```
penalized_regression(X, y, method="lasso", algorithm="lla")
```

```
## Warning in check_algorithm_compatibility(method, algorithm): LLA is primarily
## designed for SCAD or MCP penalties and may not be optimal for Ridge, Lasso, or
## Elastic Net.
```

```
## [1] -0.09547432  0.10913508  0.00000000  0.07444264 -0.21027610  0.29354573
## [7]  0.00000000 -0.32754924  0.25576045 -0.19334872 -0.22034164  0.09053869
## [13] -0.40569871
```

# Elastic Net Regression with Boston Dataset

```r
penalized_regression(X, y, method="elasticnet", algorithm="cda")
```

```
## [1] -0.09785179  0.11242987  0.00488148  0.07460145 -0.21492289  0.29276585
## [7]  0.00000000 -0.33182677  0.26841639 -0.20522069 -0.22168907  0.09146410
## [13] -0.40552475
```

```r
penalized_regression(X, y, method="elasticnet", algorithm="fista")
```

```
## [1]  0.00000000  0.00000000  0.00000000  0.00000000  0.00000000  0.07858555
## [7]  0.00000000  0.00000000  0.00000000  0.00000000  0.00000000  0.00000000
## [13] -0.18852509
```

```r
penalized_regression(X, y, method="elasticnet", algorithm="lla")
```

```
## Warning in check_algorithm_compatibility(method, algorithm): LLA is primarily
## designed for SCAD or MCP penalties and may not be optimal for Ridge, Lasso, or
## Elastic Net.
```

```
## [1] -0.08329111  0.09568474  0.00000000  0.07188826 -0.19734098  0.29783985
## [7]  0.00000000 -0.30657550  0.20645410 -0.15278811 -0.21687523  0.08604270
## [13] -0.41149174
```

# MCP Regression with Boston Dataset

```
penalized_regression(X, y, method="mcp", algorithm="cda")
```

```
## [1] -1.3176462  3.7210956  2.9544747 -0.4952345 -4.0229037 -2.8002490
## [7]  2.6712131 -4.8408153  4.1973330 -4.5548273 -0.6704180 -0.5486739
## [13] -4.5074292
```

```
penalized_regression(X, y, method="mcp", algorithm="fista")
```

```
## Error in check_algorithm_compatibility(method, algorithm): The FISTA is not
##  recommended for non-convex penalties like SCAD or MCP.
```

```
penalized_regression(X, y, method="mcp", algorithm="lla")
```

```
## [1] -0.09571331  0.10948056  0.00000000  0.07439100 -0.21082761  0.29330031
## [7]  0.00000000 -0.32850565  0.25738882 -0.19467159 -0.22050792  0.09054474
## [13] -0.40597901
```

# SCAD Regression with Boston Dataset

```r
penalized_regression(X, y, method="scad", algorithm="cda")
```

```
## [1] -0.09547432  0.10913508  0.00000000  0.07444264 -0.21027610  0.29354573
## [7]  0.00000000 -0.32754924  0.25576045 -0.19334872 -0.22034164  0.09053869
## [13] -0.40569871
```

```r
penalized_regression(X, y, method="scad", algorithm="fista")
```

```
## Error in check_algorithm_compatibility(method, algorithm): The FISTA is not
##   recommended for non-convex penalties like SCAD or MCP.
```

```r
penalized_regression(X, y, method="scad", algorithm="lla")
```

```
## [1] -0.09547432  0.10913508  0.00000000  0.07444264 -0.21027610  0.29354573
## [7]  0.00000000 -0.32754924  0.25576045 -0.19334872 -0.22034164  0.09053869
## [13] -0.40569871
```

# Q & A

# Thank you :)