SDS 385: Homework 5

G. Paulon

October 19, 2016

Problem 1. Penalized likelihood and soft thresholding

(A) The fist kind of penalized log-likelihood we analyze is called the **soft thresholding**. Let us define the function

$$S_\lambda(y) = \underset{\theta}{\operatorname{argmin}} \frac{1}{2}(y-\theta)^2 + \lambda|\theta|.$$

The intuition here is that θ is a parameter of a statistical model, and y is data. The first term rewards good fit to the data, while the second term rewards θ for being "simpler" (i.e. closer to zero). It is evident that the first term is proportional to the negative log-likelihood of a Gaussian distribution. In fact, if $y \sim \mathcal{N}(\theta, 1)$, the negative likelihood can be written as

$$L(\theta) = -\frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}(y-\theta)^2}$$

and, by taking the logarithm,

$$l(\theta) = K + \frac{1}{2}(y-\theta)^2.$$

To minimize the objective function, we can compute the derivative of the negative log-likelihood (which is convex) and setting it equal to 0 as usual, obtaining

$$\frac{\partial l(\hat{\theta})}{\partial \theta} = -(y-\hat{\theta}) + \lambda \operatorname{sign}(\hat{\theta}) = 0.$$

We can split the three cases:

- if $\hat{\theta} < 0$, then $\hat{\theta} = y + \lambda$ and the constraint becomes $y > \lambda$;
- if $\hat{\theta} > 0$, then $\hat{\theta} = y - \lambda$ and the constraint becomes $y < -\lambda$;
- if $\hat{\theta} = 0$, then $|y| < \lambda$.

These three cases can be summarized as

$$S_\lambda(y) = \hat{\theta} = \operatorname{sign}(y)(|y| - \lambda)_+$$

(B) Suppose we observe data from the following statistical model:

$$z_i|\theta_i) \sim \mathcal{N}(\theta_i, \sigma_i^2). \tag{1}$$

That is, there are n different means θ_i, and we observe 1 normally distributed observation for each one. We allow that each observation has a different variance σ_i^2; for now we will assume these are known. This is called the Gaussian sequence model, or the normal-means problem. It looks like a toy problem, but is surprisingly useful in a wide variety of applications, from curve fitting to image denoising to genome-wise association studies.

Now suppose we believe that a lot of the θ_i's are zero, i.e. that the vector $\boldsymbol{\theta} = (\theta_1, \ldots, \theta_n)^T$ is sparse. Consider an estimator for each θ_i of the form

$$\hat{\theta}(y_i) = S_{\lambda\sigma_i^2}(y_i),$$

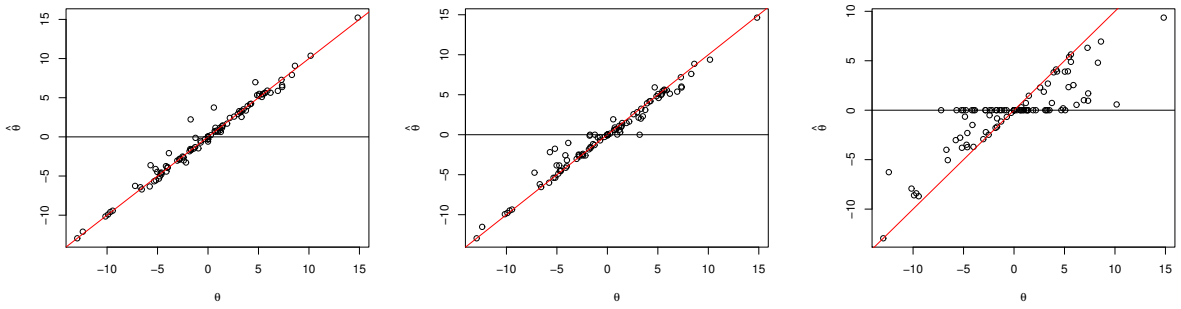
where S is the soft thresholding operator defined above with parameter $\lambda\sigma_i^2$. Here the parameter of the estimator is $\lambda\sigma_i^2$ because the objective function of this normal model is now

$$\frac{1}{2}\left(\frac{y-\theta}{\sigma_i}\right)^2 + \lambda|\theta| = \frac{1}{2}(y-\theta)^2 + \lambda\sigma_i^2|\theta|.$$

We simulate $n = 100$ data points from this model with parameter $\boldsymbol{\theta}$. Each component θ_i of the parameter vector and each σ_i are sampled according to

$$\theta_i \overset{\text{iid}}{\sim} \mathcal{N}(0, 5^2)$$
$$\sigma_i^2 \overset{\text{iid}}{\sim} \text{Gamma}(0.5, 1).$$

The vector $\boldsymbol{\theta}$ is successively made sparse by masking some of its elements. The data y_i are generated according to the model (1) and the algorithm is then tested against different levels of sparsity K and against different values of λ.



(a) $\lambda = 0$ *(b)* $\lambda = 1$ *(c)* $\lambda = 10$

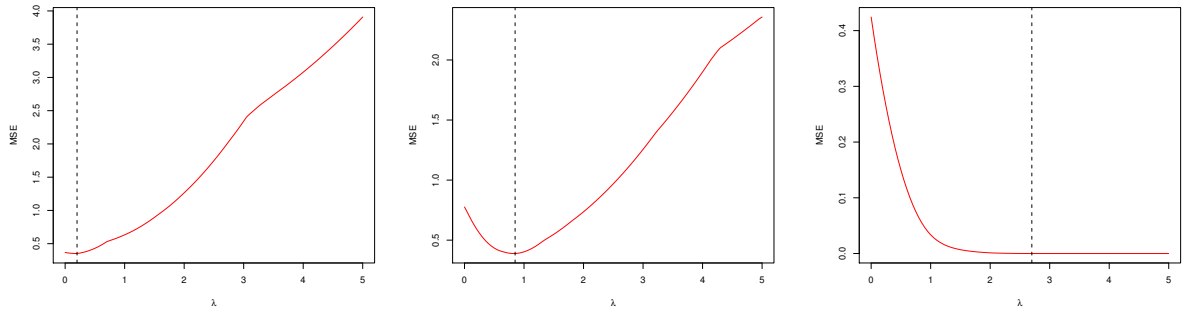
Figure 1: Fitted value plotted against the original data for increasing levels of the penalization parameter λ.

We compute $\hat{\theta}(y_i) = S_{\lambda\sigma_i^2}(y_i)$ across a discrete grid of different λ values. In Figure 1 we plot $\hat{\theta}(y_i)$ versus θ_i. As one can see, by increasing the level of sparsity, the soft thresholding estimator selects more and more θ_i's and shrinks the nonzero estimates towards 0.

Moreover, we compute the mean-squared error of the estimate, pretending we knew the real values θ_i as a function of λ:

$$\text{MSE}(\lambda) = \frac{1}{n}\sum_{i=1}^{n}(\hat{\theta}(y_i) - \theta_i)^2.$$

In Figure 2, the optimal value of λ is displayed for different choices of the sparsity level K. As one can see, for highly sparse parameter vectors, the optimal penalization parameter is higher. For small value of sparsity, instead, λ is approximately equal to 0 because there is no need to force sparsity into the model.



(a) $K = 0$ *(b)* $K = 0.5$ *(c)* $K = 1$

Figure 2: MSE computed on a grid of λ *parameters as the sparsity of the parameters vector increases. In dashed vertical line, the optimal value of* λ *is indicated.*

Problem 2. The lasso

Consider the standard linear regression model

$$\boldsymbol{y} = X\boldsymbol{\beta} + \boldsymbol{\varepsilon},$$

where $\boldsymbol{y}$ is an n-vector of responses, X is an $n \times p$ features matrix whose i^{th} row $\boldsymbol{x}_i$ is the vector of features for observation i, and $\boldsymbol{\varepsilon}$ is a vector of errors/residuals. The lasso involves estimating $\boldsymbol{\beta}$ as the solution to the penalized least-squares problem

$$\hat{\boldsymbol{\beta}} = \underset{\boldsymbol{\beta}}{\operatorname{argmin}} \frac{1}{2}||\boldsymbol{y} - X\boldsymbol{\beta}||_2^2 + \lambda||\boldsymbol{\beta}||_1$$

As in in the normal means problem, this penalty function will have the effect of both selecting a set of nonzero β_i's (i.e. sparsifying the estimate) as well as shrinking the nonzero β_i's toward 0. The bigger λ, the more aggressive the shrinkage effect.

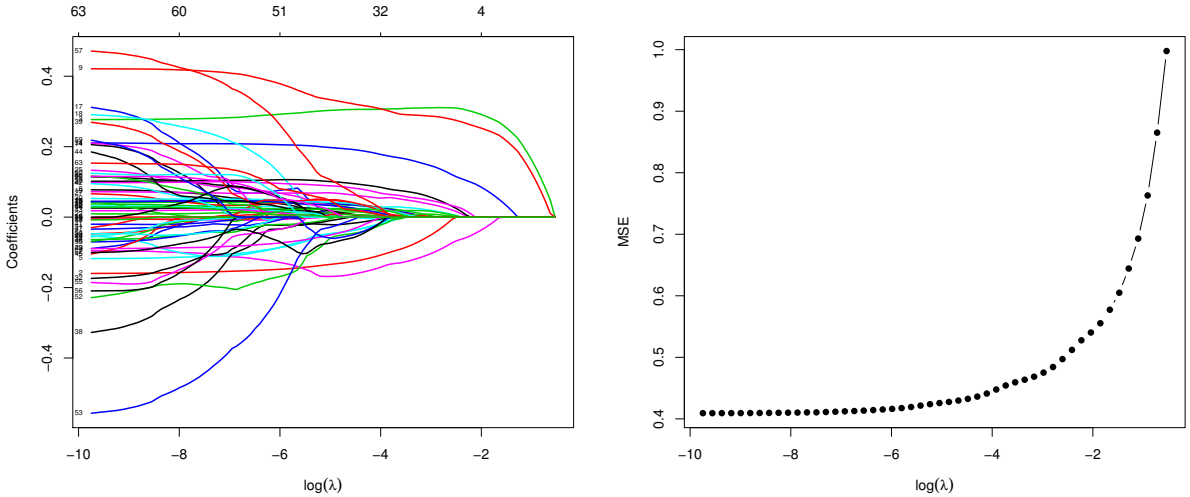
(A) We fit a Lasso regression model on the diabetes progression dataset, which consists in $n = 442$ patients with $p = 64$ associated covariates. Both the response variable $\boldsymbol{y}$ and all columns of the predictor matrix X have been standardized and have a mean of 0, therefore there is no need for an explicit intercept.

We fit the lasso model across a range of λ values and, in Figure 3, we plot the solution path $\hat{\boldsymbol{\beta}}_\lambda$ as a function of λ. As one can see, as the value of λ increases, more and more parameters get shrunk to 0.

In addition, we track the in-sample mean-squared prediction error of the fit across the solution path:

$$\text{MSE}(\hat{\boldsymbol{\beta}}_\lambda) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \boldsymbol{x}_i^T\hat{\boldsymbol{\beta}}_\lambda)^2.$$

The MSE is always increasing as a function of λ and its minimum is reached for the model with all the covariates included. This is not surprising: in fact, the unpenalized model is the



(a) Trajectories of the Lasso regression. *(b) MSE of the Lasso regression.*

Figure 3: Trajectories and MSE of the Lasso regression for 50 different values of λ.

best one in explaining the effect of the covariates on the **training** set. However, it might be overfitting the data, as we will discuss in detail in the next sections.

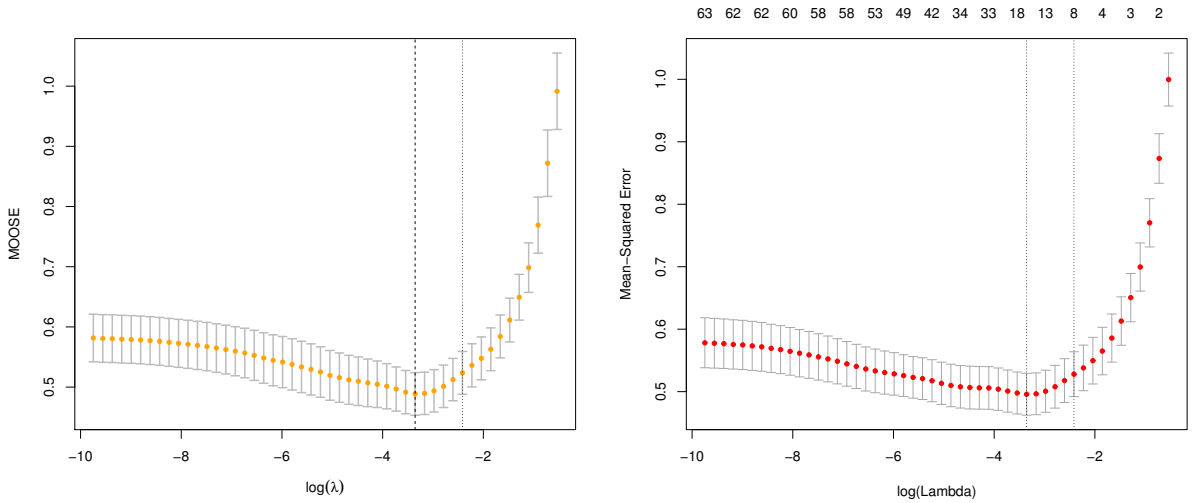
(B) A natural way to choose λ is to minimize the expected out-of-sample prediction error. Suppose that $(\boldsymbol{x}_*, y_*)$ is a future data point from the same population/data-generating process as the original data. The goal would be to make the expected error

$$\text{MOOSE}(\hat{\boldsymbol{\beta}}_\lambda) = E\left[(y_* - \boldsymbol{x}_*^T \hat{\boldsymbol{\beta}}_\lambda)^2\right]$$

as small as possible. Of course, we do not have any "future data" lying around, so we have to estimate this quantity using the data we have. The in-sample mean-squared error, $\text{MSE}(\hat{\boldsymbol{\beta}}_\lambda)$ is generally an optimistic estimate of this quantity: out-of-sample error tends to be worse, on average, than in-sample error, and we need some way of quantifying how much worse. In the following we estimate the MOOSE using **cross validation**.

In Figure 4, the plot of the cross-validated estimate of MOOSE across the solution path, as a function of λ, is displayed. As one can see, this figure is very different from the one reporting the behaviour of the MSE estimated on the training data. In fact, a minimum can be found inside the interval of possible values of λ. If we consider the optimal model as the one satisfying the minimum of the MOOSE, we end up choosing $\hat{\lambda} = 0.03489$. However, another approach is preferable. Since we always try to seek parsimonious models (that is, with high values of λ, or with the least number of covariates), we can use the 1 std. dev. criterion. This consists in choosing the most conservative model whose estimated mean MOOSE is within one standard deviation from the mean MOOSE of the optimal model. This approach leads to $\tilde{\lambda} = 0.08931$. In Figure 4, we plot two vertical lines corresponding to the two model choices.

(C) However, cross-validation isn't the only way to estimate generalization error. Another such



(a) Our function. *(b) R's builtin function.*

Figure 4: Comparison of the MOOSE using our implemented function and R's built-in function from the package `glmnet`*. The two vertical lines correspond to the optimal model and to the most conservative optimal model (*1 *std. dev. criterion).*

way is called the C_p statistic. For a linear regression model, the C_p statistic is defined as

$$C_p(\hat{\boldsymbol{\beta}}_\lambda) = \text{MSE}(\hat{\boldsymbol{\beta}}_\lambda) + 2\frac{s_\lambda}{n}\hat{\sigma}^2$$

where s_λ is the degrees of freedom of the fit (i.e. the number of nonzero parameters selected at that particular value of λ), and $\hat{\sigma}^2 = \text{Var}(\varepsilon)$ is an estimate of the residual variance. You can interpret the C_p statistic as the in-sample mean-squared error, plus a penalty for in-sample optimism.

We compute (and plot) the C_p statistic across the solution path, as a function of λ.

In Figure 5, a comparison between the three indexes is shown. As one can see, the MOOSE and the C_p achieve the minimum at the same exact value of lambda (it is not always the case, since the MOOSE involves some uncertainty). Moreover, let us remark that for high values of lambda, when the degrees of freedom of the model are approximately 0, the three approaches yield to the same values (there is no penalty for being a large model in C_p).

The downside of C_p is that it will not work if the model is wrong. In the cross-validation, instead, there is a source of **bias**: we estimate the generalization error of n data points using the generalization error of $(K-1)n/K$ data points. If we use a n-fold cross-validation, there is a contribution of **variance**: the sets are highly correlated and they look similar. Therefore a test set might be really different.

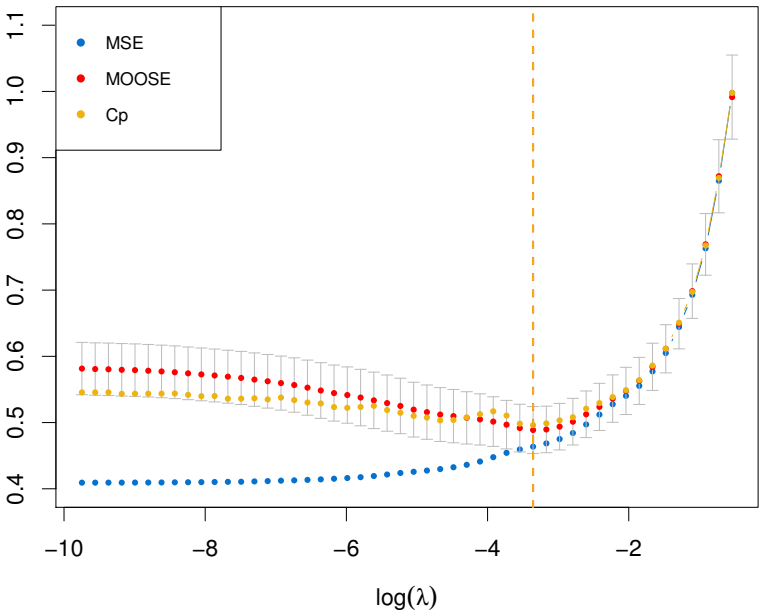


Figure 5: Comparison between the three proposed approach. The dashed red and yellow lines represent the minimum for the MOOSE and for C_p, respectively.

Appendix A

R code

```r
### PENALIZED LIKELIHOOD AND SOFT THRESHOLDING
# (B)

n <- 100 # number of data points
K <- 0.5 # Sparsity level

# Generate the parameters theta
theta <- rnorm(n, 0, 5)
mask <- rbinom(n, 1, K)
theta <- theta * mask
# Generate the variance of the model
sigma2 <- rgamma(n, shape = 0.5, scale = 1)

# Generate the data
y <- rnorm(n, theta, sigma2)

lambda <- 0 # Try several values of lambda
theta.hat <- S.lambda(y, sigma2*lambda)

# See how lambda affects the solution
par(mar=c(4,4.5,2,2), cex = 1.5)
plot(theta, theta.hat, xlab = bquote(theta), ylab=bquote(hat(theta)), pch = 1, lwd = 2)
abline(a = 0, b = 1, lwd = 2, col='red')
abline(h=0)

# Compute the minimum of the MSE for different values of the sparsity K
lambda.grid <- seq(0, 5, by = 0.05)
MSE <- array(NA, dim = length(lambda.grid))
for (i in 1:length(lambda.grid)){
  theta.hat <- S.lambda(y, sigma2*lambda.grid[i])
  MSE[i] <- mean((theta.hat - theta)^2)
}
par(mar=c(4,4.5,2,2), cex = 1.2)
plot(lambda.grid, MSE, type = 'l', lwd = 2, col='red', xlab = bquote(lambda))
abline(v=lambda.grid[which.min(MSE)], lwd = 2, lty = 2)
```

Listing A.1: Function caller for exercise 1.

```r
### THE LASSO
# Load and standardize the data
```

```
y <- as.numeric(read.csv(file = '../SDS385-master/data/diabetesY.csv', header = F)[,1])
X <- as.matrix(read.csv(file = '../SDS385-master/data/diabetesX.csv', header = T))
X <- scale(X)
y <- scale(y)
n <- dim(X)[1]
p <- dim(X)[2]

# Fit a Lasso regression for a grid of penalty coefficient lambda
fit <- glmnet(X, y, family="gaussian", alpha = 1, nlambda = 50, standardize = FALSE, intercept=
    FALSE)

# (A)
# Plot the solution paths for each beta_i
par(mar=c(4,4,2,2))
plot(fit, xvar = 'lambda', xlab = bquote(log(lambda)), lwd=2, label = T)

# Compute and track the in sample MSE
y.pred <- predict(fit, newx = X)
MSE <- array(NA, dim = dim(y.pred)[2])
for (i in 1:dim(y.pred)[2]){
  MSE[i] <- MSE.comp(y.pred[,i], y)
}
plot(log(fit$lambda), MSE, type = 'b', pch = 16, xlab = bquote(log(lambda)))
# The minimum of the in sample MSE is reached for the model with all the covariates (lambda
    approximately
# equal to 0). This is not surprising, as this model is the best in explaining the effect of the
# covariates on the TRAINING set (but it may be overfitting).

# (B)
# Choose the number of folds
k <- 10
MOOSE <- kfold.CV(X, y, k, fit$lambda)

# Plot the MOOSE along with the error bars.
plotCI(x = log(fit$lambda), y = MOOSE$mean.MSE, uiw = MOOSE$sd.MSE, col = 'orange', pch = 16,
    scol = 'gray', cex=0.8, lwd = 2, xlab = expression(paste(log(lambda))), ylab = 'MOOSE')
# Trace the line corresponding to the optimal lambda
abline(v = log(fit$lambda[which.min(MOOSE$mean.MSE)]), lwd = 1, col = 1, lty = 2)

# Just for the sake of completeness, we compare the result with R's built-in function. The two
    functions
# should give similar results.
cv <- cv.glmnet(X, y, lambda = fit$lambda, nfolds = k)
plot(cv)

# (C)
# We compute and plot the Mallow's Cp in order to pick the best model complexity.
Cp <- Cp.mallows(X, y, fit$lambda)
plot(log(fit$lambda), Cp, type = 'b', pch = 16)
```

Listing A.2: Function caller for exercise 2.

```
S.lambda <- function(y, lambda){
  # Computes the soft thresholding estimator
  # ----------------------------------------
  # Args:
  #   - y: vactor of the observations
```

```
  #   - lambda: penalization parameter (threshold)
  # Returns:
  #   - theta: the soft thresholding estimator
  # ----------------------------------------
  theta <- sign(y) * pmax(rep(0, length(y)), abs(y) - lambda)
  return (theta)
}
```

Listing A.3: Function implementing soft thresholding.

```
MSE.comp <- function(y.hat, y){
  # Computes the MSE of predicting y with y.hat
  # ----------------------------------------
  # Args:
  #   - y.hat: predicted values
  #   - y: true values
  # Returns:
  #   - MSE: the MSE
  # ----------------
  return (mean((y - y.hat)^2))
}

kfold.CV <- function(X, y, k, lambda){
  # Computes the k-folds Cross Validation
  # -------------------------------------
  # Args:
  #   - X: matrix of the features (n*p)
  #   - y: response vector (length n)
  #   - k: number of folds
  #   - lambda: vector of penalization parameters to try
  # Returns:
  #   - mean.MSE: mean of the estimated test errors for each value of lambda
  #   - sd.MSE: std. dev. of the estimated test errors for each value of lambda
  # ------------------------------------------------------------------------------
  n <- dim(X)[1]
  idx <- sample(rep(1:k, length.out = n))

  MSE <- array(NA, dim = c(k, length(lambda)))
  for (i in 1:k){
    test <- idx == i
    X.tr <- X[!test, ]
    y.tr <- y[!test]
    fit <- glmnet(X.tr, y.tr, family="gaussian", alpha = 1, lambda = lambda, intercept=FALSE)
    X.test <- X[test, ]
    y.test <- y[test]
    if (sum(test) == 1){
      y.pred <- predict(fit, newx = t(data.matrix(X.test)))
    }
    else{
      y.pred <- predict(fit, newx = X.test)
    }
    MSE[i,] <- apply(y.pred, 2, MSE.comp, y.test)
  }
  return (list("mean.MSE" = colMeans(MSE), "sd.MSE" = apply(MSE, 2, sd)/sqrt(k)))
}
```

Listing A.4: Functions implementing the k-folds Cross Validation.

```r
Cp.mallows <- function(X, y, lambda){
  # Computes the Cp Mallows
  # -----------------------
  # Args:
  #   - X: matrix of the features (n*p)
  #   - y: response vector (length n)
  #   - lambda: vector of penalization parameters to try
  # Returns:
  #   - Cp: Mallow's Cp for each value of lambda
  # -------------------------------------------
  n <- length(y)
  p <- dim(X)[2]
  fit <- glmnet(X, y, family="gaussian", alpha = 1, lambda = 0, standardize = FALSE, intercept =
      FALSE)
  sigma2 <- sum((y - predict(fit, newx = X))^2)/(n - p)

  fit <- glmnet(X, y, family="gaussian", alpha = 1, lambda = lambda, standardize = FALSE,
      intercept=FALSE)
  y.pred <- predict(fit, newx = X)
  MSE <- array(NA, dim = length(lambda))
  for (i in 1:length(lambda)){
    MSE[i] <- MSE.comp(y.pred[,i], y)
  }
  Cp <- MSE + 2*(fit$df/n)*sigma2
  return (Cp)
}
```

Listing A.5: Function computing the Mallow's C_p.