SML: Exercise 2

Thao Le, Finn-Ole Höner, Jason Wang, Ramon Snellen

2022-11-07

Introduction

There are many procedures for finding a best set of predictor variables and their corresponding coefficients. Of these procedures, some require numerical optimization. The elastic net is such a procedure. To numerically optimize elastic net estimation, different algorithms can be used. This report aims to answer the research question: how does the elastic net estimation of the 'cyclical coordinate descent' algorithm and 'MM' algorithm differ? A cyclical coordinate descent algorithm for the elastic net has already been developed in the `glmnet` package in `R` ("Glmnet: Fit a GLM with Lasso or Elasticnet Regularization" 2021). Therefore, we code the elastic net estimation using an MM algorithm, and compare the estimation output to that of the `glmnet` package.

Data

The data set contains seven years of store-level data collected at Dominick's Finer Foods by the University of Chicago Booth School of Business. The data can be found at https://www.chicagobooth.edu/research/kilts/datasets/dominicks. The data set contains 50 variables, which stem from:

1. customer count files, which contain information about in-store traffic;
2. a demographics file, which contains store-specific demographic data;
3. number identification files, which contain product information.

Of the fifty variables, `GROCERY_sum` is used as dependent variable. Furthermore, four categorical variables are dropped; `STORE`, `CITY`, `ZIP` and `SPHINDX`. The remaining variables are potential predictor variables.

Method

The elastic net is a form of penalized regression. In this section, we discuss the elastic net and how we use an MM algorithm to estimate it. In addition, we introduce diagnostics to (1) determine the optimal set of hyperparameters for the elastic net; and (2) compare the performance of the elastic net estimated through an MM algorithm to that of the `glmnet` package.

Let $P(\beta)$ denote a general penalty function. Then, the loss function of the regression equation becomes

$$L(\beta) = (\mathbf{y} - \mathbf{x}\beta)^T(\mathbf{y} - \mathbf{x}\beta) + \lambda P(\beta), \tag{1}$$

where λ is the hyperparameter that determines the strength of the penalty. When $P(\beta) = \beta^2$, the regression is called 'ridge' regression. Similarly, when $P(\beta) = |\beta|$, the regression is called 'LASSO' regression. Finally, any convex combination $P(\beta) = \alpha|\beta| + (1-\alpha)\beta^2$ of the 'ridge' and 'LASSO' penalty, where α denotes the weight on the 'LASSO' penalty, is called 'elastic net' (compare Zou and Hastie 2005). We use the elastic net to find the optimal set of predictor variables, since it exploits the LASSO property of variable selection, and the ridge ...

We use an MM-algorithm to estimate the elasting net. The MM-algorithm uses a majorizing function to find the coefficient vector β that minimizes the loss function specified in Equation 1. This is because the minimum is not obtained analytically. Let ϵ denote the desired level of precision. The majorizing function is specified as

$$L_{MM}(\beta) = \frac{1}{2}\beta^{\mathrm{T}}(\frac{1}{n}\mathbf{X}^{\mathrm{T}}\mathbf{X} + \lambda(1-\alpha)\mathbf{I} + \lambda\alpha\mathbf{D})\beta - \frac{1}{n}\beta^{\mathrm{T}}\mathbf{X}^{\mathrm{T}}\mathbf{y} + c,$$

where $\mathbf{D}$ is a $p \times p$ diagonal matrix with elements

$$d_{jj} = 1/\max\left(|\beta_j^0|, \epsilon\right).$$

Furthermore,

$$c = \frac{1}{2n}\mathbf{y}^{\mathrm{T}}\mathbf{y} + \frac{1}{2}\lambda\alpha\sum_{i=1}^{p}|\beta_j^0|.$$

To find the optimal penalty strength λ and the weight of the elastic net α, K-fold cross-validation is used. K-fold cross-validation starts with splitting the data set into K folds. Subsequently, K iterations are ran as follows: for each iteration $k \in \{1, \ldots, K\}$, fold k is the test set, whereas the remaining $(K-1)$ folds together form the training set. In this report, we use 10 fold cross validations. On the training set, the elastic net is estimated for all 50 values of $\lambda_i \in (10^{-2+i})_{i=0}^{12}$. Moreover, for each λ_i, all 50 values of α equally spaced between 0 and 1 are estimated, such that there are 2500 combinations of the hyperparameters α and λ for each fold. All estimated elastic nets (i.e., all combinations of the hyperparameters), are used to predict the dependent variable in the kth fold. When this procedure has been completed for all K folds, we have obtained fitted values for every observations in the data set, for all elastic nets. Consequently, prediction errors can be computed.

To evaluate which elastic net performs best (that is, the elastic net that maintains the optimal combination of hyperparameters), the Root Mean Square Error (RMSE) is computed for each elastic net; the elastic net that has the lowest RMSE is deemed to be the best model. Let $\hat{\mathbf{y}}_{\text{test}(k)} = \mathbf{X}_{\text{test}(k)}\hat{\beta}_{\text{train(k)}}$ denote the fitted values, and n_k the number of observations in fold k. The RMSE is computed as

$$\text{RMSE} = \sqrt{\frac{1}{K}\sum_{k=1}^{K}\text{MSE}_k},$$

where

$$\text{MSE} = \frac{1}{n_k}(\mathbf{y}_{\text{test}(k)} - \hat{\mathbf{y}}_{\text{test}(k)})^{\mathrm{T}}(\mathbf{y}_{\text{test}(k)} - \hat{\mathbf{y}}_{\text{test}(k)}).$$

To validate the results of our MM model, we compare our coefficient estimates with the ones of the established `glmnet` algorithm. We do not expect the coefficients to match exactly, e.g. due to different optimization algorithms. Hence, we need to measure their differences. We do so with the Absolute Error (AE)

$$\text{AE}_i = \left|\hat{\beta}_i^{GLM} - \hat{\beta}_i^{MM}\right|,$$

and the Absolute Percentage Error (APE)

$$\text{APE}_i = \left|\frac{\hat{\beta}_i^{GLM} - \hat{\beta}_i^{MM}}{\hat{\beta}_i^{GLM}}\right|.$$

where $\hat{\beta}_i^{GLM}$ and $\hat{\beta}_i^{MM}$ are the i-th coefficients of the `glmnet` and MM model, respectively.

Indeed, when applying the `glmnet` and MM implementations to the Dominick's Finer Food data set, we get different coeffciients. Figure 1 shows the mean-absolute-percentage differences (MAPE) of these two implementations for the different predictors. For most predictors, the APE is around to 1%.

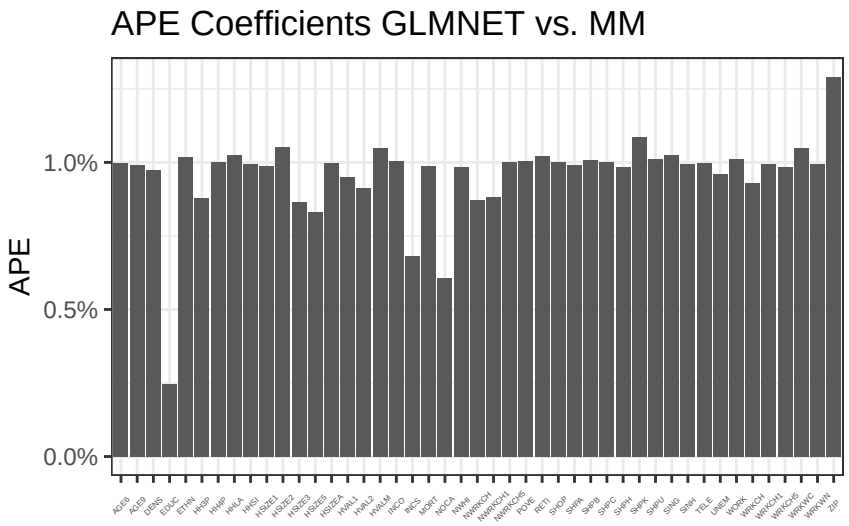

Figure 1: Absolute percentage difference between glmnet and MM coefficients.

Results

One explanation for this difference could lie in the algorithms' speed of convergence. For a given data set the MM algorithm might not converge fast enough to provide the same estimates as the `glmnet` implementation. In fact, the cyclical coordinate decent used in `glmnet` is a very efficient algorithm according to Friedman, Hastie, and Tibshirani (2010).

We investigate this hypothesis by looking at the estimates' differences for lower convergence thresholds ϵ of the MM algorithm. Figure 2 shows the zero correlation between the Mean Absolute Error and the parameter ϵ: This precision parameter appears to play no role in the discrepancy between the coefficients of `glmnet` and the MM algorithm.

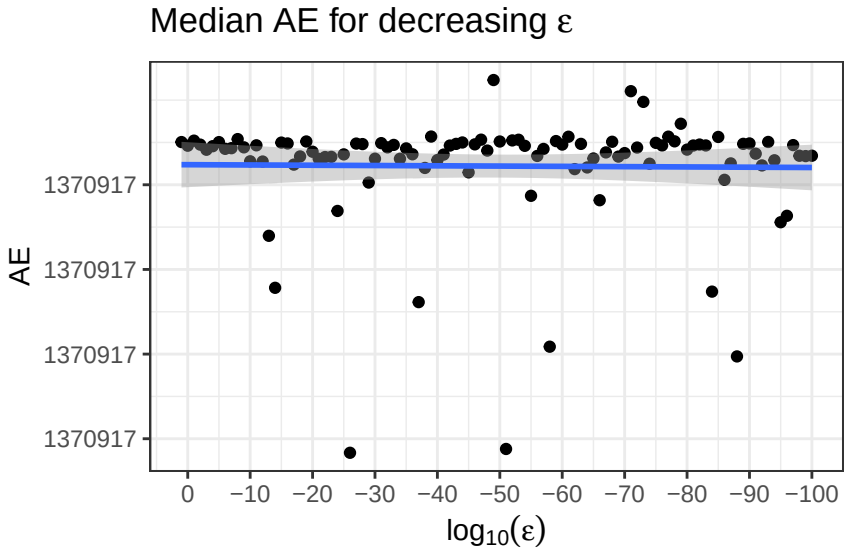

Figure 2: Development of AE fo decreasing precision threshold ϵ.

The uniform difference between the coefficients, and the result that this difference does not depend on the precision parameter ϵ. This leads us to believe that there are some structural differences between the `glmnet` and MM algorithm. However, as these discrepancies are low and similar across predictors, both algorithms seem to implement the same model.

Test on real data using k-fold

First we find the optimal Lambda and Alpha value for our MM-algorithm

```
## Alpha is:  0.5102041 . The minimum lambda is:  0.9102982
##  The minimum RMSE is:  21194047
```

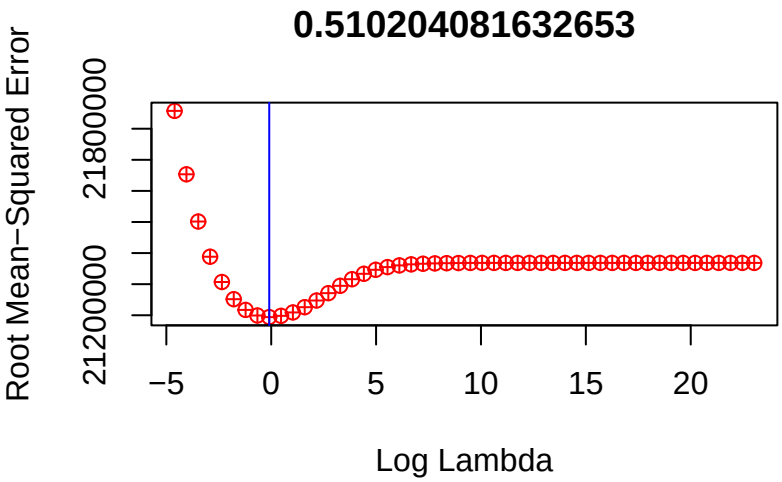

Figure 3: Changes in RMSE for each λ using the best cross validated α MM

```
## integer(0)
```

Then we compare to the optimal lambda and alpha value using the `glmnet` package.

Conclusion and Discussion

The estimations based on the `glmnet` and MM algorithm differ with respect to the size of the coefficients. In addition, the prediction $RMSE$ of the MM algorithm is higher than the one of the `glmnet` implementation.

References

Code

Toolbox

```
" Compute the residuals
#"
#' @param vY vector, y the outcome
#' @param mX matrix, X the predictors
#' @param vBeta vector, the parameters beta
```

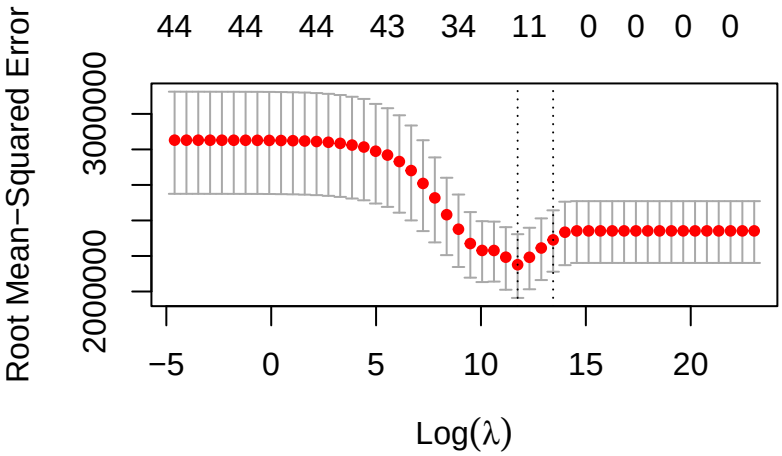

Figure 4: Changes in RMSE for each λ using the best cross validated α GLMET

```
#' @return vResiduals, vector of residuals
#' @examples
#' residuals(vY, mX, vBeta)
residuals <- function(vY, mX, vBeta) {
    vResiduals <- vY - mX %*% vBeta
    return(vResiduals)
}
#' Loss function
#'
#' @param vResiduals vector, the residuals
#' @param vBeta vector, the parameters beta
#' @param dLambda double, the lambda parameter
#' @param dAlpha double, the alpha parameter
#' @return dLoss, double the value of the loss function
#' @examples
#' loss(vResiduals, vBeta, dLambda, dAlpha)
loss <- function(vResiduals, vBeta, dLambda, dAlpha) {
    vBeta <- as.matrix(vBeta)

    # init
    iN <- nrow(vResiduals)
    dConstr <- dLambda * ((1 - dAlpha)/(2 * t(vBeta) %*%
        vBeta) + dAlpha * norm(vBeta, type = "1"))

    # compute the loss function
    dLoss <- (2 * iN)^(-1) * (t(vResiduals) %*% vResiduals) +
        dConstr

    return(dLoss)
}

#' Change of loss function
```

```r
#'
#' @param vBeta0 vector, last iterations beta
#' @param vBeta1 vector, this iterations beta
#' @param vResiduals0 vector, residual vector
#' @param vResiduals1, vector, residual vector
#' @param dLambda double, the lambda parameter
#' @param dAlpha double, the alpha parameter
#' @return dLoss, double the value of the loss function
#' @examples
#' loss_change(vBeta0, vBeta1, vResiduals, dLambda, dAlpha)
loss_change <- function(vBeta_0, vBeta_1, vResiduals0,
    vResiduals1, dLambda, dAlpha) {
    # compute the loss
    dLoss0 <- loss(vResiduals0, vBeta_0, dLambda, dAlpha)
    dLoss1 <- loss(vResiduals1, vBeta_1, dLambda, dAlpha)

    # compute the change
    dChange <- (dLoss0 - dLoss1)/dLoss0

    return(dChange)
}
#' Get the D_jj element, vectorized
#'
#' @param vBeta vector, vector of betas
#' @param dEps double, precision variable
#' @return vD, vector, diagonal elements of D
#' @examples
#' computeDjj(vBeta, dEps)
computeDjj <- Vectorize(function(vBeta, dEps) {
    dD <- 1/(max(c(abs(vBeta), dEps)))
    return(dD)
}, vectorize.args = "vBeta")
#' Create the D matrix
#'
#' @param vBeta vector, vector of betas
#' @param dEps double, precision variables
#' @return mD, matrix, matrix D with d_jj on diagonal
#' @examples
#' getD(vBeta, dEps)
getD <- function(vBeta, dEps) {
    vDiag <- computeDjj(vBeta, dEps)
    mD <- diag(vDiag)
    return(mD)
}
#' Create the A matrix
#'
#' @param mA1 matrix, pulled out value of A matrix
#' @param mD matrix, the matrix D
#' @param dAlpha double, alpha parameter
#' @param dLambda double, lambda parameter
#' @return mD, matrix, matrix D with d_jj on diagonal
#' @examples
#' getA(mX, mD, dAlpha, dLambda)
```

```r
getA <- function(mA1, mD, dAlpha, dLambda) {
    mA <- mA1 + dLambda * dAlpha * mD
    return(mA)

}
#' Calculate the update for Beta
#'
#' @param mX matrix, the predictor's data
#' @param vY vector, the vector of the outcome variable
#' @param mA matrix, A matrix
#' @return vBetaUpdate, vector, the updated betas
#' @examples
#' updateBeta(mX, vY, mA)
updateBeta <- function(mX, vY, mA) {
    # init
    iN <- nrow(mX)

    # calculate
    vBetaUpdate <- solve(mA, (1/iN) * (t(mX) %*% vY))

    return(vBetaUpdate)

}
#' Perform one iteration of the MM
#'
#' @param mX matrix, the predictor's data
#' @param vY vector, the vector of the outcome variable
#' @param vBeta vector, the betas
#' @param dEps double, the precision epsilon
#' @param dAlpha double, the alpha parameter
#' @param dLambda double, the lambda parameter
#' @return vBetaUpdate, vector, the updated betas
#' @examples
#' itElasticNetMM(mX, vY, vBeta, dEps, dAlpha, dLambda)
itElasticNetMM <- function(mX, vY, vBeta, dEps, dAlpha,
    dLambda) {
    # perform one iteration
    mD <- getD(vBeta, dEps)
    mA <- getA(mX, mD, dAlpha, dLambda)
    vBetaUpdate <- updateBeta(mX, vY, mA)

    return(vBetaUpdate)
}
#' Run the full MM
#'
#' @param mX matrix, the predictor's data
#' @param vY vector, the vector of the outcome variable
#' @param vBeta vector, the betas
#' @param dEps double, the precision epsilon
#' @param dAlpha double, the alpha parameter
#' @param dLambda double, the lambda parameter
ElasticNetMM <- function(mX, vY, dEps, dAlpha, dLambda) {
    # loop objects
```

```
    iP <- ncol(mX)
    iN <- nrow(mX)
    ik <- 1
    dLossChange <- 0
    vBeta0 <- runif(ncol(mX))
    dEps <- 1e-05
    mXtX <- t(mX) %*% mX

    # Pull out part one of A
    dA1 <- (1/iN) * mXtX + (dLambda * (1 - dAlpha)) *
        diag(iP)

    while ((ik == 1) | (dLossChange > dEps)) {
        # update counter
        ik <- ik + 1

        # perform one update
        mD <- getD(vBeta0, 1e-07)
        mA <- getA(dA1, mD, dAlpha, dLambda)
        vBeta1 <- updateBeta(mX, vY, mA)

        # get residuals
        vResiduals0 <- residuals(vY, mX, vBeta0)
        vResiduals1 <- residuals(vY, mX, vBeta1)

        # compute loss change
        dLossChange <- loss_change(vBeta0, vBeta1,
            vResiduals0, vResiduals1, dLambda, dAlpha)

        # output print(paste0('Iteration: ', ik,
        # ' \n')) print(paste0('Loss Change: ',
        # dLossChange, ' \n')) update the beta
        vBeta0 <- vBeta1
    }

    # print('Beta Estimate') print(vBeta0)
    return(vBeta0)
}
#' MAPE
#'
#' @param vBeta_A vector, estimates of model A
#' @param vBeta_B vector, estimates of model B
#' @return MAPE, vector - main absolute percentage error
MAPE <- function(vBeta_A, vBeta_B) {
    return(abs((vBeta_A - vBeta_B)/vBeta_A)/100)
}
#' MAE
#'
#' @param vBeta_A vector, estimates of model A
#' @param vBeta_B vector, estimates of model B
#' @return MAE, vector - main absolute percentage error
MAE <- function(vBeta_A, vBeta_B) {
    return(abs(vBeta_A - vBeta_B))
```

```r
}
#' Create a dataframe matching up estimates
#'
#' @param vBeta_A vector, estimates of model A
#' @param vBeta_B vector, estimates of model B
#' @param vColNames, vector of length 3, col names to use
#' @return dfCompareBetaTable, df
CompareEstimates <- function(vBeta_A, vBeta_B, vColNames = c("GLMNET",
    "MM", "Predictor")) {
    # compare estimates with each other
    dfCompareBeta <- cbind(vBeta_A, vBeta_B)
    vNameCoef <- rownames(dfCompareBeta)

    dfCompareBetaTable <- dfCompareBeta %>%
        as_tibble()
    dfCompareBetaTable$Predictor <- vNameCoef
    colnames(dfCompareBetaTable) <- vColNames

    return(dfCompareBetaTable)
}


#' Function to CV k_fold to look for optimal lambda
#'
#' @param mX matrix, the predictor's data
#' @param vY vector, the vector of the outcome variable
#' @param nfolds integer, number of folds
#' @param vBeta vector, the betas
#' @param dEps double, the precision epsilon
#' @param dAlpha double, the alpha parameter
#' @param lLambda list, list of lambda parameter
#' @return mRSS, matrix of Residual sum of squares, each column is for folds, each row is for each lamb
k_fold_lambda <- function(mX, vy, nfolds, vBeta, dEps,
    dAlpha, lLambda) {
    # Shuffle the data
    set.seed(321)
    data <- cbind(vy, mX)
    data <- data[sample(nrow(data)), ]

    # Create n equal folds, since the indexes are
    # discrete, sometimes fold length differs by
    # 1 observation
    folds <- cut(seq(1, nrow(data)), breaks = nfolds,
        labels = FALSE)

    # Perform cross validation
    mRSS = matrix(NA, nrow = 50, ncol = nfolds)  #Create a matrix, each column is the RSS for each test
    for (i in 1:ncol(mRSS)) {
        # Segment the data by the number of folds
        testIndexes <- which(folds == i, arr.ind = TRUE)
        testSet <- data[testIndexes, ]
        trainSet <- data[-testIndexes, ]
```

```
        # Use the test and train data partitions
        # on Elastic net model
        y_train <- trainSet[, 1]
        y_test <- testSet[, 1]
        X_train <- scale(trainSet[, -1])
        X_test <- scale(testSet[, -1])

        lRSS = rep(NA, nrow(mRSS))
        for (j in 1:nrow(mRSS)) {
            # Calculate RSS for each lambda value
            vBeta_new = ElasticNetMM(X_train, y_train,
                dEps, dAlpha, lLambda[j])
            lRSS[j] = sum(residuals(y_test, X_test,
                vBeta_new)^2)  # is this fine?
        }
        mRSS[, i] = lRSS  #Insert RSS values into matrix
    }
    return(mRSS)
}

#' Function to get the root mean square errors for each lambda value
#' @param mRSS matrix, residual sum of squares over k-folds for each lambda value
#' @return means list, root mean squared errors of each lambda
root_mean <- function(mRSS) {
    means = sqrt(rowMeans(mRSS))
    return(means)
}

#' Function for CV that looks for optimal lambda and alpha, produce 1 plot for each alpha value
#'
#' @param mX matrix, the predictor's data
#' @param vY vector, the vector of the outcome variable
#' @param nfolds integer, number of folds
#' @param vBeta vector, the betas
#' @param dEps double, the precision epsilon
#' @param lAlpha list, the alpha parameter
#' @param lLambda list, the lambda parameter
#' @return Alpha_min, alpha that returns the lowest RMSE for all lambda-alpha combinations
k_fold_plots <- function(mX, vy, nfolds, vBeta, dEps,
    lAlpha, lLambda) {
    lRMSE_min = list()
    for (a in 1:length(lAlpha)) {
        mRSS = k_fold_lambda(mX, vy, nfolds, vBeta,
            dEps, lAlpha[a], lLambda)

        # get root mean square errors for each
        # lambda value
        means = root_mean(mRSS)

        # Get index of the min lambda
        ind = which.min(means)
        lambda_min = lLambda[ind]
        cat("Alpha is: ", lAlpha[a], ". The minimum lambda is: ",
```

```
            lambda_min, "\n", "The minimum RMSE is: ",
            min(means), "\n")
        lRMSE_min[a] = min(means)

        # Plot
        df <- data.frame(RMSE = means, `Log(Lambda)` = log(lLambda))
        plot(df$Log.Lambda., df$RMSE, pch = 10, main = paste(lAlpha[a]),
            ylab = "Root Mean-Squared Error", xlab = "Log Lambda",
            col = "red") + abline(v = log(lambda_min),
            col = "blue")
    }
    ind = which.min(lRMSE_min)
    Alpha_min = lAlpha[ind]
    # cat('The alpha with minimum RMSE is: Alpha
    # = ',Alpha_min)
    return(Alpha_min)
}

#' Function to plot the alpha with lowest RMSE
#'
#' @param mX matrix, the predictor's data
#' @param vY vector, the vector of the outcome variable
#' @param nfolds integer, number of folds
#' @param vBeta vector, the betas
#' @param dEps double, the precision epsilon
#' @param dAlpha double, alpha that returns the lowest RMSE, the output of k_fold_plots function
#' @param lLambda list, the lambda parameter
#' @return None
#'
k_fold_alpha_plots <- function(mX, vy, nfolds, vBeta,
    dEps, dAlpha, lLambda) {
    lRMSE_min = list()

    mRSS = k_fold_lambda(mX, vy, nfolds, vBeta, dEps,
        dAlpha, lLambda)

    # get root mean square errors for each lambda
    # value
    means = root_mean(mRSS)

    # Get index of the min lambda
    ind = which.min(means)

    lambda_min = lLambda[ind]
    cat("Alpha is: ", dAlpha, ". The minimum lambda is: ",
        lambda_min, "\n", "The minimum RMSE is: ",
        min(means), "\n")

    lRMSE_min = min(means)

    # Plot
    df <- data.frame(RMSE = means, `Log(Lambda)` = log(lLambda))
    plot(df$Log.Lambda., df$RMSE, pch = 10, main = paste(dAlpha),
```

```
        ylab = "Root Mean-Squared Error", xlab = "Log Lambda",
        col = "red", font.sub = 4) + abline(v = log(lambda_min),
        col = "blue")
}


#' Function to test and plot CV results using package GLMNET
#' @param mX matrix, the predictor's data
#' @param vY vector, the vector of the outcome variable
#' @param nfolds integer, number of folds
#' @param dAlpha double, the alpha parameter
#' @param lLambda list, the lambda parameter
#' @return lRMSE_min, list of minimum root mean square error for each alpha
plot_cv_GLMET <- function(X, y, alpha) {
    set.seed(321)
    library(glmnet, quietly = TRUE)
    result.cv <- cv.glmnet(X, y, alpha = alpha, lambda = 10^seq(-2,
        10, length.out = 50), nfolds = 10, standardize = TRUE)

    ## To plot Root Mean Squared Error (RMSE) to
    ## be on the same scale as y:
    result.cv$cvm <- result.cv$cvm^0.5
    result.cv$cvup <- result.cv$cvup^0.5
    result.cv$cvlo <- result.cv$cvlo^0.5
    p.plot = plot(result.cv, ylab = "Root Mean-Squared Error",
        font.sub = 4)
}
```

Main

```
# Setup
# ------------------------------------------------------------------------
# load packages
if (!require(pacman)) {
    install.packages("pacman")
}
p_load(tidyverse, simglm, rlist, latex2exp, glmnet)

# set seed
set.seed(321)

# import custom functions
source("./dev/toolbox.r")

# load data
load("supermarket1996.RData")

# writing format objects
mytheme <- theme_bw() + theme(legend.position = "bottom")

# Synth.  Data
# ------------------------------------------------------------------
sim_arguments <- list(formula = y ~ 1 + x1 + x2 + x3 +
```

```
    x4, fixed = list(x1 = list(var_type = "continuous",
    mean = 180, sd = 50), x2 = list(var_type = "continuous",
    mean = 75, sd = 20), x3 = list(var_type = "continuous",
    mean = -23, sd = 4), x4 = list(var_type = "continuous",
    mean = 1, sd = 20)), sample_size = 1e+06, reg_weights = c(2,
    5, -0.7, 100, -23))

# data set
dfData <- simulate_fixed(data = NULL, sim_arguments) %>%
    generate_response(sim_arguments) %>%
    select(-c(level1_id, random_effects, error, fixed_outcome,
        X.Intercept.))

# data objects
mX <- dfData %>%
    select(-y) %>%
    data.matrix() %>%
    scale()
vY <- dfData %>%
    select(y) %>%
    data.matrix()

# Real Data
# ------------------------------------------------------------------
vY <- supermarket1996 %>%
    select(GROCERY_sum) %>%
    as.matrix()
mX <- supermarket1996 %>%
    select(-c("STORE", "CITY", "GROCCOUP_sum", "SHPINDX",
        "GROCERY_sum")) %>%
    as.matrix() %>%
    scale()

# Model
# ------------------------------------------------------------------

# parameters
dLambda <- 10
dAlpha <- 0.5
dEps <- 1e-09

vBeta_MM <- ElasticNetMM(mX, vY, dEps, dAlpha, dLambda)

print("Beta Estimate")
print(vBeta_MM)

# Comparison with glmnet
# ------------------------------------------------
# uses an intercept per default
model_glm <- glmnet(x = mX, y = vY, alpha = dAlpha,
    lambda = dLambda, intercept = FALSE, standardize = FALSE)
vBeta_glm <- model_glm %>%
    coef() %>%
```

```
    as.matrix()
vBeta_glm <- vBeta_glm[-1, ]
vNameCoef <- names(vBeta_glm)

dfCompareBetaTable <- CompareEstimates(vBeta_glm, vBeta_MM)

plot_coef_rmse <- dfCompareBetaTable %>%
    mutate(MAPE = MAPE(GLMNET, MM)) %>%
    ggplot(aes(x = Predictor, y = MAPE)) + geom_bar(stat = "identity") +
    labs(x = "", y = "APE") + scale_x_discrete(breaks = vNameCoef,
    labels = abbreviate) + scale_y_continuous(labels = scales::percent) +
    mytheme + theme(axis.text.x = element_text(angle = 45,
    size = 3, vjust = 0.5))


# Development for Epsilon
# ------------------------------------------------
iEpsStart <- 1
iEpsEnd <- -100
iEpsStep <- -1

# epsilon steps
vEps <- 10^seq(iEpsStart, iEpsEnd, iEpsStep)
lBeta_MM <- list()
lCompare <- list()

# estiamte model for each epsilon and compare to
# glmnet
for (i in seq_along(vEps)) {
    vBeta_MM <- ElasticNetMM(mX, vY, vEps[i], dAlpha,
        dLambda)
    lCompare[[i]] <- CompareEstimates(vBeta_glm, vBeta_MM)
    lCompare[[i]]$Epsilon <- vEps[i]
}

model_glm <- glmnet(x = mX, y = vY, alpha = dAlpha,
    lambda = dLambda, intercept = FALSE, standardize = FALSE)
vBeta_glm <- model_glm %>%
    coef() %>%
    as.matrix()
vBeta_glm <- vBeta_glm[-1, ]
vNameCoef <- names(vBeta_glm)

# transform to dataframe
dfBetaCompareEps <- list.stack(lCompare)
dfBetaCompareEps <- dfBetaCompareEps %>%
    mutate(MAPE = MAPE(GLMNET, MM), MAE = MAE(GLMNET,
        MM)) %>%
    as_tibble()

plot_MAPE_eps <- dfBetaCompareEps %>%
    group_by(Epsilon) %>%
    summarise(MAPE = mean(MAPE), MAE = median(MAE)) %>%
```

```
    ggplot(aes(x = log10(Epsilon), y = MAE)) + geom_point() +
    geom_smooth(method = "lm") + scale_x_continuous(trans = "reverse",
    breaks = seq(iEpsStart - 1, iEpsEnd, -10)) + labs(x = TeX("$log_{10} (\\epsilon)$"),
    y = "AE") + mytheme
plot_MAPE_eps

plot_MAPE_eps_pred <- dfBetaCompareEps %>%
    ggplot(aes(x = log10(Epsilon), y = MAPE, group = Predictor)) +
    facet_wrap(~Predictor, scales = "free_y", ncol = 9) +
    geom_smooth() + scale_x_continuous(trans = "reverse",
    breaks = seq(iEpsStart, iEpsEnd, iEpsStep)) + labs(x = TeX("$log_{10} (\\epsilon)$")) +
    theme(axis.text.y = element_blank()) + mytheme
plot_MAPE_eps_pred

#----------------------------------------
load("supermarket1996.Rdata")
df <- data.frame(supermarket1996)
sub_df <- subset(df, select = -c(STORE, CITY, ZIP,
    GROCCOUP_sum, SHPINDX))
vy <- as.vector(sub_df$GROCERY_sum)  # y variable
mX <- as.matrix(sub_df[, -1])
dEps = 10^(-10)
vBeta = rep(1, ncol(mX))
lAlpha = seq(0, 1, length.out = 50)
lLambda = 10^seq(-2, 10, length.out = 50)
nfolds = 10

dAlpha = k_fold_plots(mX, vy, nfolds, vBeta, dEps,
    lAlpha, lLambda)

# Plot lambda behaviour with teh optimal alpha
# value
plot_optimal_alpha = k_fold_alpha_plots(mX, vy, nfolds,
    vBeta, dEps, dAlpha, lLambda)

# plot optimal lambda, alpha using the package
plot_cv_package = plot_cv_GLMET(mX, vy, alpha = dAlpha)
```


Friedman, Jerome, Trevor Hastie, and Rob Tibshirani. 2010. "Regularization Paths for Generalized Linear Models via Coordinate Descent." *Journal of Statistical Software* 33 (1): 1.

"Glmnet: Fit a GLM with Lasso or Elasticnet Regularization." 2021. *RDocumentation*. https://www.rdocumentation.org/packages/glmnet/versions/4.1-4/topics/glmnet.

Zou, Hui, and Trevor Hastie. 2005. "Regularization and Variable Selection via the Elastic Net." *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 67 (2): 301–20.