Computational & Mathematical Statistics - Fall 2023
Assignment 3

Adaptive regression; Stepwise median regression based on AIC; Monte Carlo experiments

Ioannis Maris, math1p0004

University of Crete,
Department of Mathematics & Applied Mathematics,
Department of Computer Science,
FORTH - Foundation for Research & Technology - Institute of Applied and Computational Mathematics, Institute of Computer Science.

December 2023

Assignment 3 Part 1

Adaptive LAD LASSO

The optimization problem solved by the Adaptive LAD[1] Lasso regression:

$$\text{Minimize} \quad \sum_{i=1}^{n} |y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij}| + \lambda \sum_{j=1}^{p} \mathbf{w_j} |\beta_j|,$$

where

- λ is the regularization parameter $\big($usually tuned via CV or prior chosen, for example, take $\lambda = \log(n)/|\hat{\beta}^{init}|\big)$.
- w_j are the weights for the adaptive lasso, typically set to $1/|\hat{\beta}_j^{init}|$, where $\hat{\beta}_j^{init}$ are the coefficients from an initial fit.

[1] Least absolute deviations

Assignment 3 Part 1

- Estimate model M5 from assignment 2 using adaptive LAD LASSO. For that purpose compare estimates computed from packages rqpen and hqreg, with their penalty terms tuned via cross-validation. Finally, compare against the adaptive LASSO solution derived via glmnet.

```
data <- airquality
library(caret)
set.seed(4)
# Remove rows with NA values
data_ <- na.omit(data)

# Shuffle the cleaned data
data_ <- data_[sample(nrow(data_)), ]
```

Assignment 3 Part 1

- Estimate model M5 from assignment 2 using adaptive LAD LASSO. For that purpose compare estimates computed from packages rqpen and hqreg, with their penalty terms tuned via cross-validation. Finally, compare against the adaptive LASSO solution derived via glmnet.

```
data <- airquality
library(caret)
set.seed(4)
# Remove rows with NA values
data_ <- na.omit(data)

# Shuffle the cleaned data
data_ <- data_[sample(nrow(data_)), ]
```

- Evaluate the three alternative implementations using MAE derived from 10-fold CV.

Assignment 3 Part 1 - set up the dataset; add 3 correlated predictors

```
set.seed(3)
# Scaling
scale_var <- function(x) {
  return((x - mean(x)) / sd(x))
}
# Function to create noisy predictor with desired correlation
create_noisy_predictor <- function(variable, rho) {
  noise <- rnorm(length(variable))
  return(rho * variable + sqrt(1 - rho^2) * noise)
}
# Creating Z1, Z2, Z3 with desired correlation
data_Z1 <- create_noisy_predictor(scale_var(data_Z1 <- create_noisy_
    predictor(scale_var(data_Solar.R), 0.8)
data_Z2 <- create_noisy_predictor(scale_var(data_Z2 <- create_noisy_
    predictor(scale_var(data_Wind), 0.8)
data_Z3 <- create_noisy_predictor(scale_var(data_Z3 <- create_noisy_
    predictor(scale_var(data_Temp), 0.8)
cor(data_Z1, data_Z1, data_Solar.R) # Should be approximately 0.8
cor(data_Z2, data_Z2, data_Wind)    # Should be approximately 0.8
cor(data_Z3, data_Z3, data_Temp)    # Should be approximately 0.8
data_log_y <- log(data_log_y <- log(data_Ozone)
log_y <- log(data_$Ozone)
```

Assignment 3 Part 1 - define error metrics

```
# log acc. ration
LAR <- function(y, forcast) {
  n <- length(y)
  lar <- 0
  n_ <- 0 #counter
  for (i in 1:n) {
    if (y[i] != 0) {
      Q = as.numeric(forcast)[i]/y[i]
        if (Q > 0) {
          lar = lar + abs(log(Q))
          n_ = n_ + 1
        }
    }
  }
  lar <- lar/n_
  return(lar)
}
MAPE <- function(y, forcast) {
  n <- length(y)
  mape_sum <- 0
  n_ <- 0 #counter
  for (i in 1:n) {
    if (y[i] != 0) {
      mape_sum = mape_sum + abs((y[i]-as.numeric(forcast)[i])/y[i])
      n_ = n_ + 1
    }
  }
  mape <- (mape_sum/n_) * 100
  return(mape)
}
```

Assignment 3 Part 1 - define error metrics

```r
MAE <-  function (y, forecast) {
  finite_indices <- is.finite(y) & is.finite(forecast)
  if (any(finite_indices)) {
    mean(abs(y[finite_indices] - forecast[finite_indices]), na.rm = TRUE)
  } else {
    NA  # Failure
  }
}

RMSE <-  function (y, forecast) {
  finite_indices <- is.finite(y) & is.finite(forecast)
  if (any(finite_indices)) {
    sqrt(mean((y[finite_indices] - forecast[finite_indices])^2, na.rm = TRUE))
  } else {
    NA  # Or some other default value indicating failure
  }
}

sMdAPE <-  function (y, forecast) {
  finite_indices <- is.finite(y) & is.finite(forecast) & (y + forecast != 0)
  if (any(finite_indices)) {
    median(200*abs(y[finite_indices]-forecast[finite_indices])/(y[finite_indices] +
                                                  forecast[finite_indices]),
                                                  na.rm = TRUE)

  } else {
    NA  # Failure
  }
}
```

Assignment 3 Part 1 - build a generalized CV function

Penalized estimation - Stepwise regression - Adaptive regression

```
KfoldCVPerf.general <- function (K=10, data, formula, name='model name',
                                 p=0.005, folds.seed=42, inv.trans=function (y) return (y),
                                 method="forward", criterion="AIC", early.drop=0.01) {
                        ### Datatypes ###
    ## K: int  ##data: data.frame ## name: string -> model's name ##
    ## formula: as.formula() ## folds.seed: int: random folds
    ## inv.trans: inverse transformation of the response (function datatype) ##
    ## method: 'ridge' || 'lasso' / for penalized est. (LS) / ##
    ## method: 'adaptive_lasso' || 'adaptive_ridge' / for adaptive reg. / ##
    ## cirterion: 'hqreg' (LAD) || 'rqpen' (LAD) || 'glmnet' (LS) / for adaptive reg. /
    ## method: 'forward' || 'backward' / for stepwise reg. / ##
    ## criterion: 'AIC' || 'BIC' || 'p_val' / for stepwise reg. / ##
    ### NOTE: The response y (from the formula) must be in the dataframe ###

    # Shuffle the data to get random folds:
    set.seed(folds.seed)
    data <- data[sample(nrow(data)), ]
    y <- all.vars(formula)[1] # get response name (str)
    rows_per_fold <- nrow(data)/K
    # Create a binning var
    binning_variable <- cut(seq(1, nrow(data)), breaks = K, labels = FALSE)
    folds <- split(data, binning_variable) # Split the folds
    folds_list <- list() # Create a list of K folds
    for (i in 1:K) {
        folds_list[[i]] = folds[[i]]
        }
    # DF to store results
    perf_df <- data.frame(Fold = integer(0), RMSE = numeric(0), MAE = numeric(0),
                          sMdAPE = numeric(0), LAR = numeric(0), MAPE = numeric(0)
                          )
```

Assignment 3 Part 1 - build a generalized CV function

Penalized estimation - Stepwise regression - Adaptive regression

```
# Loop through the folds
    for (i in 1:K) {
        #Keep the i-th fold as it is (testing)
        val_fold <- folds_list[[i]]
        y_val <- val_fold[[y]]
        # Combine the remaining folds (training)
        train_folds <- do.call(rbind, folds_list[-i])
        y_train <- train_folds[[y]]
        # Temporarily assign train_folds to a global variable
        .GlobalEnv$train_folds_temp <- train_folds # Important for F-BAIC
        #Choose model
        if (method == "forward" && criterion == "AIC") {
            require(olsrr) # For stepwise
            # Use the global temporary variable for model fitting
            LM <- lm(formula, data = train_folds_temp)
            model.FAIC <- ols_step_forward_aic(LM)
            model <- model.FAIC$model
        } else if (method == "forward" && criterion == "BIC") {
            require(leaps) # For stepwise
            # max features for stepwise reg.
            nvmax <- length(attr(terms(formula), "term.labels"))
            model.forward <- regsubsets(formula, data=train_folds,
                                        method="forward", nvmax=nvmax)
            model.Fbic <- summary(model.forward)$bic
            FBIC_vars <- names(coef(model.forward, which.min(model.Fbic)))
            FBIC_formula <- as.formula(paste("y_train ~",
                                    paste(FBIC_vars[-1], collapse=' + ')))

            model <- lm(FBIC_formula, data=train_folds)
```

```r
} else if (method == "backward" && criterion == "AIC") {
          require(olsrr) # For stepwise
          LM <- lm(formula, data = train_folds_temp)
          model.BAIC <- ols_step_backward_aic(LM)
          model <- model.BAIC$model

} else if (method == "backward" && criterion == "BIC") {
    require(leaps) # For stepwise
    # max features for stepwise reg.
    nvmax <- length(attr(terms(formula), "term.labels"))
    model.backward <- regsubsets(formula, data=train_folds,
                            method="backward", nvmax=nvmax)
    model.Bbic <- summary(model.backward)$bic
    BBIC_vars <- names(coef(model.backward, which.min(model.Bbic)))
    BBIC_formula <- as.formula(paste("y_train ~",
                          paste(BBIC_vars[-1], collapse=' + ')))

    model <- lm(BBIC_formula, data=train_folds)
} else if (method == "forward" && criterion == "p_val") {
          require(olsrr) # For stepwise
          # Use the global temporary variable for model fitting
          LM <- lm(formula, data = train_folds_temp)
          model.Fp <- ols_step_forward_p(LM, p_val=p) #0.005
          model <- model.Fp$model

} else if (method == "backward" && criterion == "p_val") {
    require(olsrr) # For stepwise
    LM <- lm(formula, data = train_folds_temp)
    model.Bp <- ols_step_backward_p(LM, p_val=p) #0.005
    model <- model.Bp$model
```

Assignment 3 Part 1 - build a generalized CV function

Penalized estimation - Stepwise regression - Adaptive regression

```
} else if (method == "ridge") {
    require(glmnet)
      model <- cv.glmnet(
        x=model.matrix(formula, train_folds),
        y=as.vector(y_train), type.measure="mse",
        alpha=0, nfolds=10, parallel = TRUE)
} else if (method == "lasso") {
            require(glmnet)
            model <- cv.glmnet(
              x=model.matrix(formula, train_folds),
              y=as.vector(y_train), type.measure="mse",
              alpha=1, nfolds=10, parallel = TRUE)
} else if (method == "adaptive_lasso" && criterion == "hqreg") {
    require(hqreg)
    quiet <- function(x) { sink(tempfile())
                           on.exit(sink())
                           invisible(force(x))
                          } # Hides the CV messeges.
    # Fit an initial model to get coefficient estimates
    initial_fit <- quiet(
              cv.hqreg(X = model.matrix(formula, train_folds),
                       y = as.numeric(y_train),
                       method = 'quantile', alpha = 1, nfolds = 10,
                       type.measure = 'mae', tau = 0.5, seed = 42))
    initial_coeff <- coef(initial_fit, s=initial_fit$lambda.min)
    # Calculate adaptive weights
    # Avoid div by zero - substitute very small values for 0 coefs
    initial_coeff[initial_coeff == 0] <- 1e-5
    n <- dim(train_folds)[1]
```

Assignment 3 Part 1 - build a generalized CV function

Penalized estimation - Stepwise regression - Adaptive regression

```
        weights <- 1 / (abs(initial_coeff[-1]) + 1/n)
        # Fit the final Adaptive LAD Lasso model (cv for lambda)
        model <- quiet(
                cv.hqreg(X = model.matrix(formula, train_folds),
                         y = as.numeric(y_train),
                         method = 'quantile',
                         tau = 0.5, type.measure = 'mae',
                         penalty.factor = weights,
                         alpha = 1, seed = 42))
    } else if (method == "adaptive_lasso" && criterion == "glmnet") {
                require(glmnet)
                # Fit an initial model to get coefficient estimates
                initial_fit <- cv.glmnet(
                            x = model.matrix(formula, train_folds),
                            y = as.numeric(y_train),
                            type.measure = 'mse', alpha = 1,
                            seed = 42, parallel = TRUE)
                initial_coeff <- coef(initial_fit, s=initial_fit$lambda.min)
                # Calculate adaptive weights
                # Avoid div by zero - substitute very small values for 0 coefs
                initial_coeff[initial_coeff == 0] <- 1e-5
                n <- dim(train_folds)[1]
                weights <- 1 / (abs(initial_coeff[-1]) + 1/n)
                # Fit the final Adaptive Lasso model (tune lambda via CV)
                model <- cv.glmnet(x = model.matrix(formula, train_folds),
                             y = as.numeric(y_train),
                             penalty.factor = weights,
                             type.measure = 'mse', seed = 42,
                             alpha = 1, parallel = TRUE)
```

Assignment 3 Part 1 - build a generalized CV function

Penalized estimation - Stepwise regression - Adaptive regression

```
} else if (method == "adaptive_lasso" && criterion == "rqpen") {
            require(rqPen)
            require(doMC)
            registerDoMC(cores = 2) # Parallel backend
            suppressWarnings({
                system.time(
                    model <- rq.pen.cv(
                      x = model.matrix(formula, train_folds)[, -1],
                      y = as.vector(y_train),
                      tau = 0.5, penalty = "aLASSO",
                      cvSummary=median, nfolds = 10))
                })
} else {
    stop("Unsupported method or criterion.")
}
rm(train_folds_temp, envir = .GlobalEnv) # Remove globenv
# Check for nestedcv (ridge; lasso and d1ff dtypes)
if (method %in% c("ridge", "lasso") || criterion == "glmnet") {
    preds <- as.vector(predict(model,
                        newx = model.matrix(formula, val_fold),
                        s = "lambda.min"))
} else if (criterion == 'rqpen') {
    preds <- as.vector(predict(model,
                        newx = model.matrix(formula, val_fold)[, -1]))
} else if (method == 'adaptive_lasso' || method == 'adaptive_ridge'){
    preds <- as.vector(predict(model,
                        X = model.matrix(formula, val_fold)))
} else {
    preds <- predict(model, newdata = val_fold)
}
```

Assignment 3 Part 1 - build a generalized CV function

Penalized estimation - Stepwise regression - Adaptive regression

```
        # Take inverse trans; if not assigned, default: y |--> y
        y_val <- inv.trans(y_val)
        preds <- inv.trans(preds)
        # Performance metrics
        rmse_val <- RMSE(y_val, preds)
        mae_val <- MAE(y_val, preds)
        smdape_val <- sMdAPE(y_val, preds)
        MAPE_val <- MAPE(y_val, preds)
        LAR_val <- LAR(y_val, preds)
        perf_df <- rbind(perf_df, data.frame(
                                Fold = i, RMSE = rmse_val, MAE = mae_val,
                                sMdAPE = smdape_val, MAPE = MAPE_val, LAR = LAR_val))
    } # Close foreach loop
    mean_rmse <- mean(perf_df$RMSE)
    mean_mae <- mean(perf_df$MAE)
    mean_smdape <- mean(perf_df$sMdAPE)
    mean_MAPE <- mean(perf_df$MAPE)
    mean_LAR <- mean(perf_df$LAR)
    # Add the means to the df
    perf_df <- rbind(perf_df,
                    c("Mean", mean_rmse, mean_mae,
                      mean_smdape, mean_MAPE, mean_LAR))
    colnames(perf_df)[-1] <- paste0(name,
                            ' (', colnames(perf_df)[-1], ')')
    return (perf_df)  # Data.frame with each fold perf and the mean of them (last row)
}
```

Assignment 3 Part 1 - build a generalized CV function

Penalized estimation - Stepwise regression - Adaptive regression

Example

```
KfoldCVPerf.general(K=10, data=data_, formula=M5_formula, name='alasso-rqpen',
                    method='adaptive_lasso', criterion='rqpen',
    inv.trans= function (y) exp(y)) #inv.trans set to be exp() due to log-trans. of the response.
```

Fold	alasso-rqpen (**RMSE**)	alasso-rqpen (**MAE**)	alasso-rqpen (**sMdAPE**)	alasso-rqpen (**MAPE**)	alasso-rqpen (**LAR**)
1	12.9108	11.3189	34.5676	37.7134	0.3811
2	16.8527	13.1877	47.4686	48.9315	0.4145
3	36.7735	18.9935	29.4660	33.6621	0.3519
4	24.6673	15.2427	23.6478	26.2524	0.2978
5	13.8460	11.9184	35.0978	39.4078	0.4020
6	17.6953	12.0643	25.7870	38.5973	0.3945
7	25.8408	18.9851	41.6082	58.3033	0.5656
8	11.3753	8.3000	23.2837	37.5017	0.3083
9	13.3606	10.8264	30.3626	47.9084	0.3661
10	22.2944	15.4633	40.4748	166.0574	0.6024
Mean	19.5617	13.6300	33.1764	53.4335	0.4084

Assignment 3 Part 1 - Repeated cross-validation

Reducing the variance and impact of random data splits

```
RepeatedCV <- function (repeats=100, K=10, data, formula,
                        method, criterion, name,
                        inv.trans = function(y) y) {
    require(foreach)
    mean_rows_list <- list() # store the row_means
    # Choose diff random.sds in order to get diff folds
    foreach (seed = 1:repeats) %do% {
        cv_results <- KfoldCVPerf.general(K=K, data=data,
                                    formula=formula, name=name,
                                    folds.seed = seed, method=method,
                                    criterion=criterion, inv.trans=inv.trans)
        mean_row <- cv_results[cv_results[,1]=="Mean", -1]
        mean_rows_list[[seed]] <- as.numeric(mean_row)
    }
    # Combine the mean rows into a df
    mean_rows_df <- do.call(rbind, mean_rows_list)
    # Average for each metric
    average_metrics <- colMeans(mean_rows_df, na.rm = TRUE)
    average_metrics_df <- as.data.frame(t(average_metrics))
    colnames(average_metrics_df) <- names(cv_results)[-1]
    rownames(average_metrics_df) <- "RCV Average"

    return(average_metrics_df)
}
```

Repeated CV performance based on 15 repeats

Example

```
lasso.rcv <- RepeatedCV(repeats=15, K=10, data=data_, formula=M5_formula,
                  name='lasso', method='lasso', inv.trans= function (y) exp(y))
aLADlasso_hqreg.rcv <- RepeatedCV(repeats=15, K=10, data=data_, formula=M5_formula,
                  name='alasso-hqreg', method='adaptive_lasso',
                  criterion = 'hqreg', inv.trans= function (y) exp(y))
aLADlasso_rqpen.rcv <- RepeatedCV(repeats=15, K=10, data=data_, formula=M5_formula,
                  name='alasso-rqPen', method='adaptive_lasso',
                  criterion = 'rqpen', inv.trans= function (y) exp(y))
alasso_glmnet.rcv <- RepeatedCV(repeats=15, K=10, data=data_, formula=M5_formula,
                  name='alasso-glmnet', method='adaptive_lasso',
                  criterion = 'glmnet', inv.trans= function (y) exp(y))
```

lasso.rcv	lasso (RMSE)	lasso (MAE)	lasso (sMdAPE)	lasso (MAPE)	lasso (LAR)
RCV Average	19.5976	13.9733	33.7809	48.7838	0.4037

aLADlasso	alasso-hqreg (RMSE)	alasso-hqreg (MAE)	alasso-hqreg (sMdAPE)	alasso-hqreg (MAPE)	alasso-hqreg (LAR)
RCV Average	20.0593	14.5525	33.3904	66.3096	0.4303

aLADlasso	alasso-rqPen (RMSE)	alasso-rqPen (MAE)	alasso-rqPen (sMdAPE)	alasso-rqPen (MAPE)	alasso-rqPen (LAR)
RCV Average	19.3313	13.9133	34.7678	52.5467	0.4146

alasso	alasso-glmnet (RMSE)	alasso-glmnet (MAE)	alasso-glmnet (sMdAPE)	alasso-glmnet (MAPE)	alasso-glmnet (LAR)
RCV Average	19.7073	14.0543	33.7698	49.40478	0.4098

Assignment 3 Part 1 - 95%CI

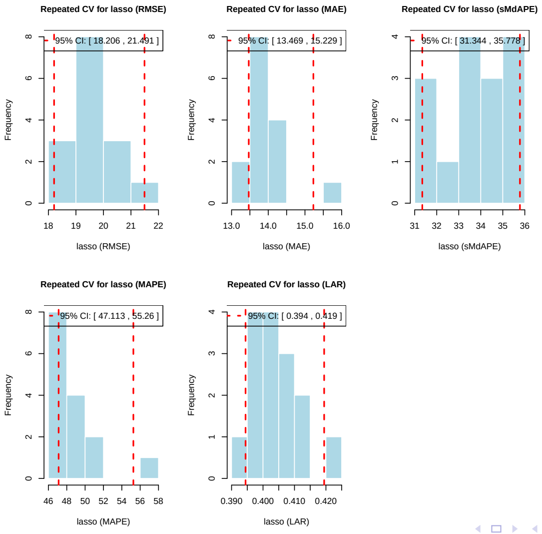
```r
RCV.histograms <- function (repeats=100, K=10, data, formula,
                            method, criterion, name, bins=5,
                            inv.trans = function(y) y) {
    require(foreach)
    metrics_values_list <- list()
    # Perform repeated cross-validation
    foreach (seed = 1:repeats) %do% {
        model.cv <- KfoldCVPerf.general(K=K, data=data,
                                    formula=formula, name=name, folds.seed = seed, method=method,
                                    criterion=criterion, inv.trans=inv.trans)
        # Extract the "Mean" row and store the metrics' values
        mean_row <- model.cv[model.cv[,1] == "Mean", -1]
        for (metric in names(mean_row)) {
            metrics_values_list[[metric]] <- c(metrics_values_list[[metric]],
                                             as.numeric(mean_row[metric]))
        }
    }
    # plot the histograms for each metric and add the CI lines
    par(mfrow=c(2, 3))  # Set up the plotting area to display multiple plots
    for (metric in names(metrics_values_list)) {
        # Calculate the 95% CI for the metric
        metric_values <- unlist(metrics_values_list[metric])
        ci <- quantile(metric_values, probs=c(0.025, 0.975))
        hist(metric_values, main=paste("Repeated CV for", metric),
             xlab=metric, col='lightblue', border='white', breaks=bins, cex.main = 0.965)
        abline(v=ci[1], col="red", lwd=2, lty=2) # CI lines
        abline(v=ci[2], col="red", lwd=2, lty=2) # CI lines
        legend("topright", legend=paste("95% CI: [", round(ci[1], 3), ',',
                              round(ci[2], 3), "]"),  col="red", lwd=2, lty=2)
    }
    par(mfrow=c(1, 1))
}
```

Assignment 3 Part 1 - 95%CI based on 15 repeats

```r
## Conf. ints based on RCV ###
lasso_rcv.hists <- RCV.histograms(repeats=15,
                   data=data_, formula=M5_formula,
                   name='lasso', method='lasso',
                   inv.trans= function (y) exp(y))

aLADlasso_hqreg_rcv.hists <- RCV.histograms(repeats=15,
                   data=data_, formula=M5_formula,
                   name='alasso-hqreg',
                   method='adaptive_lasso',
                   criterion = 'hqreg',
                   inv.trans= function (y) exp(y))

aLADlasso_rqpen_rcv.hists <- RCV.histograms(repeats=15,
                   data=data_, formula=M5_formula,
                   name='alasso-rqPen',
                   method='adaptive_lasso',
                   criterion = 'rqpen',
                   inv.trans= function (y) exp(y))

alasso_glmnet_rcv.hists <- RCV.histograms(repeats=15,
                   data=data_, formula=M5_formula,
                   name='alasso-glmnet',
                   method='adaptive_lasso',
                   criterion = 'glmnet',
                   inv.trans= function (y) exp(y))
```

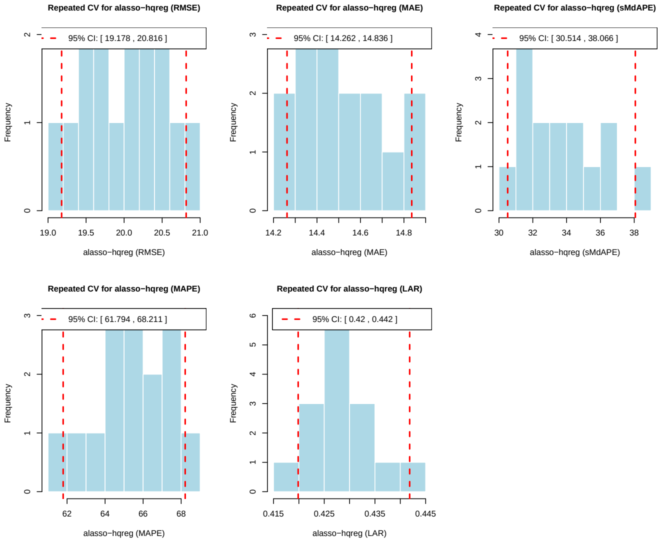
Assignment 3 Part 1 - 95%CI based on 15 repeats

```
lasso_rcv.hists
>>
```



Assignment 3 Part 1 - 95%CI based on 15 repeats

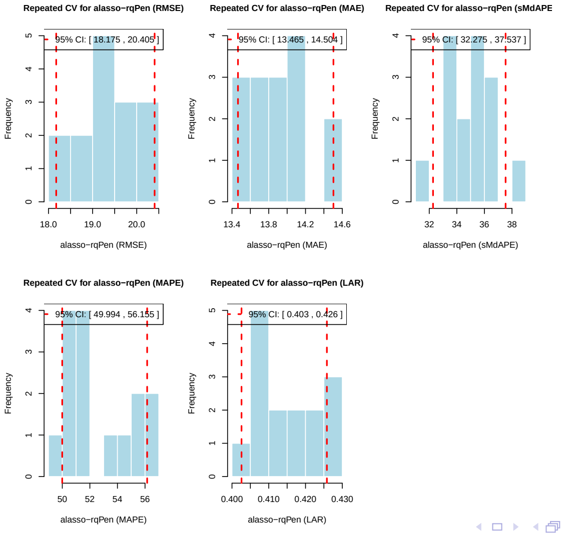
```
aLADlasso_hqreg_rcv.hists
>>
```



FORTH

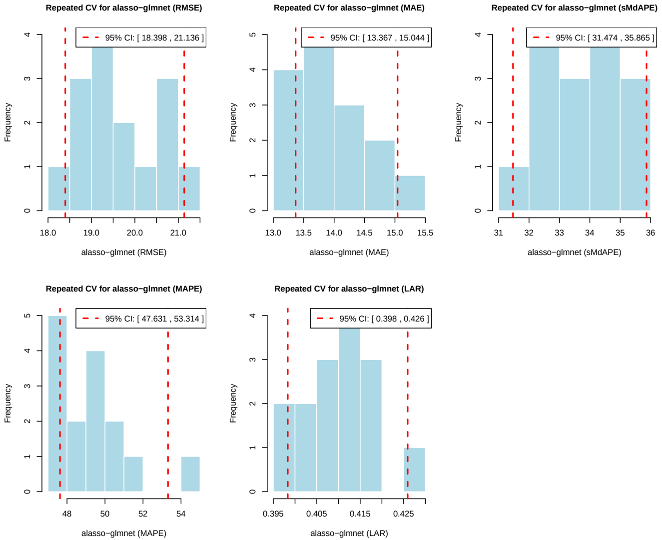
Assignment 3 Part 1 - 95%CI based on 15 repeats

```
aLADlasso_rqpen_rcv.hists
>>
```



Assignment 3 Part 1 - 95%CI based on 15 repeats

```
alasso_glmnet_rcv.hists
>>
```



Assignment 3 Part 1 - Compare models using box-plots

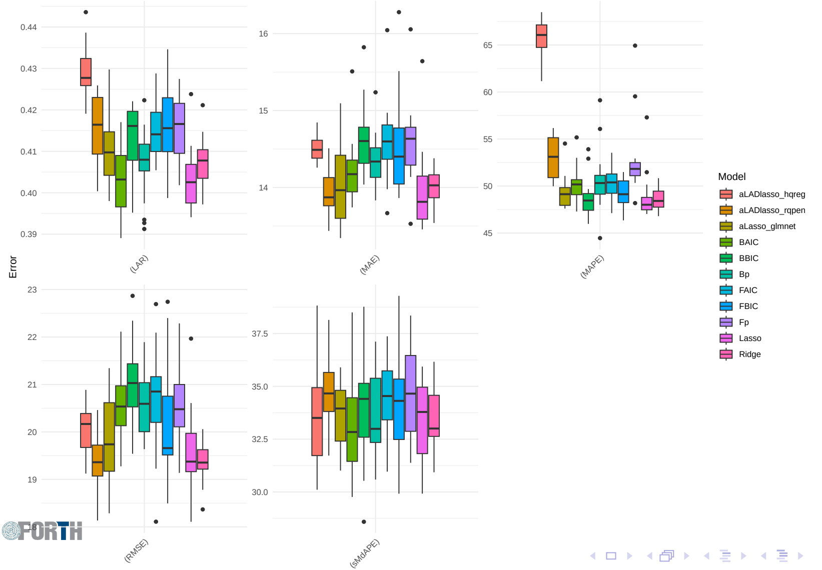
```r
get_RCV.boxplots <-  function (repeats, K, data, formula,
                               inv.trans =  function (y) y) {
    require(ggplot2)
    require(foreach)

    combined_metrics <- data.frame(metric = character(),
                                   value = numeric(),
                                   model = factor(),
                                   stringsAsFactors = FALSE)

    # List of models and their  inverse transformations and methods
    models_list <- list(
        FAIC = list(trans = inv.trans, method = "forward", criterion = "AIC"),
        FBIC = list(trans = inv.trans, method = "forward", criterion = "BIC"),
        BAIC = list(trans = inv.trans, method = "backward", criterion = "AIC"),
        BBIC = list(trans = inv.trans, method = "backward", criterion = "BIC"),
        Fp = list(trans = inv.trans, method = "forward", criterion = "p_val"),
        Bp = list(trans = inv.trans, method = "backward", criterion = "p_val"),
        Ridge = list(trans = inv.trans, method = "ridge"),
        Lasso = list(trans = inv.trans, method = "lasso"),
        aLADlasso_hqreg=list(trans=inv.trans, method="adaptive_lasso", criterion="hqreg"),
        aLADlasso_rqpen=list(trans=inv.trans, method="adaptive_lasso", criterion="rqpen"),
        aLasso_glmnet=list(trans=inv.trans, method="adaptive_lasso", criterion="glmnet")
    )
```

Assignment 3 Part 1 - Compare models using box-plots

```
    # Perform cross-validation and collect data
    for (model_name in names(models_list)) {
        foreach (seed = 1:repeats) %do% {
            cv_results <- KfoldCVPerf.general(K=K, data=data, formula=formula,
                                               name=model_name, folds.seed=seed,
                                               inv.trans=models_list[[model_name]]$trans,
                                               method=models_list[[model_name]]$method,
                                               criterion=models_list[[model_name]]$criterion)
            mean_row <- cv_results[cv_results[, 1] == "Mean", -1]
            # Append results to the combined metrics data frame
            combined_metrics <- rbind(combined_metrics, data.frame(
                metric = names(mean_row),
                value = as.numeric(mean_row),
                model = model_name
            ))
        }
    }
    # Extract only the metric name (after the space)
    combined_metrics$metric <- sapply(strsplit(combined_metrics$metric, " "), function (x) x[2])

    # Create the boxplot
    p <- ggplot(combined_metrics, aes(x = metric, y = value, fill = model)) +
        geom_boxplot() +
        facet_wrap(~ metric, scales = 'free') +
        theme_minimal() +
        theme(axis.text.x = element_text(angle = 45, hjust = 1),
              strip.text.x = element_blank()) +
        labs(x = NULL, y = "Error", fill = "Model")
    print(p)
}
```

Assignment 3 Part 1 - box-plots based on 20 RCV samples

```
get_RCV.boxplots(repeats=20, data=data_, formula=M5_formula, inv.trans= function (y) exp(y))
```



Assignment 3 Part 1 - box- plots based on 20 RCV samples

- We can observe that adaptive LAD lasso using `rqpen` generally outperforms the version using `hqreg`, showing better overall results in performance metrics.
- Furthermore, adaptive LAD lasso (`rqpen`) slightly outperforms adaptive lasso (via `glmnet`) in terms of MAE and RMSE, whereas in terms of log accuracy ratio (LAR), the opposite is observed.
- In terms of MAE and RMSE, the adaptive LAD lasso (`rqpen`) and the conventional lasso are quite similar, showing no substantial differences between them.

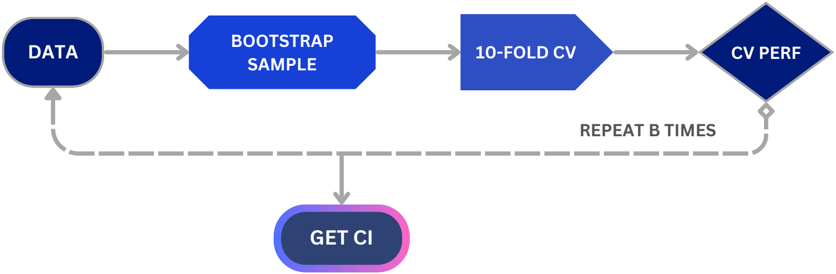
Assignment 3 Part 1 - box- plots based on 20 RCV samples

- We can observe that adaptive LAD lasso using `rqpen` generally outperforms the version using `hqreg`, showing better overall results in performance metrics.
- Furthermore, adaptive LAD lasso (`rqpen`) slightly outperforms adaptive lasso (via `glmnet`) in terms of MAE and RMSE, whereas in terms of log accuracy ratio (LAR), the opposite is observed.
- In terms of MAE and RMSE, the adaptive LAD lasso (`rqpen`) and the conventional lasso are quite similar, showing no substantial differences between them.

◇ Evaluate bootstrap-based confidence intervals for all three adaptive implementations. Display, side-by-side results for different solutions, using boxplots.

Bootstrapping based on CV performances

The following function generates bootstrap samples; for each sample, it derives a generalized performance measure through cross-validation. With these performance measures, one can calculate the corresponding confidence intervals for each performance metric. In the following figure, we can see the procedure described above.



Bootstrapping based on CV performances

```r
get.bootstrapCI <- function (B=100, data, formula, method, name,
                            criterion, inv.trans = function (y) y,
                            alpha = 0.05, plot.histogram = FALSE, bins = 10) {
  boot_stats <- matrix(NA, nrow = B, ncol = 5)
  colnames(boot_stats) <- c("RMSE", "MAE", "sMdAPE", "MAPE", "LAR")

  pb <- txtProgressBar(min = 0, max = B, style = 3) # Progress bar setup
  # Bootstrap loop
  for(b in 1:B) {
    setTxtProgressBar(pb, b)
    # Set a random state for each bootstrap sample
    set.seed(b)
    boot_data <- data[sample(nrow(data), replace = TRUE), ]
    # Get the performance metrics from the K-fold CV funct.
    perf_metrics <- KfoldCVPerf.general(K = 10, data = boot_data, formula = formula,
                                        method = method, criterion = criterion,
                                        inv.trans = inv.trans, name=name)
    boot_stats[b, ] <- as.numeric(perf_metrics[nrow(perf_metrics), -1])
  }
  close(pb) # Close progress bar
  if (!plot.histogram) {
    # Calc. the (1-0.05)*100% CI (set alpha=0.05 for 95% CI)
    ci_lower <- apply(boot_stats, 2, function (x) quantile(x, probs = alpha/2))
    ci_upper <- apply(boot_stats, 2, function (x) quantile(x, probs = 1 - alpha/2))
    CI <- data.frame(
      LowerCI = ci_lower, #lower
      UpperCI = ci_upper  #upper
    )
    return (CI)
```

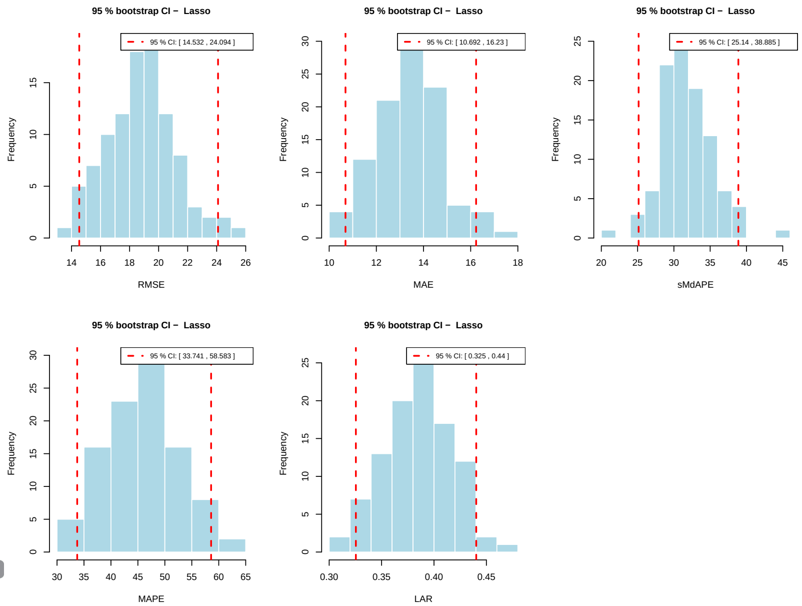
Bootstrapping based on CV performances

```
  } else {
    # Get the histograms
    par(mfrow=c(2, 3))
    for (i in 1:ncol(boot_stats)) {
      metric <- colnames(boot_stats)[i]
      metric_values <- boot_stats[, i]
      ci <- quantile(metric_values, probs=c(alpha/2, 1 - alpha/2))
      x_lim <- range(c(metric_values, ci[1], ci[2])) # CI in the range
      x_lim <- c(x_lim[1] - diff(x_lim) * 0.1, x_lim[2] + diff(x_lim) * 0.1)
      hist(metric_values, main=paste((1-alpha)*100, "% bootstrap CI - ", name),
           xlab=metric, col='lightblue', border='white',
           breaks=bins, cex.main = 0.956)
      # CI lines
      abline(v=ci[1], col="red", lwd=2, lty=2)
      abline(v=ci[2], col="red", lwd=2, lty=2)

      # Add a legend with the CI
      legend("topright", legend=paste((1-alpha)*100, "% CI: [", round(ci[1], 3),
                                    ',', round(ci[2], 3), "]"),
             col="red", lwd=2, lty=2, cex = 0.7)
    }
    par(mfrow=c(1, 1))
  }
}
```

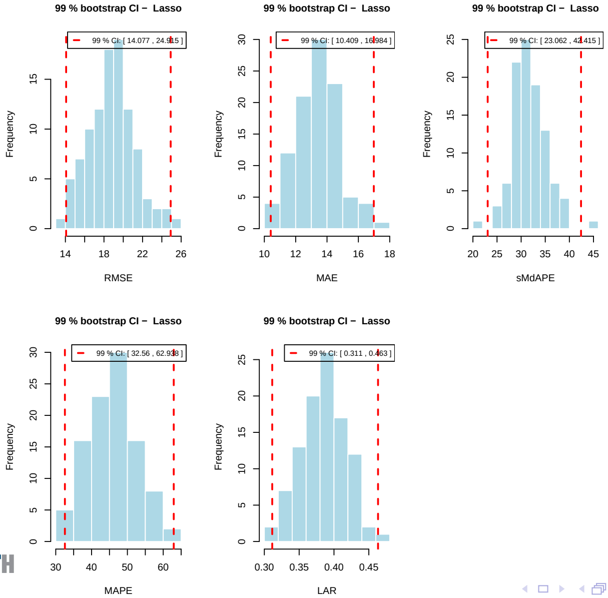
Generate 100 bootstrap samples

```
### This might take a while; be prepared to destroy your RAM ###
get.bootstrapCI(B=100, data=data_, formula=M5_formula,
                inv.trans=function(y) exp(y), method="lasso",
                plot.histogram = TRUE, name="Lasso")

get.bootstrapCI(B=100, data=data_, formula=M5_formula,
                inv.trans=function(y) exp(y), method="lasso",
                plot.histogram = TRUE, name="Lasso", alpha=0.01)

get.bootstrapCI(B=100, data=data_, formula=M5_formula,
                inv.trans=function(y) exp(y), method="adaptive_lasso",
                criterion="hqreg", plot.histogram = TRUE, name="aLADlasso (hqreg)")

get.bootstrapCI(B=100, data=data_, formula=M5_formula,
                inv.trans=function(y) exp(y), method="adaptive_lasso",
                criterion="hqreg", plot.histogram = TRUE, name="aLADlasso (hqreg)", alpha=0.01)

get.bootstrapCI(B=100, data=data_, formula=M5_formula,
                inv.trans=function(y) exp(y), method="adaptive_lasso",
                criterion="rqpen", plot.histogram = TRUE, name="aLADlasso (rqpen)")

get.bootstrapCI(B=100, data=data_, formula=M5_formula,
                inv.trans=function(y) exp(y), method="adaptive_lasso",
                criterion="rqpen", plot.histogram = TRUE, name="aLADlasso (rqpen)", alpha=0.01)

get.bootstrapCI(B=100, data=data_, formula=M5_formula,
                inv.trans=function(y) exp(y), method="adaptive_lasso",
                criterion="glmnet", plot.histogram = TRUE, name="aLasso (glmnet)")

get.bootstrapCI(B=100, data=data_, formula=M5_formula,
                inv.trans=function(y) exp(y), method="adaptive_lasso",
                criterion="glmnet", plot.histogram = TRUE, name="aLasso (glmnet)", alpha=0.01)
```

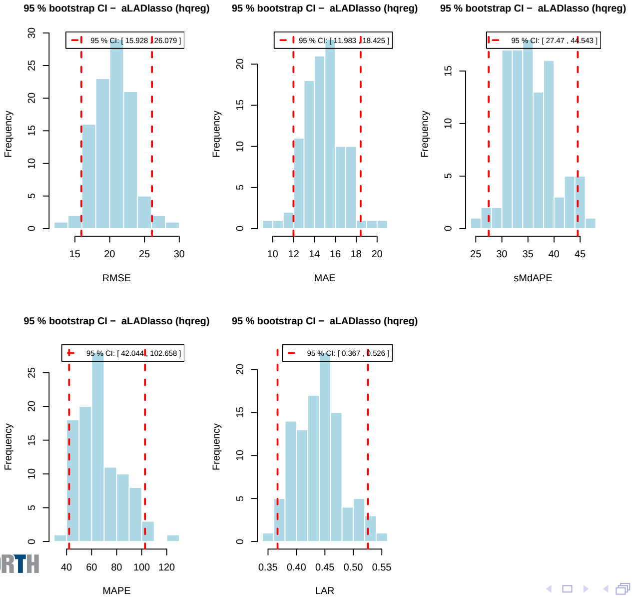
Bootstrapping based on CV performances - 95% CI LASSO



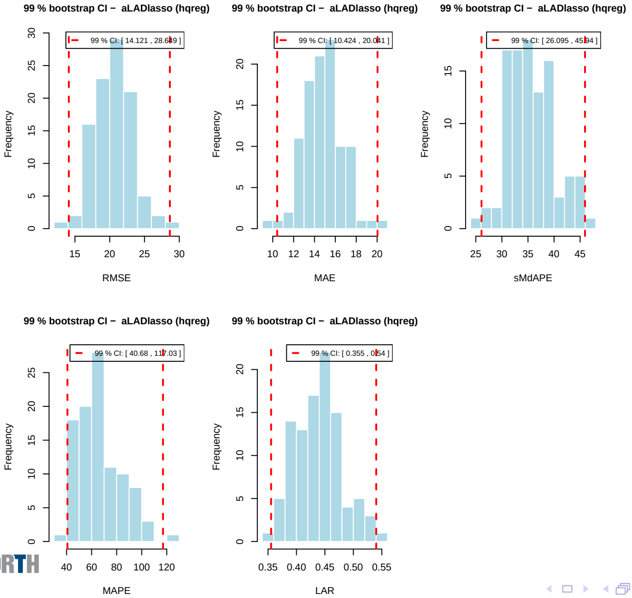
Bootstrapping based on CV performances - 99% CI LASSO



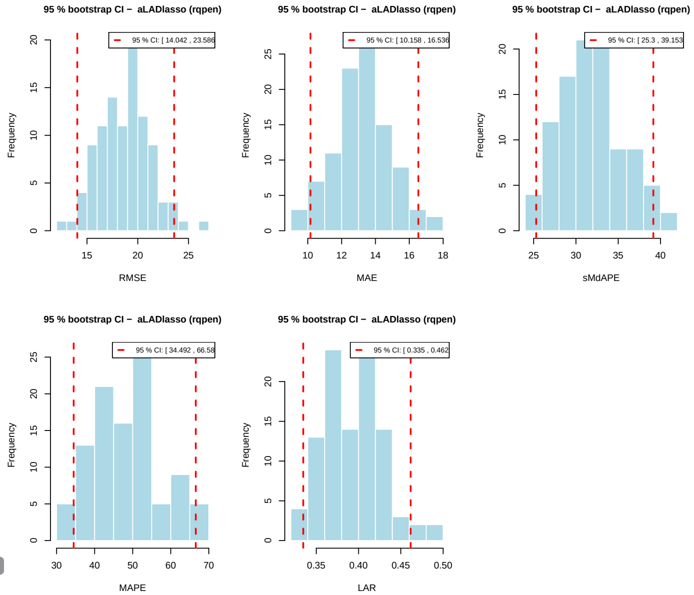
Bootstrapping based on CV performances - 95% CI hqreg



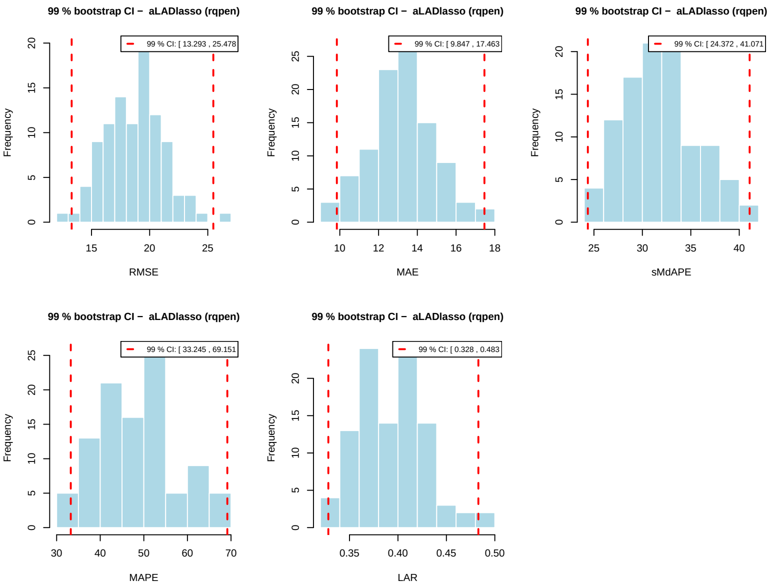
Bootstrapping based on CV performances - 99% CI hqreg



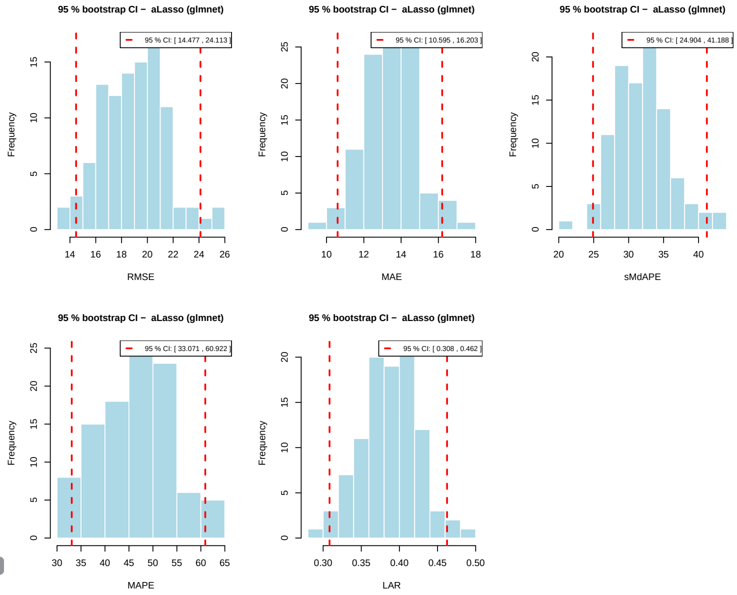
Bootstrapping based on CV performances - 95% CI rqpen



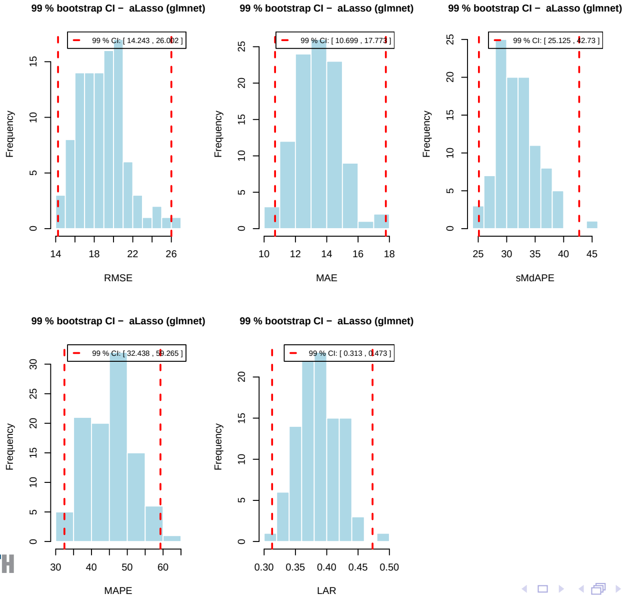
Bootstrapping based on CV performances - 99% CI rqpen



Bootstrapping based on CV performances - 95% CI glmnet



Bootstrapping based on CV performances - 99% CI glmnet



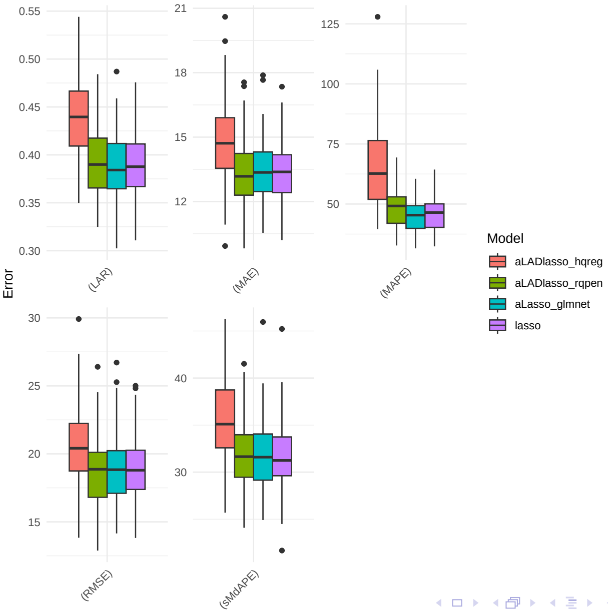
Compare models using box-plots based on bootstrap

```
get_boot.boxplots <- function(B=100, data, formula, inv.trans = function(y) y) {
  require(ggplot2)
  require(dplyr)
  require(foreach)
  # List of models to bootstrap (assuming same inv.trans)
  models_list <- list(
    lasso = list(method = "lasso", criterion = NA),
    aLADlasso_hqreg = list(method = "adaptive_lasso", criterion = "hqreg"),
    aLADlasso_rqpen = list(method = "adaptive_lasso", criterion = "rqpen"),
    aLasso_glmnet = list(method = "adaptive_lasso", criterion = "glmnet")
  )
  # Initialize a df to collect results
  combined_metrics <- data.frame(metric = character(),
                                 value = numeric(),
                                 model = character(),
                                 stringsAsFactors = FALSE)
  pb <- txtProgressBar(min = 0, max = length(models_list) * B, style = 3) # Progress bar
  progress <- 0
  # Bootstrap loop for each model
  for (model_name in names(models_list)) {
    foreach (b = 1:B) %do% {
      progress <- progress + 1
      setTxtProgressBar(pb, progress)
      set.seed(b)
      boot_data <- data[sample(nrow(data), replace = TRUE), ]
      cv_results <- KfoldCVPerf.general(K = 10, data = boot_data, formula = formula,
                                        method = models_list[[model_name]]$method,
                                        criterion = models_list[[model_name]]$criterion,
                                        inv.trans = inv.trans)
      # Last row of cv_results contains the mean performance metrics
      mean_metrics <- as.numeric(cv_results[nrow(cv_results), -1])
      metrics_names <- colnames(cv_results)[-1]
```

Compare models using box-plots based on bootstrap

```
      # Collect results
      for(i in seq_along(mean_metrics)) {
        combined_metrics <- rbind(combined_metrics, data.frame(
          metric = metrics_names[i],
          value = mean_metrics[i],
          model = model_name
        ))
      }
    }
  }
  close(pb) # close prog. bar
  combined_metrics$metric <- sapply(strsplit(combined_metrics$metric, " "), function (x) x[3])

  # Plot the box-plots foreach metric
  p <- ggplot(combined_metrics, aes(x = metric, y = value, fill = model)) +
        geom_boxplot(position=position_dodge(width=0.75)) +
        facet_wrap(~ metric, scales = 'free') +
        theme_minimal() +
        theme(axis.text.x = element_text(angle = 45, hjust = 1),
              strip.text.x = element_blank()) +
        labs(x = NULL, y = "Error", fill = "Model")

  print(p)
}

# Simulate 100 bootstrap samples
get_boot.boxplots(B = 100, data = data_, formula = M5_formula,
                  inv.trans = function (y) exp(y))
>>
```

Compare models; box-plots based on 100 bootstrap samples; lasso - adaptive lasso



Assignment 3 Part 2 - backward-stepwise median regression based on `AIC`

- Use `vif_func` as your basis, to develop a function for backward stepwise median regression based on AIC. For that purpose, use the formulation presented in Koenker's classic book titled "Quantile Regression" (2005) p. 135.
  - Implement your function on M5 and compare your results against the Adaptive LAD LASSO estimates.
  - Median regression estimates for M5, correspond to a modified loss function, relative to the one presented in Tofallis (2015, Section 3.3). Formulate this loss function.

Backward-stepwise median regression based on AIC

Koenker's definition of AIC

- The Akaike Information Criterion (AIC) for the j-th model in the context of quantile regression is given by:

$$\mathtt{AIC}(j) = \log(\hat{\sigma}_j) + p_j,$$

where:

- $\hat{\sigma}_j$ is the estimated scale parameter of the residuals for the j-th model.
- p_j is the number of parameters in the j-th model.
- The scale parameter $\hat{\sigma}_j$ is calculated as follows:

$$\hat{\sigma}_j = n^{-1} \sum_{i=1}^{n} \rho_{1/2}\left(y_i - x_i^T \hat{\beta}_n(1/2)\right),$$

where $\rho_{1/2}$ is the check function for the median regression, also known as quantile regression at $\tau = 0.5$.

Backward-stepwise median regression based on AIC

- $\hat{\beta}_n(\tau, \lambda)$ is defined as:

$$\hat{\beta}_n(\tau, \lambda) = \text{argmin}_{b \in \mathbb{R}^p} \left\{ \sum_{i=1}^{n} \rho_\tau(y_i - x_i^T b) + \lambda \|b - \beta_0\|_1 \right\}$$

- Quantile Regression and $\rho_{1/2}$: The check function ρ_τ for quantile regression at quantile τ is defined as:

$$\rho_\tau(u) = u \times (\tau - I(u < 0)).$$

For $\tau = 1/2$, which corresponds to the median regression, the function simplifies to

$$\rho_{1/2}(u) = |u|.$$

Data augmentation; go from LAD to LAD-Lasso with shrinkage towards $\hat{\beta}_0$ via data augmentation!

- **LAD-Lasso** coefficient estimates, with a penalty term that **shrinks** coefficients towards β_0, can be obtained by performing LAD regression on an **augmented** dataset.

Data augmentation; go from LAD to LAD-Lasso with shrinkage towards $\hat{\beta}_0$ via data augmentation!

- **LAD-Lasso** coefficient estimates, with a penalty term that **shrinks** coefficients towards β_0, can be obtained by performing LAD regression on an **augmented** dataset.
- **LAD-Lasso Regression with Shrinkage Towards β_0:** Objective function to minimize:

$$\min_{\beta \in \mathbb{R}^p} \left(\sum_{i=1}^{n} |y_i - X_i\beta| + \lambda \sum_{j=1}^{p} |\beta_j - \beta_{0j}| \right)$$

where β are the regression coefficients and β_{0j} is the j-th component of β_0.

Data augmentation; go from LAD to LAD-Lasso with shrinkage towards $\hat{\beta}_0$ via data augmentation!

- **LAD-Lasso** coefficient estimates, with a penalty term that **shrinks** coefficients towards β_0, can be obtained by performing LAD regression on an **augmented** dataset.
- **LAD-Lasso Regression with Shrinkage Towards β_0:** Objective function to minimize:

$$\min_{\beta \in \mathbb{R}^p} \left(\sum_{i=1}^{n} |y_i - X_i\beta| + \lambda \sum_{j=1}^{p} |\beta_j - \beta_{0j}| \right)$$

where β are the regression coefficients and β_{0j} is the j-th component of β_0.

Augmentation Process:

- Augment X to X^* by adding λI below X, where I is the $p \times p$ identity matrix; $X^* \in \mathbb{R}^{(n+p)\times p}$.
- Augment y to y^* by adding $\lambda(\beta_0)_j$; $y^* \in \mathbb{R}^{n+p}$.

Data augmentation; go from LAD to LAD-Lasso with shrinkage towards $\hat{\beta}_0$ via data augmentation

Augmented LAD Regression: New objective function to minimize:

$$\min_{\beta \in \mathbb{R}^p} \sum_{i=1}^{n+p} |y_i^* - X_i^* \beta|$$

For the added p rows, the term $|y_i^* - X_i^* \beta|$ simplifies to $|\lambda \beta_j - \lambda \beta_{0j}|$ for each $j \in \{1, 2, \ldots, p\}$.

Result: The objective function for augmented LAD regression becomes:

$$\sum_{i=1}^{n} |y_i - X_i \beta| + \sum_{j=1}^{p} |\lambda \beta_j - \lambda \beta_{0j}| = \|y - X\beta\|_1 + \lambda \|\beta - \beta_0\|_1,$$

which is equivalent to the LAD-Lasso objective function with shrinkage towards β_0. Therefore, solving the augmented LAD regression yields the same coefficient estimates as LAD-Lasso with shrinkage towards β_0!

Backward-stepwise median regression based on AIC

Algorithm Backward Median Regression; based on Koenker's AIC

```
Require: Data matrix X, response vector y, initial set of predictors P
  currentModel ← fit median regression on all of P
  currentAIC ← AIC_Koenkers(currentModel)
  while there are predictors left in P do
    bestModel ← currentModel
    bestAIC ← currentAIC
    for each predictor p_i in P do
      tempModel ← fit median regression without p_i
      tempAIC ← AIC_Koenkers(tempModel)
      if tempAIC < bestAIC then
        bestModel ← tempModel
        bestAIC ← tempAIC
      end if
    end for
    if bestAIC < currentAIC then
      currentModel ← bestModel
      currentAIC ← bestAIC
      P ← P \ { predictors in bestModel }
    else
      break
    end if
  end while
  return currentModel
```

Construct Koenker's LAD-Lasso with shrinkage $\rightarrow \hat{\beta}_0$ in R

```
LADlasso.shrinkage <- function(formula, data, lambda) {
    require(quantreg)
    X <- as.data.frame(model.matrix(formula, data))[,-1]
    y <- data[[all.vars(formula)[1]]]
    n <- nrow(X)
    p <- ncol(X)

    # Data augmentation
    diag_p_scaled <- lambda * diag(p)
    diag_p <- as.data.frame(diag_p_scaled) # Convert to data frame
    colnames(diag_p) <- names(X)

    # Fit a standard LAD-Lasso model to determine beta0
    cv_fit_init <- rq(formula, data = data, method = 'lasso', tau = 0.5)
    beta0 <- coef(cv_fit_init, s=cv_fit_init$lambda.min)[-1]
    # Augment X and y for the LAD-Lasso problem with shrinkage towards beta0
    X_aug <- rbind(X, diag_p)
    y_aug <- c(y, as.vector(lambda * beta0))

    # Fit the final augmented LAD-Lasso model
    final.formula <- as.formula(paste("y_aug ~ ", paste(names(X_aug), collapse="+")))
    final_fit <- rq(final.formula, data = cbind(y_aug = y_aug, X_aug), tau = 0.5)

    return(list(coef = final_fit$coef, fit = final_fit))
}
```

Example

```
lls <- LADlasso.shrinkage(M5_formula, data_, lambda=0.01)
summary(lls$fit)$coeff
>>
```

Term	Coefficients	Lower Bound	Upper Bound
(Intercept)	1.92e-23	1.92e-23	1.92e-23
Solar.R	8.00e-03	-1.29e-02	1.66e-02
Wind	-2.35e-01	-5.21e-01	2.67e-01
Temp	4.76e-02	9.09e-03	1.16e-01
I(Solar.R^2)	-1.21e-05	-2.83e-05	-6.58e-06
I(Wind^2)	7.00e-03	-5.46e-04	1.24e-02
I(Temp^2)	5.75e-05	-7.64e-05	3.64e-04
Z1	-2.63e-02	-1.94e-01	8.10e-02
Z2	8.92e-02	-1.02e-01	2.04e-01
Z3	-1.11e-01	-1.93e-01	9.45e-02
Solar.R:Wind	-8.05e-05	-5.44e-05	3.94e-04
Solar.R:Temp	-5.34e-06	-1.44e-06	3.28e-04
Wind:Temp	1.08e-04	-4.94e-03	2.18e-03

Construct Koenker's LAD-Lasso with shrinkage $\rightarrow \hat{\beta}_0$ in R

Tune lambda via cross-validation & get residuals ($\hat{\sigma}_n = n^{-1} \cdot$ sum(|residuals|))

```
cv_LADlasso.shrinkage <- function (formula, data, K = 10, folds.seed = 42,
                                    lambda_seq = seq(0.001, 5, by = 0.05)) {
    require(foreach)
    set.seed(folds.seed)
    data <- data[sample(nrow(data)), ] # Shuffle data
    binning_variable <- cut(seq(1, nrow(data)), breaks = K, labels = FALSE)
    folds <- split(data, binning_variable)
    # Store performance metrics for each lambda
    performance <- matrix(NA, nrow = length(lambda_seq), ncol = K)

    foreach (i = 1:K) %do% {
        # Training and validation sets
        validation_set <- folds[[i]]
        training_set <- do.call(rbind, folds[-i])  # Combine all other folds

        foreach (j = seq_along(lambda_seq)) %do% {
            lambda <- lambda_seq[j]
            # Fit the model on the training set
            model_fit <- LADlasso.shrinkage(formula, training_set, lambda)
            # Evaluate the model on the validation set
            # Apply the transformations to the validation set
            X_val <- as.data.frame(model.matrix(formula, validation_set))[,-1]
            y_val <- validation_set[[all.vars(formula)[1]]]

            pred <- predict(model_fit$fit, newdata = X_val)
            performance[j, i] <- MAE(y_val, pred)
        }
    }
```

Construct Koenker's LAD-Lasso with shrinkage $\rightarrow \hat{\beta}_0$ in R

Tune lambda via cross-validation & get residuals ($\hat{\sigma}_n = n^{-1} \cdot$ sum(|residuals|))

```
    avg_performance <- apply(performance, 1, mean) # Mean perf across folds for each lambda
    best_lambda <- lambda_seq[which.min(avg_performance)]
    cv.fit <- LADlasso.shrinkage(formula, data, best_lambda)
    beta_hat_n <- cv.fit$coef[-1]
    X <- model.matrix(formula, data)[,-1]
    y <- data[[all.vars(formula)[1]]]
    residuals <- y - X %*% beta_hat_n # y - X·beta_hat_n
    return(list(lambda.tuned = best_lambda, cvfit = cv.fit,
                lambdas = lambda_seq, residuals = residuals))
}
```

Example

```
LADLassoS.cv <- cv_LADlasso.shrinkage(M5_formula, data_)
LADLassoS.cv$lambda.tuned
>> 3.451
dim(LADLassoS.cv$residuals)
>>  111  ·  1
head(LADLassoS.cv$residuals)
```

Index	Value
14	-3.31e-01
68	-1.30e-01
120	-2.66e-15
29	5.22e-01
16	-3.11e-15
109	8.03e-01

Define Koenker's information criteria

AIC (Akaike); AICc (corrected Akaike); SIC (Schwarz)

```
AIC.Koenker <- function(fitted_model) {
  sigma_hat <- mean(abs(fitted_model$residuals))
  # Number of non-zero coefficients
  p_j <- sum(fitted_model$cvfit$coef != 0) - 1
  AIC.value <- log(sigma_hat) + p_j

  return(AIC.value)
}
AICc.Koenker <- function(fitted_model) {
  n <- length(fitted_model$residuals) # get nrow size
  p_j <- sum(fitted_model$cvfit$coef != 0) - 1
  AICc.value <- AIC.Koenker(fitted_model) + (2*p_j*(p_j + 1))/(n - p_j - 1)

  return(AICc.value)
}
# Schwarz Information Criterion (or BIC)
SIC.Koenker <- function(fitted_model) {
  sigma_hat <- mean(abs(fitted_model$residuals))
  n <- length(fitted_model$residuals) # get nrow size
  # Number of non-zero coefficients
  p_j <- sum(fitted_model$cvfit$coef != 0) - 1
  SIC.value <- log(sigma_hat) + 0.5 * p_j * log(n)

  return(SIC.value)
}
```

Construct backward procedure based on Koenker's IC in R

```r
backward.Koenker <- function(formula, data, IC = AIC.Koenker, threshold = 0.01) {
    require(foreach)
    y <- all.vars(formula)[1] # Get response name (str)
    P <- colnames(model.matrix(formula, data)[, -1]) # Predictors (no intercept)

    current.model <- cv_LADlasso.shrinkage(formula, data)
    current.IC <- IC(current.model)
    steps <- data.frame(step = "Initial model", IC = current.IC, delta_IC = '')

    while (length(P) > 0) {
        best.IC <- current.IC
        best.model <- current.model
        removed_predictor <- NULL

        foreach (p_i = P) %do% {
            # Skip the iteration if removing p_i results in a singular matrix
            tryCatch({
                P_minus_p_i.formula <- as.formula(paste(y, '~',
                                                  paste(setdiff(P, p_i), collapse=' + ')))
                temp.model <- cv_LADlasso.shrinkage(P_minus_p_i.formula, data,
                                                    lambda_seq = seq(0.05, 2, by = 0.5))
                temp.IC <- IC(temp.model)

                if (temp.IC < best.IC) {
                    best.model <- temp.model
                    best.IC <- temp.IC
                    removed_predictor <- p_i
                }
            }, error = function(e) {
                return
            })
        }
```

Construct backward procedure based on Koenker's IC in R

```
        # Stepwise with early-dropping; if delta gets < threshold; stop
        if (best.IC < current.IC - threshold && !is.null(removed_predictor)) {
            current.model <- best.model
            current.IC <- best.IC
            P <- setdiff(P, removed_predictor)
            delta_ic <- steps[['IC']][length(steps[['IC']])] - best.IC # prev_IC - curr_IC
            steps <- rbind(steps, data.frame(step = paste('-', removed_predictor),
                                             IC = best.IC, delta_IC = delta_ic))
        } else {
            break  # No significant improvement
        }
    }
    return(list(model = current.model, steps = steps))
}
```

Example

```
system.time({
    LADlassoShr_backward.AIC <- backward.Koenker(M5_formula, data_, IC=AIC.Koenker)

    LADlassoShr_backward.AICc <- backward.Koenker(M5_formula, data_, IC=AICc.Koenker)

    LADlassoShr_backward.SIC <- backward.Koenker(M5_formula, data_, IC=SIC.Koenker)
})

>>     user  system elapsed
>>   580.946 132.744  98.183
```

Construct backward procedure based on Koenker's IC in R

```
LADlassoShr_backward.AIC$step
LADlassoShr_backward.AICc$step
LADlassoShr_backward.SIC$step
>>
```

Step	AIC	ΔAIC
Initial model	10.982	
- Solar.R:Temp	9.967	1.015
- Wind:Temp	8.967	0.999
- I(Temp^2)	7.968	0.999
- Temp	6.197	1.770
- I(Wind^2)	5.206	0.990
- Z1	4.226	0.979
- I(Solar.R^2)	3.311	0.915

Step	AICc	ΔAICc
Initial model	14.166	
- Solar.R:Temp	12.633	1.532
- Wind:Temp	11.167	1.467
- I(Temp^2)	9.750	1.417
- Temp	7.284	2.465
- I(Wind^2)	6.014	1.270
- Z1	4.798	1.216
- I(Solar.R^2)	3.689	1.108

Step	SIC	ΔSIC
Initial model	27.397	
- Solar.R:Temp	24.869	2.370
- Wind:Temp	22.515	2.354
- I(Temp^2)	20.161	2.354
- Temp	15.680	4.481
- I(Wind^2)	13.335	2.345
- Z1	11.003	2.334
- I(Solar.R^2)	8.731	2.269

- We can observe that each **IC** results in the removal of the same predictors.

CV for backward-stepwise median regression

We can extend the `KfoldCVPerf.general()` function in order to include the backward-stepwise median regression for LAD lasso with shrinkage towards $\hat{\beta}_0$.

```
    ###  ... same code ...  ###

} else if (method == "BLADlasso.shr") {
       require(doMC)
       registerDoMC(cores = 2)
       backward <- backward.Koenker(formula, as.data.frame(train_folds),
                             IC=criterion, # Koenker's IC
                             threshold=early.drop)
       model <- backward$model$cvfit$fit

} else {
    stop("Unsupported method or criterion.")
}
rm(train_folds_temp, envir = .GlobalEnv) # Remove globenv
# Determine datatypes in order to predict()
if (method == "BLADlasso.shr") {
    preds <- as.vector(predict(model, newdata = as.data.frame(val_fold)))
} else if (method %in% c("ridge", "lasso") || criterion == "glmnet") {
    suppressWarnings({
    preds <- as.vector(predict(model,
                    newx = model.matrix(formula, val_fold),
                    s = "lambda.min"))
                    })

    ###  ... same code ...  ###
```

```
KfoldCVPerf.general(K=10, data=data_, formula=M5_formula,
                    name='BAIC.LADlasso.shr', method='BLADlasso.shr',
                    criterion = AIC.Koenker, inv.trans= function (y) exp(y))
>>
```

Fold	BAIC.LADlasso.shr (RMSE)	BAIC.LADlasso.shr (MAE)	BAIC.LADlasso.shr (sMdAPE)	BAIC.LADlasso.shr (MAPE)	BAIC.LADlasso.shr (LAR)
1	19.495	15.483	38.842	50.929	0.481
2	15.597	12.271	45.634	46.694	0.391
3	31.737	19.594	36.062	38.504	0.406
4	21.199	13.602	25.035	27.376	0.339
5	19.043	13.819	37.951	45.718	0.408
6	16.177	11.367	30.666	34.509	0.362
7	25.33	19.293	44.069	59.516	0.57
8	13.735	10.861	22.645	40.399	0.318
9	15.776	12.368	30.129	44.314	0.371
10	25.895	18.793	33.848	109.948	0.59
Mean	**20.398**	**14.745**	**34.488**	**49.787**	**0.424**

Backward-stepwise median regression based on AIC with **early dropping** - Cross-validation performance

```
KfoldCVPerf.general(K=10, data=data_, formula=M5_formula, name='BAIC.LADlasso.shr.ED',
                    method='BLADlasso.shr', criterion = AIC.Koenker,
                    early.drop = 0.95, inv.trans= function (y) exp(y))
>>
```

Fold	BAIC.LADlasso.shr.ED (RMSE)	BAIC.LADlasso.shr.ED (MAE)	BAIC.LADlasso.shr.ED (sMdAPE)	BAIC.LADlasso.shr.ED (MAPE)	BAIC.LADlasso.shr.ED (LAR)
1	19.496	15.483	38.842	50.929	0.481
2	15.597	12.271	45.634	46.694	0.391
3	31.737	19.594	36.062	38.504	0.406
4	21.199	13.602	25.035	27.376	0.339
5	27.497	17.940	37.419	48.794	0.523
6	16.177	11.367	30.666	34.509	0.362
7	25.330	19.293	44.069	59.516	0.570
8	11.591	7.717	13.570	36.034	0.286
9	15.777	12.368	30.129	44.314	0.371
10	25.896	18.793	33.848	109.948	0.590
Mean	**21.029**	**14.843**	**33.528**	**49.658**	**0.432**

```
KfoldCVPerf.general(K=10, data=data_, formula=M5_formula,
                    name='BAICc.LADlasso.shr', method='BLADlasso.shr',
                    criterion = AICc.Koenker, inv.trans= function(y) exp(y))
>>
```

Fold	BAICc.LADlasso.shr (RMSE)	BAICc.LADlasso.shr (MAE)	BAICc.LADlasso.shr (sMdAPE)	BAICc.LADlasso.shr (MAPE)	BAICc.LADlasso.shr (LAR)
1	19.495	15.483	38.842	50.929	0.481
2	15.597	12.271	45.634	46.694	0.391
3	31.737	19.594	36.062	38.504	0.406
4	21.199	13.602	25.035	27.376	0.339
5	19.043	13.819	37.951	45.718	0.408
6	16.177	11.367	30.666	34.509	0.362
7	25.33	19.293	44.069	59.516	0.57
8	13.735	10.861	22.645	40.399	0.318
9	15.776	12.368	30.129	44.314	0.371
10	25.895	18.793	33.848	109.948	0.59
Mean	**20.398**	**14.745**	**34.488**	**49.787**	**0.424**

Backward-stepwise median regression based on SIC

Cross-validation performance

```
KfoldCVPerf.general(K=10, data=data_, formula=M5_formula,
                    name='BSIC.LADlasso.shr', method='BLADlasso.shr',
                    criterion = SIC.Koenker, inv.trans= function(y) exp(y))
>>
```

Fold	BSIC.LADlasso.shr (RMSE)	BSIC.LADlasso.shr (MAE)	BSIC.LADlasso.shr (sMdAPE)	BSIC.LADlasso.shr (MAPE)	BSIC.LADlasso.shr (LAR)
1	19.495	15.483	38.842	50.929	0.481
2	15.597	12.271	45.634	46.694	0.391
3	31.737	19.594	36.062	38.504	0.406
4	21.199	13.602	25.035	27.376	0.339
5	19.043	13.819	37.951	45.718	0.408
6	16.177	11.367	30.666	34.509	0.362
7	25.33	19.293	44.069	59.516	0.57
8	13.735	10.861	22.645	40.399	0.318
9	15.776	12.368	30.129	44.314	0.371
10	25.895	18.793	33.848	109.948	0.59
Mean	**20.398**	**14.745**	**34.488**	**49.787**	**0.424**

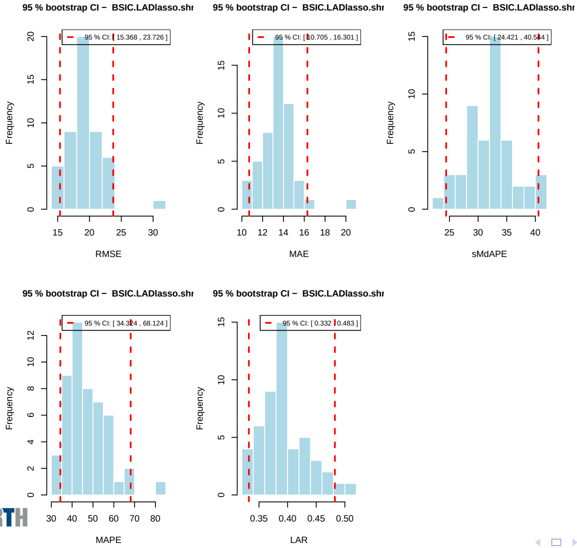
Backward-stepwise median regression; observations

Cross-validation performance

- One can observe that the backward-stepwise LAD lasso behaves similarly for each `IC` (Akaike, corrected Akaike, Schwarz).
- Backward-stepwise with **early-dropping** (at 0.95 in our case) performance seems to be slightly worse, but much faster.
- Keep in mind that for each fold, we fit a backward-stepwise LAD lasso with shrinkage towards $\hat{\beta}_0$; this is computationally expensive. It's a nested CV scenario due to the lambda hyperparameter tuning, and the fact that we need to calculate $\hat{\beta}_0$ makes the procedure even slower.
- The results are very promising, compared to the previous models and very close to the **BAIC**/**BBIC** from the `olsr` package (using conventional `AIC` and `BIC`).

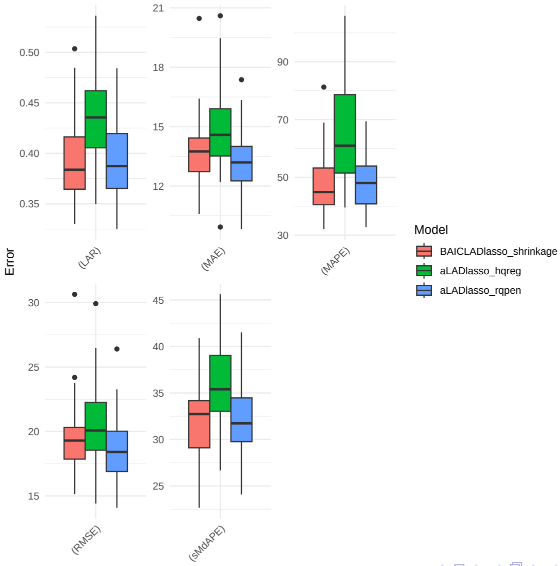
FORTH

Bootstrapping based on CV performances - 50 bootstrap samples - 95% CI `Backward.AIC.LADlasso.shrinkage`



Compare LAD Lasso models; box-plots based on 50 bootstrap samples

Adaptive LAD Lasso VS Backward.AIC.LAD.Lasso.shrinkage.$\hat{\beta}_0$ (system.time ≈8 hours required)



Bootstrapping the estimating coefficients

```r
boot_coeffs.boxplots <- function (B, data, formula, method) {
  require(ggplot2)
  require(dplyr)
  require(foreach)
   coeffs_names <- colnames(model.matrix(formula, data))
   Fit.model <- function (formula, data, method) {
     X <- model.matrix(formula, data)[,-1]
     y <- data[[all.vars(formula)[1]]]
     if (method == "aLADlasso.hqreg") {
        require(hqreg)
        quiet <- function (x) { sink(tempfile())
                                  on.exit(sink())
                                  invisible(force(x))
                                 } # Hides the CV messeges.
        # Fit an initial model to get coefficient estimates
        initial_fit <- quiet(
                    cv.hqreg(X = X,
                     y = y,
                     method = 'quantile',
                     alpha = 1, nfolds = 10,
                     type.measure = 'mae',
                     tau = 0.5, seed = 42))
        initial_coeff <- coef(initial_fit, s=initial_fit$lambda.min)
        # Calculate adaptive weights
        # Avoid div by zero - substitute very small values for 0 coefs
        initial_coeff[initial_coeff == 0] <- 1e-5
        n <- dim(data)[1]
        weights <- 1 / (abs(initial_coeff[-1]) + 1/n)
```

Bootstrapping the estimating coefficients

```
    # Fit the final Adaptive LAD Lasso model (CV for lambda)
    model <- quiet(
            cv.hqreg(X = X,
                   y = y,
                   method = 'quantile',
                   tau = 0.5,
                   type.measure = 'mae',
                   penalty.factor = weights,
                   alpha = 1, seed=42))
   coeffs <- coef(model, s = model$lambda.min)
 } else if (method == "alasso.glmnet") {
    require(glmnet)
    # Fit an initial model to get coefficient estimates
    initial_fit <- cv.glmnet(
                  x = X,
                  y = y,
                  type.measure = 'mse',
                  alpha = 1, nfolds = 10,
                  seed = 42, parallel = TRUE)
    initial_coeff <- coef(initial_fit, s=initial_fit$lambda.min)
    # Calculate adaptive weights
    # Avoid div by zero - substitute very small values for 0 coefs
    initial_coeff[initial_coeff == 0] <- 1e-5
    n <- dim(data)[1]
    weights <- 1 / (abs(initial_coeff[-1]) + 1/n)
    # Fit the final Adaptive Lasso model (tune lambda via CV)
    model <- cv.glmnet(x = X,
                       y = y,
                       type.measure = 'mse', penalty.factor = weights,
                       alpha = 1, seed=42, parallel = TRUE)

    coeffs <- as.vector(as.matrix(coef(model, s = model$lambda.min)))
```

Bootstrapping the estimating coefficients

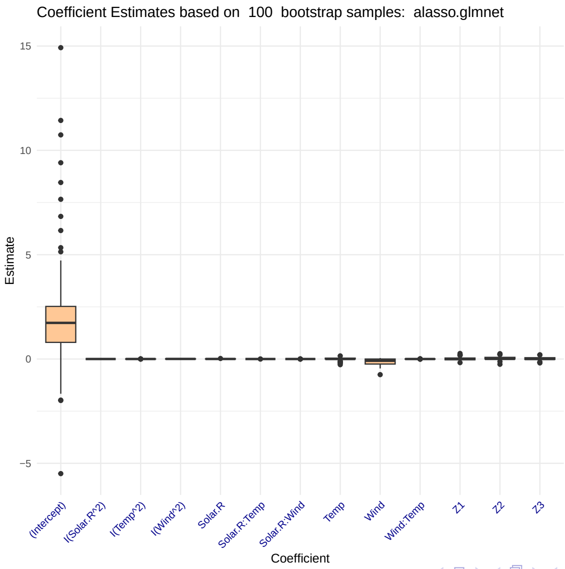
```r
  } else if (method == "aLADlasso.rqpen") {
        require(rqPen)
        require(doMC)
        registerDoMC(cores = 2) # Parallel backend
        suppressWarnings({
            system.time(
                model <- rq.pen.cv(
                  x = X,
                  y = y,
                  tau = 0.5, penalty = "aLASSO", cvSummary=median, nfolds = 10))
            })
      lambda.min <- model$btr[[4]]
      coeffs <- model$fit$models[[1]]$coef[,lambda.min]
  } else if (method == "BLADlasso.shr") {
        require(doMC)
        require(foreach)
        registerDoMC(cores = 2)
        backward <- backward.Koenker(formula, data,
                                     IC=AIC.Koenker, # Koenker's IC
                                     threshold=0.01)
        model <- backward$model$cvfit$fit
        coeffs <- coef(model)
  }
  return (coeffs)
}
combined_coeffs <- data.frame(coefficient = character(), value = numeric(),
                              stringsAsFactors = FALSE)
pb <- txtProgressBar(min = 0, max = B, style = 3)
```

Bootstrapping the estimating coefficients

```
  for (b in 1:B) {
    setTxtProgressBar(pb, b)
    set.seed(b)
    boot_data <- data[sample(nrow(data), replace = TRUE), ]
    coeffs <- Fit.model(formula, boot_data, method)
    for(i in seq_along(coeffs)) {
      combined_coeffs <- rbind(combined_coeffs,
                               data.frame(coefficient = coeffs_names[i],
                                          value = coeffs[i]))
    }
  }
  close(pb)
  p <- ggplot(combined_coeffs, aes(x = coefficient, y = value)) +
      geom_boxplot(fill = rgb(255, 200, 150, maxColorValue = 255)) +
      theme_minimal() +
      theme(axis.text.x = element_text(angle = 45, hjust = 1, color = "darkblue"),
            strip.text.x = element_blank()) +
      labs(x = "Coefficient", y = "Estimate",
           title = paste("Coefficient Estimates based on ",
                         B, " bootstrap samples: ", method))
  print(p)
  dev.off()
}

# Example
boot_coeffs.boxplots(B = 100, data = data_, formula = M5_formula, method = "alasso.glmnet")
>>
```

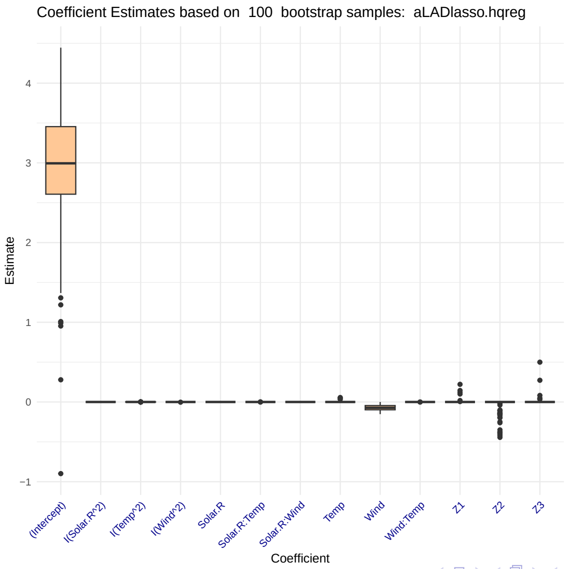
Bootstrapping the estimating coefficients

Adaptive lasso - glmnet



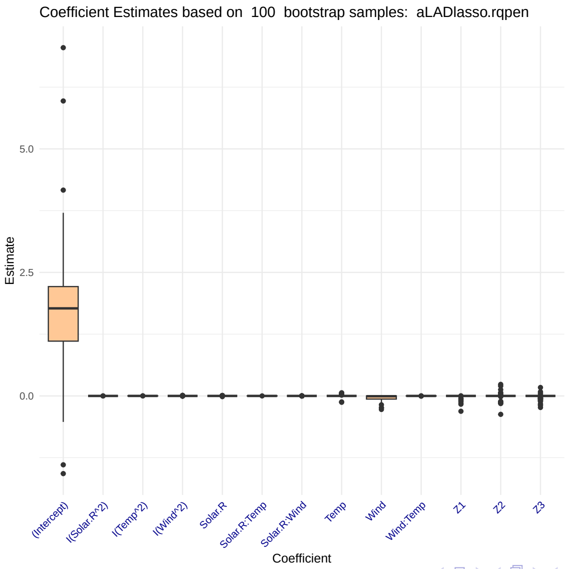
Bootstrapping the estimating coefficients

Adaptive LAD lasso - `hqreg`



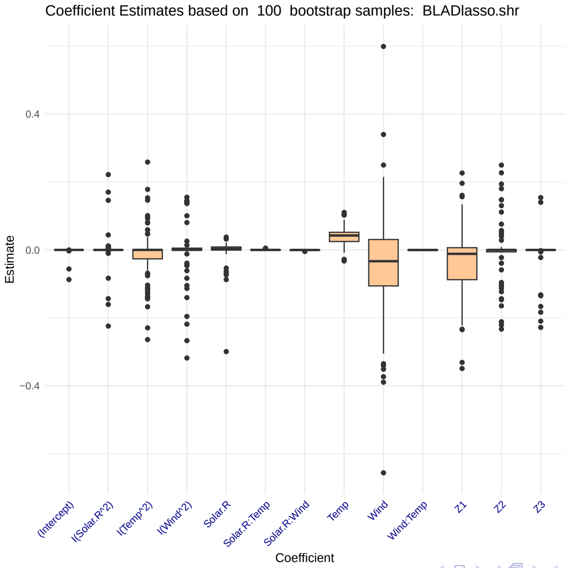
Bootstrapping the estimating coefficients

Adaptive LAD lasso - `rqpen`



Bootstrapping the estimating coefficients

Backwards LAD lasso - AIC.Koenker's (shrinkage $\rightarrow \hat{\beta}_0$)



Bootstrapping the estimating coefficients - observations

- **For `BLADlasso.shr`:**
  - The `` `Wind` ``, `` `Temp` ``, `` `I(Temp2)` ``, `` `Z1` ``[2] coefficients have a wide spread of estimates, suggesting they are important predictors.
  - Coefficients such as the `` `Intercept` ``, `` `Wind:Temp` ``, `` `Solar.R:Temp` `` and `` `Solar.R:Wind` `` are tightly clustered around zero, indicating they're likely to be insignificant.
  - Coefficients like `` `I(Wind2)` ``, `` `Z2` `` and `` `Z3` `` have estimates close to zero but also show outliers, hinting at potential model sensitivity to specific data points or the presence of interaction effects not accounted for by other variables.

[2] Associated with `` `Solar.R` ``

Bootstrapping the estimating coefficients - observations

- **For `BLADlasso.shr`:**
  - The `` `Wind` ``, `` `Temp` ``, `` `I(Temp2)` ``, `` `Z1` ``[2] coefficients have a wide spread of estimates, suggesting they are important predictors.
  - Coefficients such as the `` `Intercept` ``, `` `Wind:Temp` ``, `` `Solar.R:Temp` `` and `` `Solar.R:Wind` `` are tightly clustered around zero, indicating they're likely to be insignificant.
  - Coefficients like `` `I(Wind2)` ``, `` `Z2` `` and `` `Z3` `` have estimates close to zero but also show outliers, hinting at potential model sensitivity to specific data points or the presence of interaction effects not accounted for by other variables.
- **For adaptive lasso (`glmnet; hqreg; rqpen`):**
  - Most of the predictors are very close to zero; it seems that the intercept dominates.
  - For `` `Wind` ``, there is substantial evidence to suggest that it has a non-zero coefficient.

[2] Associated with `` `Solar.R` ``

Assignment 3 Part 3 - **Monte Carlo experiment**

Implement a Monte Carlo experiment.

- Your experiment should evaluate MAE performance of your backward selection function, against the Adaptive LAD LASSO solution you prefer.
- The data generating process should be similar to Scenario 3, with t-distributed error terms ($5 < \text{df} < 10$).
- Report results from 100 replications of the experiment.

Implementation of the **Monte Carlo experiment**

- **Generating Correlated Predictors and Response Variable**

1. $$\mathbf{X} = \begin{bmatrix} 1 & \rho & \rho^2 & \cdots & \rho^{p-1} \\ \rho & 1 & \rho & \cdots & \rho^{p-2} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \rho^{p-1} & \rho^{p-2} & \rho^{p-3} & \cdots & 1 \end{bmatrix} (\texttt{toeplitz matrix}),$$

   where ρ is the correlation coefficient, and p is the number of predictors.
2. *Generation of Predictors*
   - Predictors $\mathbf{X}$ are generated as multivariate normal variables:

   $$\mathbf{X} \sim \mathcal{N}(\mathbf{0}, \Sigma),$$
3. *Generation of Response Variable*

   $$y = \mathbf{X}\beta + \epsilon,$$

   where β is the vector of true coefficients, and ϵ is the error term.The error term ϵ follows a t-distribution:

   $$\epsilon \sim t(\texttt{df}, 0, \texttt{scale}),$$

   where `df` is the degrees of freedom, and `scale` is adjusted to match a specific variance.

Implementation of the **Monte Carlo experiment**

In our experiment, we define:

$$\mathtt{scale} \leftarrow 2.2 \cdot \sqrt{(\mathtt{df} - 2)/\mathtt{df}},$$

$$y \leftarrow \mathbf{X}\beta + {}^{3}\mathtt{rt}(\mathtt{n}, \mathtt{df}) \cdot \mathtt{scale},$$

where,

$$\mathtt{n} \leftarrow 100,$$

$$\rho \leftarrow 0.9,$$

$$\mathtt{p} \leftarrow \mathtt{length}(\beta),$$

$$\beta \leftarrow \mathtt{c(2,\ -2,\ 1,\ -1,\ 0.5,\ 0.2,\ -0.3,\ -0.15,\ rep(0,\ 12))}.$$

This procedure can be repeated `R` times, typically involving more than 50 simulations, to ensure convergence.

[3] `rt` generates `n` random samples from the `t-distribution`, $5 < \mathtt{df} < 10$

```
genData.t <- function(n, p, beta, rho=0.9, df=7) {
  library(mvtnorm)
  # Create a vector x that defines the correlation structure for the predictors
  x <- c(1, rho^seq(1:(length(beta) - 1)))
  x <- toeplitz(x)

  # Generate a matrix X of n observations of p multivariate normal variables
  # with means of zero and the spec1f1ed corr. structure x
  X <- matrix(rmvnorm(n, rep(0, length(beta)), x), nrow = n, ncol = p)

  # Generate the response variable y with a t-distributed error term;
  # The scale parameter is adjusted to match the variance as 1f it was normal with sd = 2.2
  # This is because the var of a t-dist. with df degrees of freedom is df / (df-2)
  scale <- 2.2 * sqrt((df - 2) / df)
  y <- X %*% beta + rt(n, df) * scale

  return(list(X = as.data.frame(X), y = as.vector(y)))
}
```

Implementation of the Monte Carlo experiment in R

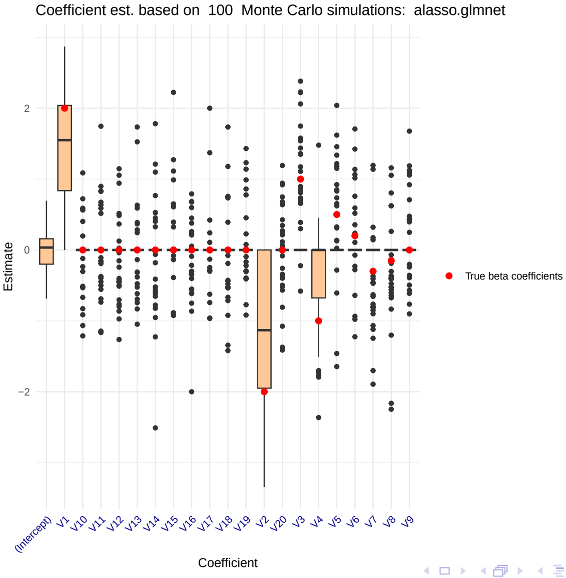
Coefficient Estimates

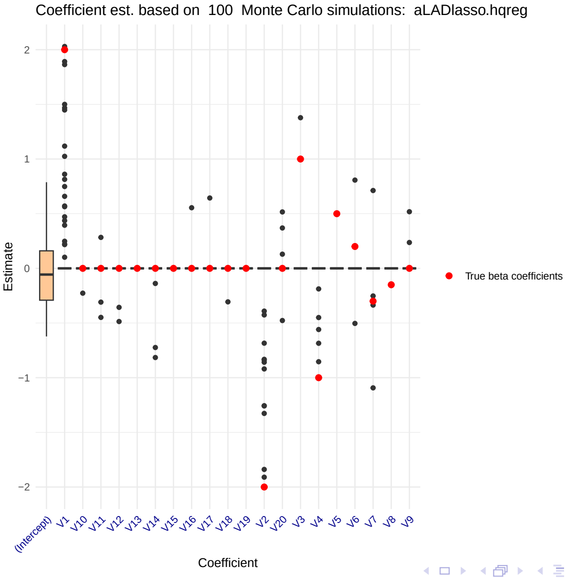
```
MC_coeffs.boxplots <- function (R, beta, method, genData=genData.t, n=100, rho=0.9) {
  require(ggplot2)
  require(dplyr)
  require(foreach)
  p <- length(beta)
  coeffs_names <- c("(Intercept)", paste0('V', seq_along(beta)))
  combined_coeffs <- data.frame(coefficient = character(),
                               value = numeric(),
                               stringsAsFactors = FALSE)
  pb <- txtProgressBar(min = 0, max = R, style = 3)
  for (r in 1:R) {
    setTxtProgressBar(pb, r)
    set.seed(r)
    # Generate data
    df <- sample(c(6,7,8,9))[1] # Choose randomly 5<df<10
    simulated.data <- genData(n=n, p=p, beta=beta, rho=rho, df=df)
    MC.X <- simulated.data$X
    MC.y <- simulated.data$y
    MC_synthetic.data <- cbind(MC.y, MC.X)
    MC.formula <- as.formula(paste('MC.y~', paste(colnames(MC.X), collapse='+')))
    # Estimate coefficients using Fit.model()
    coeffs <- Fit.model(MC.formula, MC_synthetic.data, method)
    for(i in seq_along(coeffs)) {
      combined_coeffs <- rbind(combined_coeffs,
                              data.frame(coefficient = coeffs_names[i],
                                         value = coeffs[i]))
    }
  }
  close(pb)
```

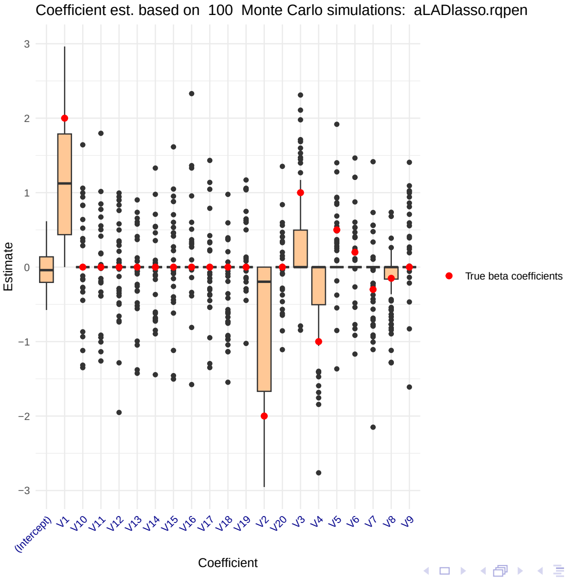
```
  # Plot the box-plots
  true_beta_df <- data.frame(coefficient = coeffs_names[-1], value = beta)

  p <- ggplot(combined_coeffs, aes(x = coefficient, y = value)) +
      geom_boxplot(fill = rgb(255, 200, 150, maxColorValue = 255)) +
      geom_point(data = true_beta_df,
                 aes(x = coefficient, y = value,
                     color = "True beta coefficients"), size = 2.2) +
      scale_color_manual(values = "red", labels = "True beta coefficients") +
      theme_minimal() +
      theme(axis.text.x = element_text(angle = 45, hjust = 1, color = "darkblue"),
            strip.text.x = element_blank()) +
      labs(x = "Coefficient", y = "Estimate",
           title = paste("Coefficient est. based on ",
                         R, " Monte Carlo simulations: ", method)) +
      guides(color = guide_legend(title = ''))

  suppressWarnings(print(p))
}

# Example
beta <- c(2, -2, 1, -1, 0.5, 0.2, -0.3, -0.15, rep(0, 12))
MC_coeffs.boxplots(R=100, beta=beta, method="alasso.glmnet")
>>
```

Implementation of the **Monte Carlo experiment** in R

Coefficient Estimates: `alasso-glmnet`

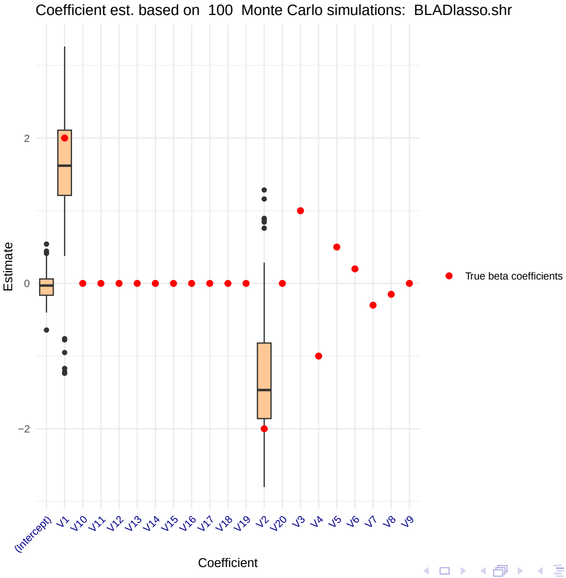



Coefficient est. based on 100 Monte Carlo simulations: aLADlasso.hqreg
Estimate
2
1
0
−1
−2
(Intercept)
V1
V10
V11
V12
V13
V14
V15
V16
V17
V18
V19
V2
V20
V3
V4
V5
V6
V7
V8
V9
Coefficient
True beta coefficients


Coefficient est. based on 100 Monte Carlo simulations: aLADlasso.rqpen
Estimate
3
2
1
0
−1
−2
−3
(Intercept) V1 V10 V11 V12 V13 V14 V15 V16 V17 V18 V19 V2 V20 V3 V4 V5 V6 V7 V8 V9
Coefficient
True beta coefficients

Implementation of the **Monte Carlo experiment** in R

Coefficient Estimates: `BAIC.LADlasso.shr`

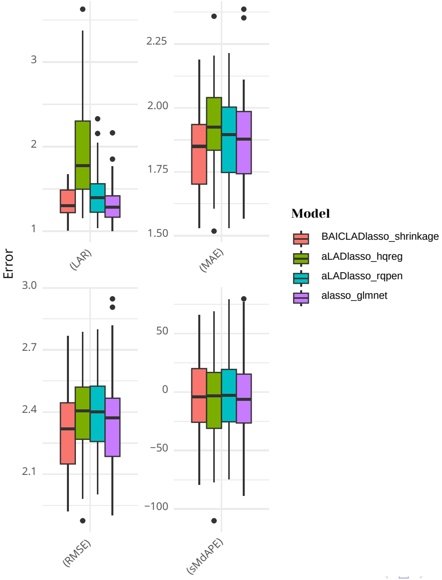


Implementation of the **Monte Carlo experiment** in R

Monte-Carlo performance based on 10-fold CV

```
MC_perf.boxplots <- function (R=100, beta, genData=genData.t, n=100,
                              rho=0.9, inv.trans = function (y) y) {
  require(ggplot2)
  require(dplyr)
  require(foreach)
  p <- length(beta)
  models_list <- list(
    aLADlasso_hqreg = list(method = "adaptive_lasso", criterion = "hqreg"),
    aLADlasso_rqpen = list(method = "adaptive_lasso", criterion = "rqpen"),
    alasso_glmnet = list(method = "adaptive_lasso", criterion = "glmnet"),
    BAICLADlasso_shrinkage = list(method = "BLADlasso.shr", criterion = AIC.Koenker))
  combined_metrics <- data.frame(metric = character(),
                                value = numeric(),
                                model = character(),
                                stringsAsFactors = FALSE)
  pb <- txtProgressBar(min = 0, max = length(models_list) * R, style = 3)
  progress <- 0

  for (model_name in names(models_list)) {
    for (r in 1:R) {
      progress <- progress + 1
      setTxtProgressBar(pb, progress)
      set.seed(r)
      # Generate data
      df <- sample(c(6,7,8,9))[1] # Choose randomly 5<df<10
      simulated.data <- genData(n=n, p=p, beta=beta, rho=rho, df=df)
      MC.X <- simulated.data$X
      MC.y <- simulated.data$y
      MC.synthetic.data <- cbind(MC.y, MC.X)
      MC.formula <- as.formula(paste('MC.y~', paste(colnames(MC.X), collapse='+')))
```

```
      cv_results <- KfoldCVPerf.general(K = 10, data = MC_synthetic.data, formula = MC.formula,
                                        method = models_list[[model_name]]$method,
                                        criterion = models_list[[model_name]]$criterion,
                                        inv.trans = inv.trans)
      # Last row of cv_results contains the mean performance metrics
      mean_metrics <- as.numeric(cv_results[nrow(cv_results), -1])
      metrics_names <- colnames(cv_results)[-1]
      for(i in seq_along(mean_metrics)) {
        combined_metrics <- rbind(combined_metrics, data.frame(
          metric = metrics_names[i],
          value = mean_metrics[i],
          model = model_name
        ))
      }
    }
  }
  close(pb)
  combined_metrics$metric <- sapply(strsplit(combined_metrics$metric, " "), function (x) x[3])
  p <- ggplot(combined_metrics, aes(x = metric, y = value, fill = model)) +
        geom_boxplot(position=position_dodge(width=0.75)) +
        facet_wrap(~ metric, scales = 'free') +
        theme_minimal() +
        theme(axis.text.x = element_text(angle = 45, hjust = 1),
              strip.text.x = element_blank()) +
        labs(x = NULL , y = "Error", fill = "Model")
  print(p)
}
beta <- c(2, -2, 1, -1, 0.5, 0.2, -0.3, -0.15, rep(0, 12))

MC_perf.boxplots(R = 50, beta=beta, genData=genData.t)
>>
```



Error
3
2
1
(LAR)
2.25
2.00
1.75
1.50
(MAE)
3.0
2.7
2.4
2.1
(RMSE)
50
0
−50
−100
(sMdAPE)
Model
BAICLADlasso_shrinkage
aLADlasso_hqreg
aLADlasso_rqpen
alasso_glmnet

Conclusions

- Based on the `Monte Carlo` results observed above, it is evident that backward LAD lasso regression using `AIC` performs the best in terms of mean absolute error (MAE). Its worth to note, however, that its performance is quite similar to that of adaptive LAD lasso with `rqpen` and adaptive regression with `glmnet`.

Conclusions

- Based on the `Monte Carlo` results observed above, it is evident that backward LAD lasso regression using `AIC` performs the best in terms of mean absolute error (MAE). Its worth to note, however, that its performance is quite similar to that of adaptive LAD lasso with `rqpen` and adaptive regression with `glmnet`.
- Although backward LAD lasso appears to be the superior choice performance-wise, it is computationally expensive. Consequently, adaptive lasso with `glmnet` emerges as an excellent alternative. Its performance closely rivals that of backward LAD lasso, yet it is significantly faster, making it a practical choice in many scenarios.

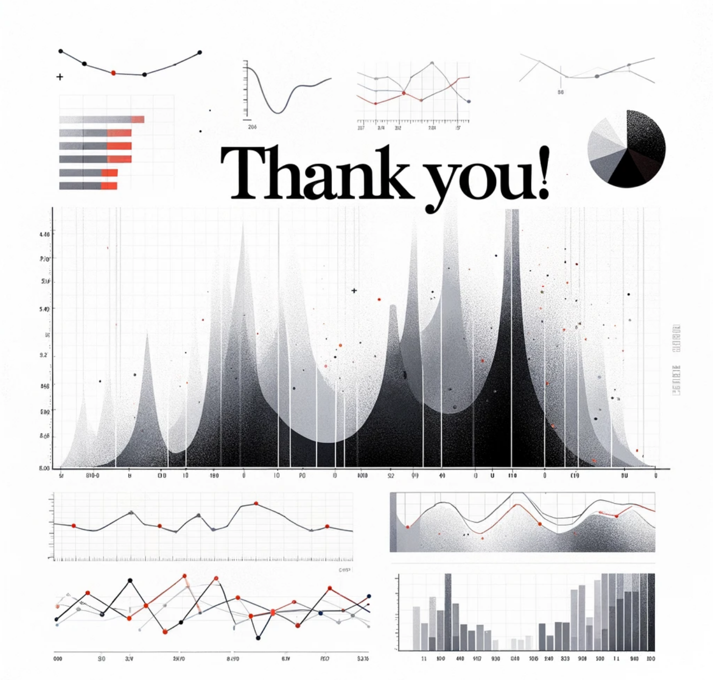
Conclusions

- Based on the `Monte Carlo` results observed above, it is evident that backward LAD lasso regression using `AIC` performs the best in terms of mean absolute error (MAE). Its worth to note, however, that its performance is quite similar to that of adaptive LAD lasso with `rqpen` and adaptive regression with `glmnet`.
- Although backward LAD lasso appears to be the superior choice performance-wise, it is computationally expensive. Consequently, adaptive lasso with `glmnet` emerges as an excellent alternative. Its performance closely rivals that of backward LAD lasso, yet it is significantly faster, making it a practical choice in many scenarios.
- In conclusion, if one prioritizes performance above all, backward LAD lasso with shrinkage towards $\hat{\beta}_0$, guided by `AIC`, is the optimal choice in terms of `MAE`. However, for scenarios demanding quicker, possibly real-time results, the adaptive lasso with `glmnet` stands out as a highly effective and more efficient alternative.

References

Roger Koenker (2005)
Quantile Regression
Cambridge University Press

Chris Tofallis
A Better Measure of Relative Prediction Accuracy for Model Selection and Model Estimation
Journal of the Operational Research Society (2015) 66, 1352–1362, Available at:
https://papers.ssrn.com/sol3/papers.cfm?abstract_id=2635088


Thank you!