Autotune Lasso

The R package `Autotune` implements the Lasso with data-driven tuning for linear models.

Installation and Loading

Installation

The development version of the `Autotune` package can be installed from GitHub using

```
# install.packages("devtools")
# Ensure that you have the Rcpp package installed with version >=1.0.13
# devtools::install_github("Tathagata-S/Autotune")
```

When installing from GitHub, in order to build the package from source, you need to have the appropriate R development tools installed (Rtools on Windows, or these tools on Mac).

Load Package

After installation, the package can be loaded in the standard way:

```
library(Autotune)
```

High Dimensional Regression

`Autotune` performs lasso via the `autotune_lasso()` function.

We illustrate autotune lasso on simulated data using a linear model with $s = 10$, $n = 300$, $p = 500$. Reader can also specify different configurations of data generating process in the following code chunk.

```
set.seed(10)
n = 300
p = 500
s = 10
beta = c(rep(1, s), rep(0, p - s))
x = matrix(rnorm(n * p), ncol = p)
# Maunal sigma allocation
# y = x %*% beta + rnorm(n, sd = 1)
# Dynamic sigma allocation with snr specified
snr = 2
y = x %*% beta + rnorm(n, sd = sqrt(var(x%*%beta)/snr))
```

Runtime of Autotune Lasso

Given data (x, y), run the autotune lasso as follows with default $\alpha = 0.01$.

```
ptm <- proc.time()
fit.autotune <- autotune_lasso(x, y, alpha = 0.01)
proc.time() - ptm
#>    user  system elapsed
#>   0.014   0.001   0.017
```

The regression coefficients βs, intercept, final lambda and the sequence of estimated sigmas can be extracted from the fitted autotune lasso object as follows

```
b.autotune <- fit.autotune$beta
intercept.autotune <- fit.autotune$a0
lambda.autotune <- fit.autotune$lambda
sigma.seq.autotune <- fit.autotune$CD.path.details$sigma_sq_seq
sigma.estimate.autotune <- fit.autotune$sigma_sq
```

Comparison with Lasso tuned via Cross Validation (CV Lasso)

We contrast our solution with Cross-Validation-tuned Lasso using `cv.glmnet()` in the `glmnet` package.

```
library(glmnet)
#> Warning: package 'glmnet' was built under R version 4.3.3
#> Loading required package: Matrix
#> Loaded glmnet 4.1-9
```

Runtime of CV Lasso

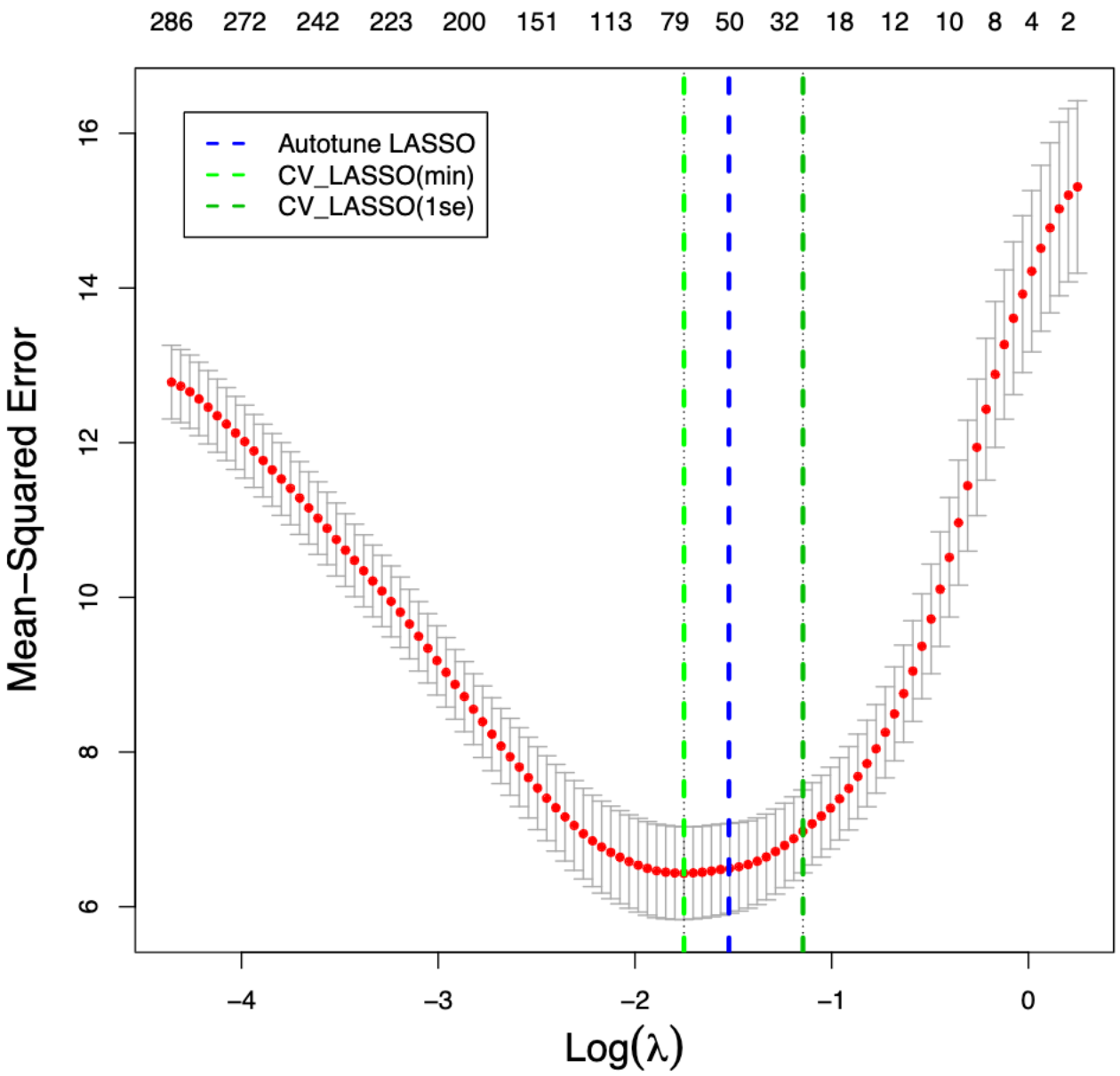
```
ptm2 <- proc.time()
fit.glmnet <- cv.glmnet(x, y)
proc.time()-ptm2
#>    user  system elapsed
#>   0.282   0.011   0.292
```

So, Autotune Lasso shows faster runtimes as compared to CV Lasso.

Comparison of quality of tuning between Autotune and CV Lasso

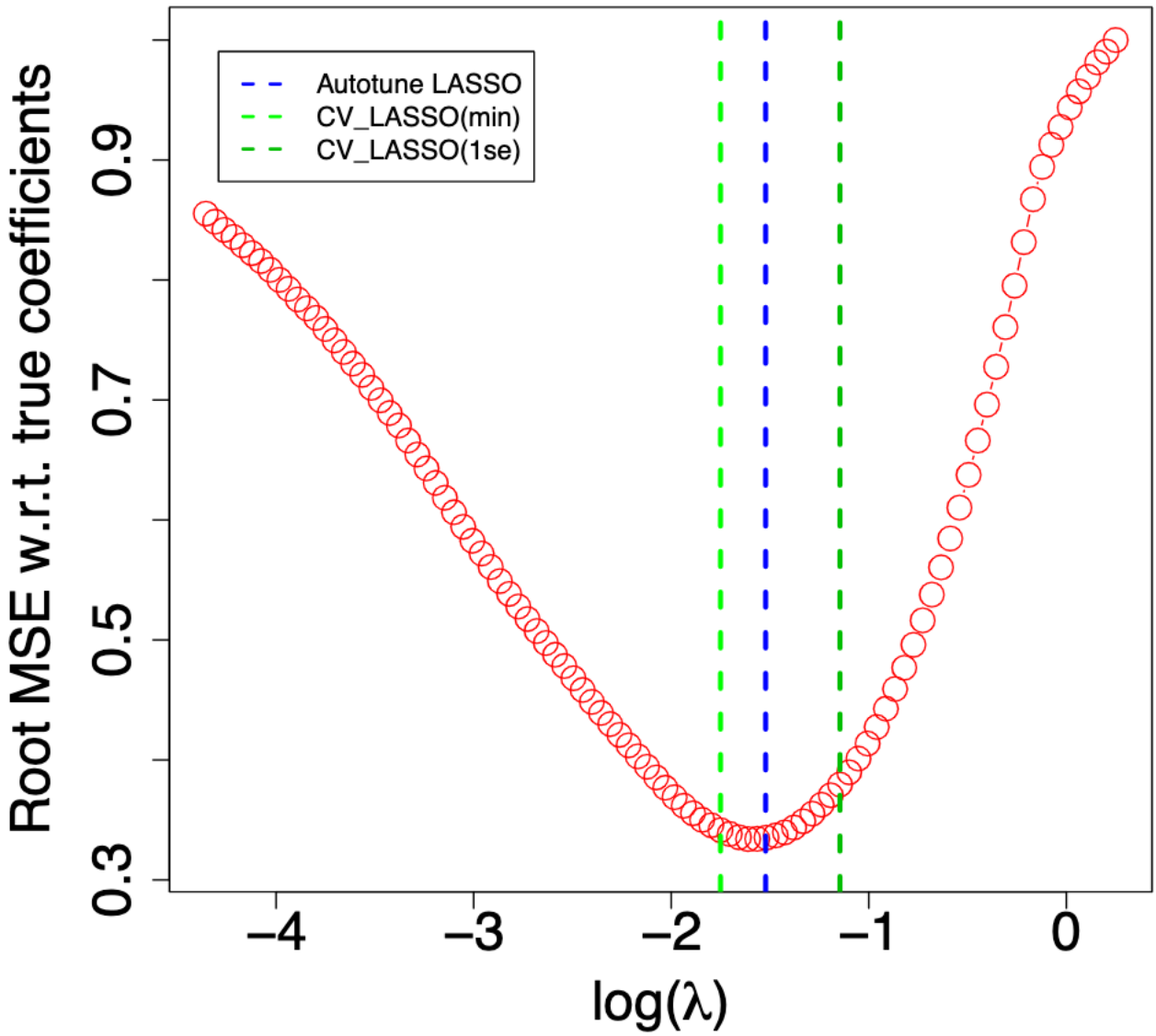
Now, we will visualize the quality of lambdas selected by autotune and CV with respect to the 10-fold CV MSE and true Relative MSE.

```
plot(fit.glmnet, cex.lab = 1.5)

# plotting log of lambdas selected by CV
abline(v = log(c(fit.glmnet$lambda.min, fit.glmnet$lambda.1se)), lty = "dashed",col = c(rgb(0,1,0), rgb

# plotting log of lambda selected by autotune
abline(v = log(lambda.autotune), col = "blue", lty = "dashed", lwd = 3)
legend("topleft", inset = 0.05,             legend = c("Autotune LASSO",
             "CV_LASSO(min)",
             "CV_LASSO(1se)"),
  col = c("blue", rgb(0,1,0), rgb(0,0.75,0)),
  lty = "dashed", lwd = 2, cex = 1)
```




```
mse_glmnet <- apply(fit.glmnet$glmnet.fit$beta, 2, function(x) sqrt(mean((x- beta)^2))/sqrt(mean(beta^2)
ymin = 0.95 * min(mse_glmnet)
ymax = max(1, max(mse_glmnet))
par(mgp = c(3.5, 1, 0), mar = c(5, 4, 2.5, 0.5) + 1.4)
plot(log(fit.glmnet$lambda), mse_glmnet,
     type = 'b', col = "red", cex.lab = 2, cex=2, cex.axis = 2,
     ylim = c(ymin, ymax),
     ylab = "Root MSE w.r.t. true coefficients", xlab = expression(paste("log(",lambda,")")))

# plotting log of lambdas selected by CV
abline(v = log(c(fit.glmnet$lambda.min, fit.glmnet$lambda.1se)), lty = "dashed",col = c(rgb(0,1,0), rgb

# plotting log of lambdas selected by autotune
abline(v = log(lambda.autotune), col = "blue", lty = "dashed", lwd = 3)
legend(
  "topleft", inset = 0.05,
```

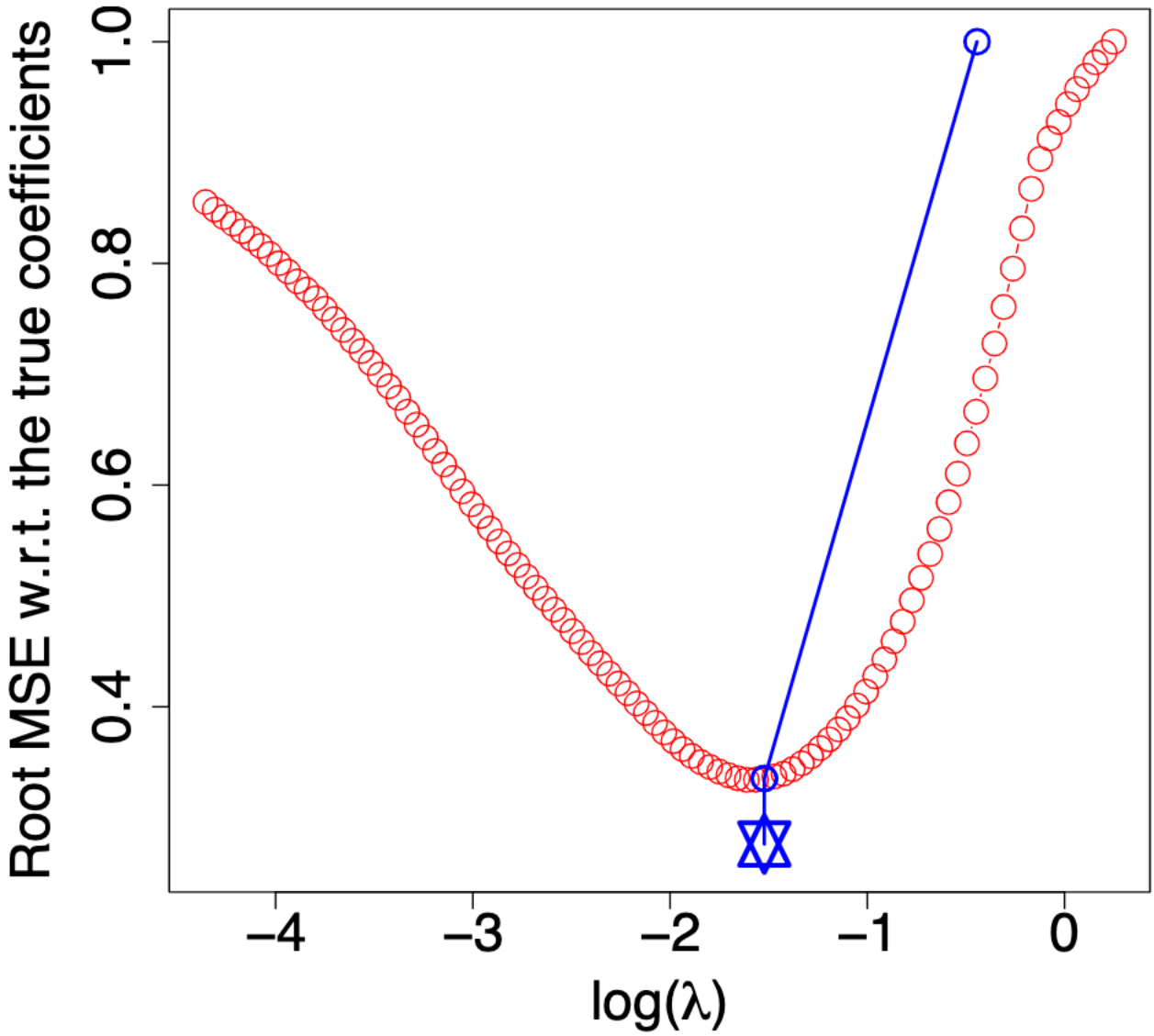
```
  legend = c("Autotune LASSO",
             "CV_LASSO(min)",
             "CV_LASSO(1se)"),
  col = c("blue", rgb(0,1,0), rgb(0,0.75,0)),
  lty = "dashed", lwd = 2, cex = 1)
```



Across the lambda grid, we plot RMSE of solution path taken by different tuners.

```
temp <- fit.autotune$CD.path.details$lambda0
seq.lambdas.autotune <- temp * c(var(y), sigma.seq.autotune)

final.rmse.autotune <- sqrt(mean(b.autotune - beta)^2)/sqrt(mean(beta)^2)
intermediate.rmse.autotune <- approx(x = fit.glmnet$lambda, y = mse_glmnet, xout = seq.lambdas.autotune
intermediate.rmse.autotune$y[1] <- 1
rmse.path.autotune <- c(intermediate.rmse.autotune$y, final.rmse.autotune)

par(mgp = c(3.5, 1, 0), mar = c(5, 4, 2.5, 0.5) + 1.4)
plot(log(fit.glmnet$lambda), mse_glmnet,
```

```
     type = 'b', col = "red", cex.lab = 2, cex=2, cex.axis = 2,
     ylim = c(min(ymin, 0.95 * final.rmse.autotune), ymax),
     ylab = "Root MSE w.r.t. the true coefficients", xlab = expression(paste("log(",lambda,")")))

lines(c(log(seq.lambdas.autotune), log(fit.autotune$lambda)),
      rmse.path.autotune,
      col = "blue",
      lwd = 2)
points(log(seq.lambdas.autotune),
       intermediate.rmse.autotune$y,
       col = "blue",
       pch = 1,
       lwd = 2,
       cex = 2)
points(log(fit.autotune$lambda),
      final.rmse.autotune,
      col = "blue",
      pch = 11,
      lwd = 3,
      cex = 3
    )
```




```
library(AUC)
#> AUC 0.3.2
#> Type AUCNews() to see the change log and ?AUC to get an overview.
```

```
b.glmnet = coef(fit.glmnet, s = "lambda.min")[-1]
```

```
auc(roc(abs(b.autotune), as.factor(beta != 0)))
#> [1] 1
auc(roc(abs(b.glmnet), as.factor(beta != 0)))
#> [1] 1
```

Comparison on real data provided in Scaled Lasso's R package `scalreg`

```
library(scalreg)
#> Loading required package: lars
#> Loaded lars 1.3
```

Comparing prediction errors of autotune and benchmarks: CV and Scaled Lasso

```
data("sp500")
attach(sp500)

n <- 200
rang <- 252 - n

X_train = sp500.percent[1:n, 3: (dim(sp500.percent)[2])]
Y_train = sp500.percent[1:n, 1]

x_train = scale(X_train)
y_train = Y_train - mean(Y_train)

X_test = sp500.percent[(1:rang) + n, 3: (dim(sp500.percent)[2])]
Y_test = sp500.percent[(1:rang) + n, 1]
x_test <- scale(X_test)
y_test <- Y_test - mean(Y_test)

# Default value of beta_iter_max is 40
ans_autotune <- autotune_lasso(x_train, y_train, beta_iter_max = 40, trace_it = TRUE)
#> Iteration: 1Iteration: 2Iteration: 3Iteration: 4Lambda converged, Iteration: 5Lambda converged, Iter
#>
#> No of predictors significant for sigma estimation: 3
pred_err_autotune <- mean( (y_test - x_test %*% ans_autotune$beta)^2 )

object = scalreg(x_train, y_train)
pred_err_scallas <- mean( (y_test - x_test %*% object$coefficients)^2 )

cv_fit <- cv.glmnet(x_train, y_train, alpha = 1, intercept = F)
pred_err_cv_min <- mean( (y_test - predict(cv_fit, newx = x_test, s = "lambda.min"))^2 )
pred_err_cv_1se <- mean( (y_test - predict(cv_fit, newx = x_test, s = "lambda.1se"))^2 )


pred_err_autotune
#> [1] 0.5145037
pred_err_scallas
#> [1] 0.6826112
pred_err_cv_min
#> [1] 0.6714067
pred_err_cv_1se
#> [1] 0.6450888

sum(object$coefficients != 0)
#> [1] 64
sum(ans_autotune$beta != 0)
#> [1] 41
sum(coef(cv_fit, s = "lambda.min") != 0)
#> [1] 61
sum(coef(cv_fit, s = "lambda.1se") != 0)
#> [1] 60
detach(sp500)
```

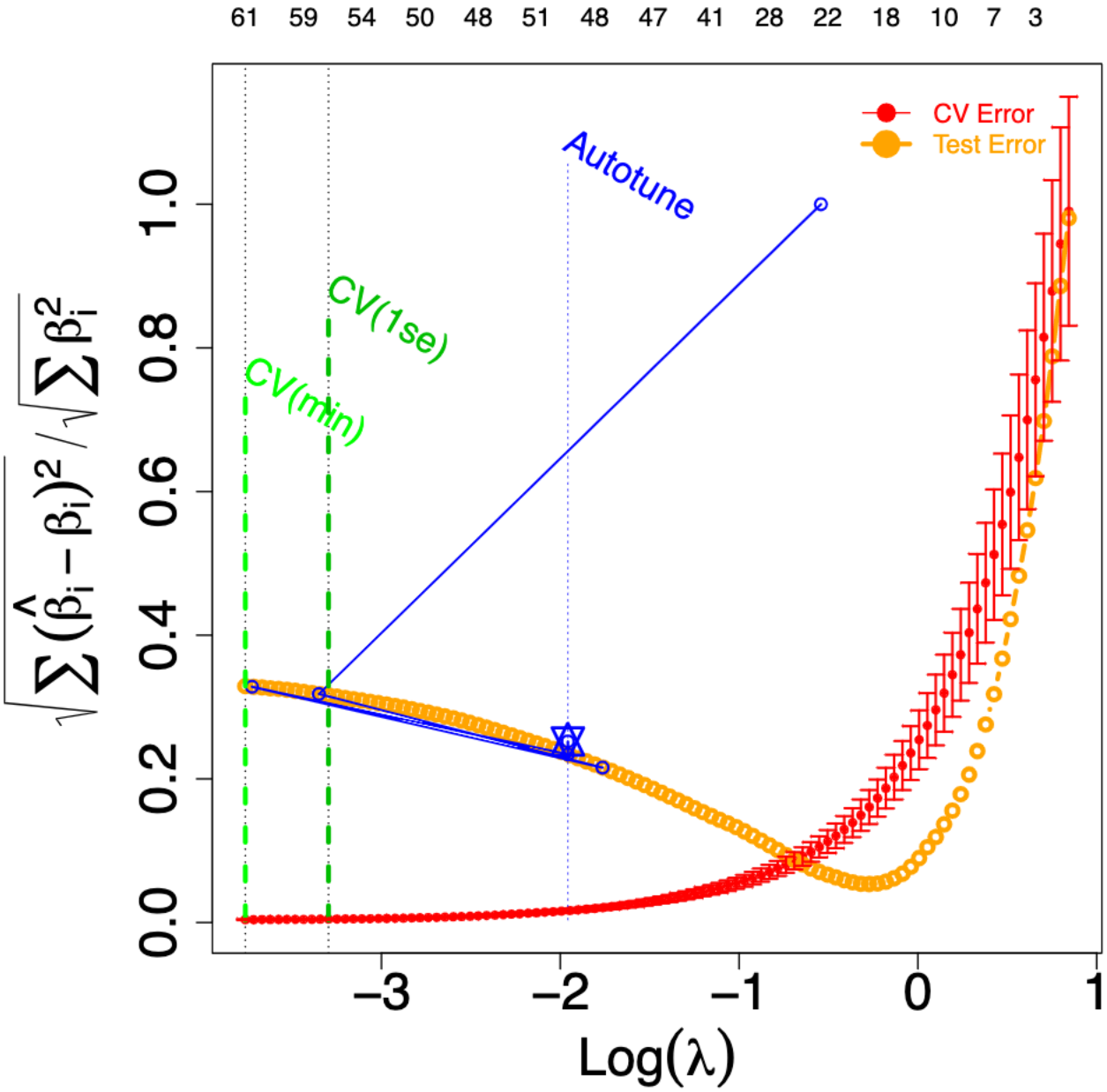
Comparing via bootstrap

```
real_life_prediction_comparison <- function(x, y, x_test, y_test, alpha = 0.01, beta_iter_max = 40, plo

ans_autotune <- autotune_lasso(x, y, alpha = alpha, beta_iter_max = beta_iter_max, trace_it = trace_it)
# ans_glmnet <- glmnet(x, y, alpha = 1, intercept = FALSE)
cv_fit <- cv.glmnet(x, y, alpha = 1, intercept = F)
vary <- var(y_test)
mse_glmnet <- apply(cv_fit$glmnet.fit$beta, 2, function(b) mean((y_test - x_test%*%b)^2)/vary)

object = scalreg(x, y)


# log_glmnet_lambdas <- log(fit.glmnet$lambda)
# log_autotune_lambdas <- log(intermediate_lambdas_used)



if(plot_regu) {
  intermediate_lambdas_used <- c( c(var(y)/2, ans_autotune$CD.path.details$sigma_sq_seq) * ans_autotune

  autotune_lambdas <- c(intermediate_lambdas_used, ans_autotune$lambda)

  final.rmse.autotune <- mean((y_test - x_test%*%ans_autotune$beta)^2) / vary
  intermediate.rmse.autotune <- approx(x = log(cv_fit$lambda), y = mse_glmnet, xout = log(intermediate_
  intermediate.rmse.autotune$y[1] <- 1
  rmse.path.autotune <- c(intermediate.rmse.autotune$y, final.rmse.autotune)


  ysquaredmean <- mean(y^2)
  cv_fit$cvm <- cv_fit$cvm/ysquaredmean
  cv_fit$cvsd <- cv_fit$cvsd/ysquaredmean
  cv_fit$cvup <- cv_fit$cvm + cv_fit$cvsd
  cv_fit$cvlo <- cv_fit$cvm - cv_fit$cvsd

  cv_fit_lowerlim <- min(cv_fit$cvm - cv_fit$cvsd, mse_glmnet)
  cv_fit_upperlim <- max(cv_fit$cvm + cv_fit$cvsd, mse_glmnet)

  ymin = 0.95 * min(cv_fit_lowerlim , min(rmse.path.autotune))
  ymax = max(cv_fit_upperlim, rmse.path.autotune)
  ymax_trunc = ymin + 0.8 * (ymax - ymin)  # Truncate vertical lines here

  ydiff = ymax - ymin
  par(mgp = c(3.5, 1, 0), mar = c(6, 5, 4, 2) + 0.1)

  par(mar = c(5, 7, 4, 2))
  plot(cv_fit,
       ylim = c(ymin, max(cv_fit_upperlim, rmse.path.autotune)),
       cex.lab = 2,
       cex.axis = 2,
       ylab = expression( sqrt( sum( (hat(beta)[i] - beta[i])^2 ) ) /
                    sqrt( sum( beta[i]^2 ) ) ))
  # expression(paste(sqrt("RMSE")))
```

```r
points(log(cv_fit$lambda), mse_glmnet,
       type = 'b', col = "orange", lwd = 3)


lines(rep(log(ans_autotune$lambda), 2),
      c(ymin, ymin + 0.92 * ydiff), col = "blue", lwd = 0.5, lty = "dashed")

lines(log(autotune_lambdas), rmse.path.autotune,
      col = "blue", lwd = 1.5)

points(log(autotune_lambdas), rmse.path.autotune,
       col = "blue", pch = 1, lwd = 1.5, cex = 1)

points(log(ans_autotune$lambda), final.rmse.autotune,
       col = "blue", pch = 11, lwd = 2, cex = 2)



vlines <- c(
  log(cv_fit$lambda.min),
  log(cv_fit$lambda.1se)
)
vline_labels <- c("CV(min)", "CV(1se)")
vline_colors <- c(rgb(0,1,0), rgb(0,0.75,0))
vline_ltys <- c("dashed", "dashed")
vline_lwds <- c(3, 3)

for (i in seq_along(vlines)) {
  lines(rep(vlines[i], 2),
        c(ymin, ymin + (0.55 + i * 0.1) * ydiff),
        col = vline_colors[i], lty = vline_ltys[i],
        lwd = vline_lwds[i])

  text(x = vlines[i], y = ymin + (0.57 + i * 0.1) * ydiff,
       labels = vline_labels[i], srt = -30, adj = 0,
       xpd = TRUE, cex = 1.4, col = vline_colors[i])
}

text(x = log(ans_autotune$lambda),
     y =  ymin + 0.95 * ydiff,
     labels = "Autotune", srt = -30, adj = 0,
     xpd = TRUE, cex = 1.4, col = "blue")

if (!is.null(cv_fit$cvsd)) {
  arrows(x0 = log(cv_fit$lambda),
         y0 = cv_fit$cvm - cv_fit$cvsd,
         x1 = log(cv_fit$lambda),
         y1 = cv_fit$cvm + cv_fit$cvsd,
         angle = 90, code = 3, length = 0.05, col = "red", lwd = 1.5)
}


legend("topright",
```

```
          inset = c(0.05, 0.02),
          legend = c("CV Error", "Test Error"),
          col = c("red", "orange"),
          pt.cex = c(1.5, 2), pch = c(19, 19),
          lwd = c(1, 3),
          text.col = c("red", "orange"),
          horiz = FALSE,
          bty = "n")

}

# mse_scallas <- mean( (y_test - x_test %*% object$coefficients)^2 )
# mse_autotune <- mean( (y_test - x_test %*% ans_autotune$beta)^2 )
# mse_cv_min <- mean( (y_test - predict(cv_fit, newx = x_test, s = "lambda.min"))^2 )
# mse_cv_1se <- mean( (y_test - predict(cv_fit, newx = x_test, s = "lambda.1se"))^2 )


oos_scallas <- 1 - mean( (y_test - x_test %*% object$coefficients)^2 ) / vary
oos_autotune <- 1 -  mean( (y_test - x_test %*% ans_autotune$beta)^2 ) / vary
oos_cv_min <- 1 - mean( (y_test - predict(cv_fit, newx = x_test, s = "lambda.min"))^2 ) / vary
oos_cv_1se <- 1 - mean( (y_test - predict(cv_fit, newx = x_test, s = "lambda.1se"))^2 ) / vary

return(data.frame(out_of_sample_autotune = oos_autotune,
out_of_sample_scaled_lasso = oos_scallas,
out_of_sample_cv_min = oos_cv_min,
out_of_sample_cv_1se = oos_cv_1se,
nonzero_coefs_autotune = sum(ans_autotune$beta != 0),
nonzero_coefs_scaled_lasso = sum(object$coefficients != 0),
nonzero_coefs_cvmin = sum(coef(cv_fit, s = "lambda.min") != 0),
nonzero_coefs_cv1se = sum(coef(cv_fit, s = "lambda.1se") != 0)))
}
```

```
ans <- real_life_prediction_comparison(x_train, y_train, x_test, y_test, plot_regu = TRUE)
#> Iteration: 1Iteration: 2Iteration: 3Iteration: 4Lambda converged, Iteration: 5Lambda converged, Iter
#>
#> No of predictors significant for sigma estimation: 3
```

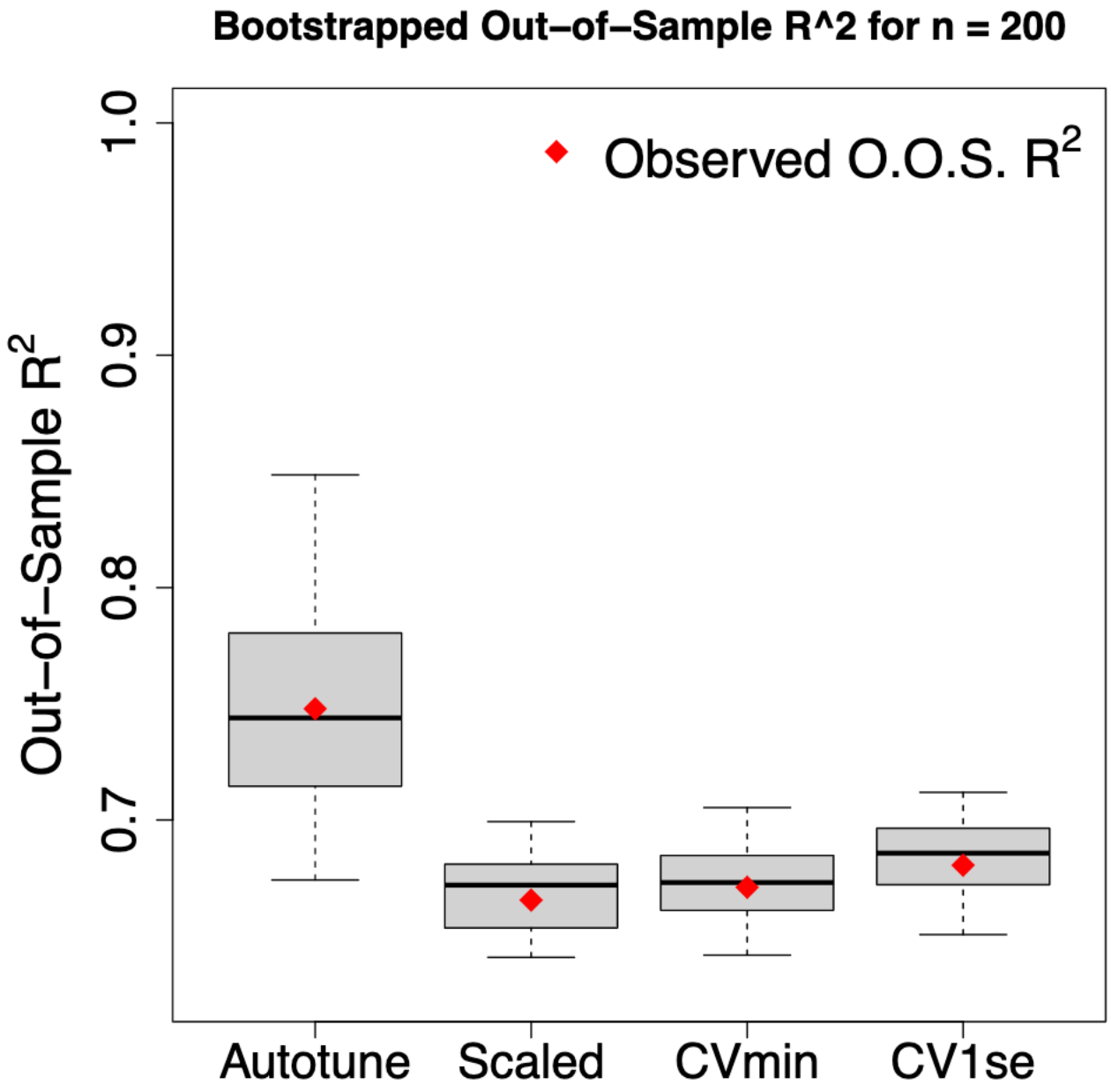



```
names(ans)
#> [1] "out_of_sample_autotune"     "out_of_sample_scaled_lasso"
#> [3] "out_of_sample_cv_min"       "out_of_sample_cv_1se"
#> [5] "nonzero_coefs_autotune"     "nonzero_coefs_scaled_lasso"
#> [7] "nonzero_coefs_cvmin"        "nonzero_coefs_cv1se"
```

```
library(dplyr)
#>
#> Attaching package: 'dplyr'
#> The following objects are masked from 'package:stats':
#>
#>     filter, lag
#> The following objects are masked from 'package:base':
#>
#>     intersect, setdiff, setequal, union
```

```r
B <- 20
result_list <- list()

result_list[[1]] <- data.frame(n = n, bootstrap = FALSE, real_life_prediction_comparison(x_train, y_tra

for( j in (1:B)+1) {
  set.seed(j)
  boot.sample <- sample(n, size = n, replace = TRUE)
  boot.x <- x_train[boot.sample, ]
  boot.y <- y_train[boot.sample]
  result_list[[j]] <- c(n = n, bootstrap = TRUE, real_life_prediction_comparison(boot.x, boot.y, x_test
}

mat <- do.call(rbind, result_list)
df <- as.data.frame(mat, stringsAsFactors = FALSE)

colnames(df) <- c("n", "bootstrap", "Autotune", "Scaled", "CVmin",
                  "CV1se")
df <- df %>%
  mutate(
    n = as.integer(n),
    bootstrap = as.logical(bootstrap),
    across(!c(n, bootstrap), as.numeric)
  )
```

```r
alg_cols <- colnames(df)[3:6]
boot_only <- df %>% filter(bootstrap == TRUE)
boxlist <- lapply(alg_cols, function(cn) boot_only[[cn]])
names(boxlist) <- alg_cols

# Uncomment the following line and line 448 for saving the boxplot in your working directory

# pdf(file.path(paste0("OOS_boxplot_for_n=",df[1,1],"_B=", B,".pdf")), width = 8, height = 9)

par(mar = c(4, 6, 4, 2))
boxplot(boxlist, main = paste0("Bootstrapped Out-of-Sample R^2 for n = ", df[1,1]), ylab = expression(p
        cex.lab = 2,
  cex.axis = 1.8,
  cex.main = 1.5)

nonboot <- df %>% filter(bootstrap == FALSE)
if(nrow(nonboot) > 0){
  points(1:4, as.numeric(nonboot[1, alg_cols]), pch = 18, col = "red", cex = 2)
}

legend(
  "topright",
  legend = expression(paste("Observed O.O.S. ",R^2)),
  col = "red",
  pch = 18,
  pt.cex = 2,
  cex = 2,
  bty = "n"
)
```


Bootstrapped Out−of−Sample R^2 for n = 200
Observed O.O.S. R2
Out−of−Sample R2
1.0
0.9
0.8
0.7
Autotune
Scaled
CVmin
CV1se

```
# dev.off()
```