Exercise 5 (2025) — Advanced Methods for Regression and Classification

Olesia Galynskaia 12321492

2025-11-12

Loading and observing data

```
# Load required package
suppressPackageStartupMessages({
  library(robustHD)   # contains nci60 dataset
})

# 1) Load the dataset
data("nci60", package = "robustHD")

# After loading, two objects should appear: gene and protein
objs <- ls()
cat("Objects in environment:", paste(objs, collapse = ", "), "\n")
```

```
## Objects in environment: cellLineInfo, gene, geneInfo, protein, proteinInfo
```

```
# 3) Show structure of the main components
#    (gene and protein should appear after loading)
cat("Structure of gene:\n")
```

```
## Structure of gene:
```

```
str(gene)
```

```
##  num [1:59, 1:22283] 9.52 6.5 6.33 5.98 10.36 ...
##  - attr(*, "dimnames")=List of 2
##   ..$ : chr [1:59] "1" "2" "3" "4" ...
##   ..$ : chr [1:22283] "1" "2" "3" "4" ...
```

```r
cat("\nStructure of protein:\n")
```

```
##
## Structure of protein:
```

```r
str(protein)
```

```
##  num [1:59, 1:162] -1.67 -2.2 -0.58 -1.56 0.01 -1.95 -0.36 -1.53 -1 0.17 ...
##  - attr(*, "dimnames")=List of 2
##   ..$ : chr [1:59] "1" "2" "3" "4" ...
##   ..$ : NULL
```

```r
# 4) Open help page for the dataset (shows description, variables, and context)
#    Note: In knit or markdown, this will print the path rather than open a help tab.
#cat("\nHelp file:\n")
#help("nci60", package = "robustHD")

# 5) Dimensions summary
cat("\nDimensions:\n")
```

```
##
## Dimensions:
```

```r
cat("gene    ->", paste(dim(gene), collapse = " x "), "\n")
```

```
## gene    -> 59 x 22283
```

```r
cat("protein ->", paste(dim(protein), collapse = " x "), "\n")
```

```
## protein -> 59 x 162
```

```r
cat("=== Dimensions ===\n")
```

```
## === Dimensions ===
```

```r
cat("gene        :", paste(dim(gene), collapse = " x "), "\n")
```

```
## gene        : 59 x 22283
```

```r
cat("protein     :", paste(dim(protein), collapse = " x "), "\n")
```

```
## protein     : 59 x 162
```

```r
cat("geneInfo    :", paste(dim(geneInfo), collapse = " x "), "\n")
```

```
## geneInfo    : 22283 x 4
```

```r
cat("proteinInfo :", paste(dim(proteinInfo), collapse = " x "), "\n")
```

```
## proteinInfo : 162 x 4
```

```r
cat("cellLineInfo:", paste(dim(cellLineInfo), collapse = " x "), "\n\n")
```

```
## cellLineInfo: 59 x 15
```

```r
# Sanity: 59 observations everywhere
stopifnot(nrow(gene) == 59, nrow(protein) == 59, nrow(cellLineInfo) == 59)

# Check alignment of samples across matrices and cellLineInfo
cat("=== Sample alignment checks ===\n")
```

```
## === Sample alignment checks ===
```

```r
same_samples_gp <- identical(rownames(gene), rownames(protein))
same_samples_ci <- identical(rownames(gene), rownames(cellLineInfo))
cat("gene vs protein rownames identical?  ->", same_samples_gp, "\n")
```

```
## gene vs protein rownames identical?  -> TRUE
```

```r
cat("gene vs cellLineInfo rownames match? ->", same_samples_ci, "\n\n")
```

```
## gene vs cellLineInfo rownames match? -> TRUE
```

```r
# Check alignment of feature metadata:
# For protein: columns of 'protein' should correspond to rows of proteinInfo.
# Often colnames(protein) equals proteinInfo$Probe.
cat("=== Protein feature alignment ===\n")
```

```
## === Protein feature alignment ===
```

```r
prot_by_probe <- !is.null(colnames(protein)) &&
                 all(colnames(protein) %in% proteinInfo$Probe)
cat("protein colnames in proteinInfo$Probe? ->", prot_by_probe, "\n")
```

```
## protein colnames in proteinInfo$Probe? -> FALSE
```

```r
if (prot_by_probe) {
  # Reorder info to match matrix columns, for later joins/plots
  proteinInfo <- proteinInfo[match(colnames(protein), proteinInfo$Probe), , drop = FALSE]
}

# For gene: columns of 'gene' should correspond to rows of geneInfo.
cat("=== Gene feature alignment ===\n")
```

```
## === Gene feature alignment ===
```

```r
gene_by_probe <- !is.null(colnames(gene)) &&
                 all(colnames(gene) %in% geneInfo$Probe)
cat("gene colnames in geneInfo$Probe? ->", gene_by_probe, "\n")
```

```
## gene colnames in geneInfo$Probe? -> FALSE
```

```r
if (gene_by_probe) {
  geneInfo <- geneInfo[match(colnames(gene), geneInfo$Probe), , drop = FALSE]
}
cat("\n")
```

```r
# Missing values overview (both matrices are preprocessed, but let's verify)
cat("=== Missing values ===\n")
```

```
## === Missing values ===
```

```r
cat("Total NA in gene   :", sum(is.na(gene)), "\n")
```

```
## Total NA in gene   : 0
```

```r
cat("Total NA in protein:", sum(is.na(protein)), "\n\n")
```

```
## Total NA in protein: 0
```

```r
# Quick variance snapshot to spot useless features (we WON'T filter yet)
cat("=== Zero-variance features (quick count) ===\n")
```

```
## === Zero-variance features (quick count) ===
```

```r
gene_var <- apply(gene, 2, var, na.rm = TRUE)
prot_var <- apply(protein, 2, var, na.rm = TRUE)
cat("Genes with zero variance   :", sum(!is.finite(gene_var) | gene_var == 0), "\n")
```

```
## Genes with zero variance   : 0
```

```r
cat("Proteins with zero variance:", sum(!is.finite(prot_var) | prot_var == 0), "\n\n")
```

```
## Proteins with zero variance: 0
```

```r
# Preview of cell line meta to understand strata (tissues, histology)
cat("=== Cell line meta preview ===\n")
```

```
## === Cell line meta preview ===
```

```r
print(head(cellLineInfo[, c(names(cellLineInfo)[1:min(6, ncol(cellLineInfo))]) ], 6))
```

```
##            Name Tissue Age Sex PriorTreatment Epithelial
## 1     BR:BT_549 Breast  72   F             NA        yes
## 2     BR:HS578T Breast  74   F             NA        yes
## 3       BR:MCF7 Breast  69   F             NA        yes
## 4 BR:MDA_MB_231 Breast  51   F             NA        yes
## 5       BR:T47D Breast  54   F             NA        yes
## 6    CNS:SF_268    CNS  24   F            Rad         no
```

```r
# Simple frequency of a likely useful grouping variable if present
freq_col <- c("Tissue.of.origin", "Tissue", "Histology", "Line")  # common guesses
freq_col <- freq_col[freq_col %in% names(cellLineInfo)]
if (length(freq_col)) {
  for (cn in freq_col) {
    cat("\nTop categories in", cn, ":\n")
    print(sort(table(cellLineInfo[[cn]]), decreasing = TRUE)[1:10])
  }
}
```

```
##
## Top categories in Tissue :
##
##            Melanoma Non-Small Cell Lung              Renal              Colon
##                   9                   9                  8                  7
##             Ovarian                 CNS           Leukemia             Breast
##                   7                   6                  6                  5
##            Prostate                <NA>
##                   2
##
## Top categories in Histology :
##
##                                                  Adenocarcinoma-md
##                                                                  4
##                                                      Adenocarcinoma
```

```
##                                                                                 3
##                                                                  Glioblastoma, ud
##                                                                                 3
##                                                  Malignant melanotic melanoma
##                                                                                 3
##                                                          Melanotic melanoma
##                                                                                 3
##                                                          Adenocarcinoma-vpd
##                                                                                 2
##                                                               Carcinoma-ud
##                                                                                 2
## Ductal carcinoma- mammary gland; breast; duct; metastatic site: pleural effusion;
##                                                                                 2
##         Adenocarcinoma- mammary gland; breast; metastatic site: pleural effusion;
##                                                                                 1
##                                    Adenocarcinoma- prostate; metastatic site: bone;
##                                                                                 1
```

```r
# Protein symbols: duplicates are expected (multiple probes per symbol).
cat("\n=== Protein symbols: duplicated? ===\n")
```

```
##
## === Protein symbols: duplicated? ===
```

```r
if ("Symbol" %in% names(proteinInfo)) {
  dup_sym <- sort(table(proteinInfo$Symbol), decreasing = TRUE)
  cat("Number of unique symbols:", length(dup_sym), "out of", nrow(proteinInfo), "probes\n")
  cat("Most frequent symbols (top 10):\n")
  print(head(dup_sym, 10))
}
```

```
## Number of unique symbols: 94 out of 162 probes
## Most frequent symbols (top 10):
##
## EP300  TP53  RELA CASP7  CDH1  CDH2  FADD  JAK1 STAT1 STAT3
##    31     8     6     4     3     3     3     3     3     3
```

```r
# Gene symbols: same story; many NA or multiple probes per gene.
cat("\n=== Gene symbols: duplicated? ===\n")
```

```
##
## === Gene symbols: duplicated? ===
```

```r
if ("Symbol" %in% names(geneInfo)) {
  dup_gsym <- sort(table(geneInfo$Symbol), decreasing = TRUE)
  cat("Number of unique symbols:", length(dup_gsym), "out of", nrow(geneInfo), "probes\n")
  cat("Most frequent gene symbols (top 10):\n")
  print(head(dup_gsym, 10))
}
```

```
## Number of unique symbols: 12980 out of 22283 probes
## Most frequent gene symbols (top 10):
##
##  HLA-C    HFE   IGL@   TRA@  CFLAR  RUNX1  FGFR2 PTGER3   TCF3    IDS
##     14     13     13     12     11     11     10     10     10      9
```

```r
# Missing values overview (both matrices are preprocessed, but let's verify)
cat("=== Missing values ===\n")
```

```
## === Missing values ===
```

```r
cat("Total NA in gene   :", sum(is.na(gene)), "\n")
```

```
## Total NA in gene   : 0
```

```r
cat("Total NA in protein:", sum(is.na(protein)), "\n\n")
```

```
## Total NA in protein: 0
```

```r
# Quick variance snapshot to spot useless features (we WON'T filter yet)
cat("=== Zero-variance features (quick count) ===\n")
```

```
## === Zero-variance features (quick count) ===
```

```r
gene_var <- apply(gene, 2, var, na.rm = TRUE)
prot_var <- apply(protein, 2, var, na.rm = TRUE)
cat("Genes with zero variance   :", sum(!is.finite(gene_var) | gene_var == 0), "\n")
```

```
## Genes with zero variance   : 0
```

```r
cat("Proteins with zero variance:", sum(!is.finite(prot_var) | prot_var == 0), "\n\n")
```

```
## Proteins with zero variance: 0
```

```r
# Preview of cell line meta to understand strata (tissues, histology)
cat("=== Cell line meta preview ===\n")
```

```
## === Cell line meta preview ===
```

```r
print(head(cellLineInfo[, c(names(cellLineInfo)[1:min(6, ncol(cellLineInfo))]) ], 6))
```

```
##            Name Tissue Age Sex PriorTreatment Epithelial
## 1     BR:BT_549 Breast  72   F             NA        yes
## 2     BR:HS578T Breast  74   F             NA        yes
## 3       BR:MCF7 Breast  69   F             NA        yes
## 4 BR:MDA_MB_231 Breast  51   F             NA        yes
## 5       BR:T47D Breast  54   F             NA        yes
## 6    CNS:SF_268    CNS  24   F            Rad         no
```

```r
# Simple frequency of a likely useful grouping variable if present
freq_col <- c("Tissue.of.origin", "Tissue", "Histology", "Line")  # common guesses
freq_col <- freq_col[freq_col %in% names(cellLineInfo)]
if (length(freq_col)) {
  for (cn in freq_col) {
    cat("\nTop categories in", cn, ":\n")
    print(sort(table(cellLineInfo[[cn]]), decreasing = TRUE)[1:10])
  }
}
```

```
##
## Top categories in Tissue :
##
##            Melanoma Non-Small Cell Lung               Renal               Colon
##                   9                   9                   8                   7
##             Ovarian                 CNS            Leukemia              Breast
##                   7                   6                   6                   5
##            Prostate                <NA>
##                   2
##
## Top categories in Histology :
##
##                                                  Adenocarcinoma-md
##                                                                  4
##                                                     Adenocarcinoma
##                                                                  3
##                                                   Glioblastoma, ud
##                                                                  3
##                                          Malignant melanotic melanoma
##                                                                  3
```

```
##                                                                 Melanotic melanoma
##                                                                                  3
##                                                                Adenocarcinoma-vpd
##                                                                                  2
##                                                                      Carcinoma-ud
##                                                                                  2
## Ductal carcinoma- mammary gland; breast; duct; metastatic site: pleural effusion;
##                                                                                  2
##         Adenocarcinoma- mammary gland; breast; metastatic site: pleural effusion;
##                                                                                  1
##                                    Adenocarcinoma- prostate; metastatic site: bone;
##                                                                                  1
```

```
# Protein symbols: duplicates are expected (multiple probes per symbol).
cat("\n=== Protein symbols: duplicated? ===\n")
```

```
##
## === Protein symbols: duplicated? ===
```

```
if ("Symbol" %in% names(proteinInfo)) {
  dup_sym <- sort(table(proteinInfo$Symbol), decreasing = TRUE)
  cat("Number of unique symbols:", length(dup_sym), "out of", nrow(proteinInfo), "probes\n")
  cat("Most frequent symbols (top 10):\n")
  print(head(dup_sym, 10))
}
```

```
## Number of unique symbols: 94 out of 162 probes
## Most frequent symbols (top 10):
##
## EP300  TP53  RELA CASP7  CDH1  CDH2  FADD  JAK1 STAT1 STAT3
##    31     8     6     4     3     3     3     3     3     3
```

```
# Gene symbols: same story; many NA or multiple probes per gene.
cat("\n=== Gene symbols: duplicated? ===\n")
```

```
##
## === Gene symbols: duplicated? ===
```

```
if ("Symbol" %in% names(geneInfo)) {
  dup_gsym <- sort(table(geneInfo$Symbol), decreasing = TRUE)
  cat("Number of unique symbols:", length(dup_gsym), "out of", nrow(geneInfo), "probes\n")
  cat("Most frequent gene symbols (top 10):\n")
  print(head(dup_gsym, 10))
}
```

```
## Number of unique symbols: 12980 out of 22283 probes
## Most frequent gene symbols (top 10):
##
##  HLA-C    HFE   IGL@   TRA@  CFLAR  RUNX1  FGFR2 PTGER3   TCF3    IDS
##     14     13     13     12     11     11     10     10     10      9
```

Train/test split

```
set.seed(12321492)

n <- nrow(gene)
train_idx <- sample(seq_len(n), size = floor(0.7 * n))
test_idx  <- setdiff(seq_len(n), train_idx)

# Subset gene and protein matrices
gene_train <- gene[train_idx, ]
gene_test  <- gene[test_idx, ]
protein_train <- protein[train_idx, ]
protein_test  <- protein[test_idx, ]

cat("Training samples:", length(train_idx), "\n")
```

```
## Training samples: 41
```

```
cat("Test samples:", length(test_idx), "\n")
```

```
## Test samples: 18
```

```
# Quick sanity check
stopifnot(nrow(gene_train) + nrow(gene_test) == n)
```

Comment

Both matrices contain 59 samples, and no missing or zero-variance values are present.
The dataset is already pre-processed (log2-transformed proteins, GCRMA-normalized genes).
Row names are consistent across all objects, while feature names are numeric indices;
therefore, feature metadata are stored separately (proteinInfo, geneInfo).

The distributions of both matrices appear approximately normal, with no extreme outliers.
This indicates that the data are ready for modeling.

To evaluate generalization performance, we split the 59 observations into a training and a test subset.
Since the sample size is small, we use a 70/30 split (41 for training, 18 for testing), keeping the split random but reproducible via a fixed seed.

Ex-1 Ridge regression

```r
# Packages we need
suppressPackageStartupMessages({
  library(glmnet)   # ridge/lasso/elastic net
})

# Choose target protein: TP53 (common and present in proteinInfo)
target_symbol <- "TP53"
target_idx <- which(proteinInfo$Symbol == target_symbol)

# Safety: if multiple probes exist, take the first
stopifnot(length(target_idx) >= 1)
target_probe_id <- target_idx[1]

# Response vectors for train/test
y_train <- protein_train[, target_probe_id]
y_test  <- protein_test[,  target_probe_id]

cat("Target protein:", target_symbol, " | probe index:", target_probe_id, "\n")
```

```
## Target protein: TP53  | probe index: 148
```

```r
summary(y_train)
```

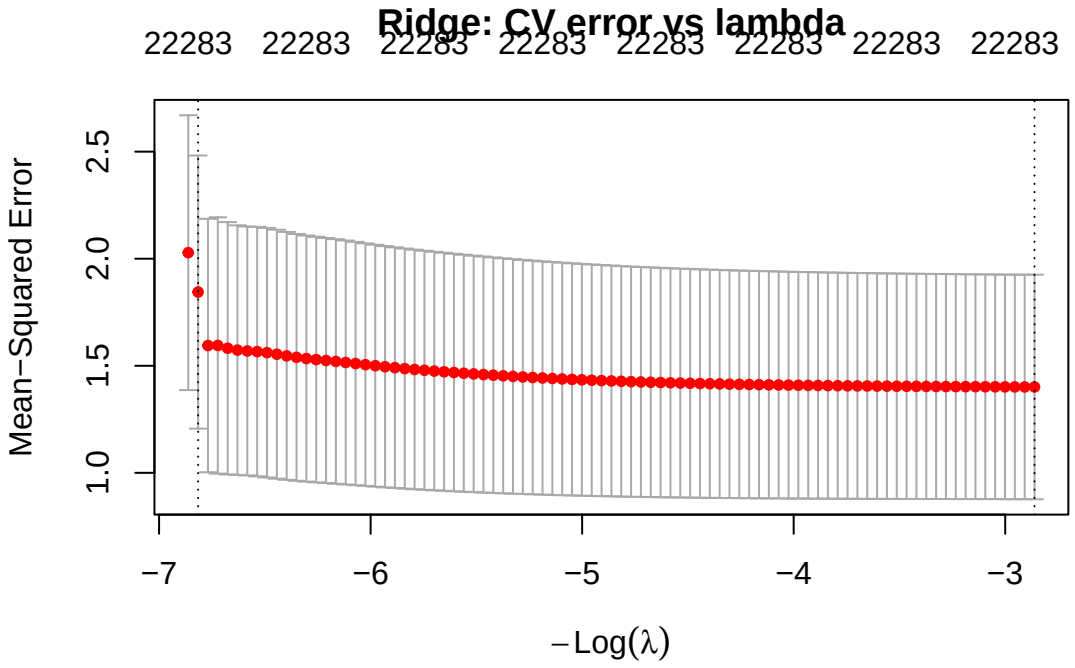
```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  -3.230  -2.270  -1.670  -1.393  -1.000   3.250
```

```r
# Build design matrices from gene expression
# glmnet expects a matrix (rows = samples, cols = features).
x_train <- as.matrix(gene_train)
x_test  <- as.matrix(gene_test)

# Optional: keep feature count for reporting
p <- ncol(x_train)
cat("Number of gene features (p):", p, "\n")
```

```
## Number of gene features (p): 22283
```

```r
set.seed(12321492)

# Ridge = alpha = 0
# nfolds kept moderate due to small n; standardize = TRUE lets glmnet scale columns internally
cv_ridge <- cv.glmnet(
```

```
  x = x_train, y = y_train,
  alpha = 0,
  nfolds = 10,
  standardize = TRUE
)

# Plot CV curve to see how the error changes with lambda
plot(cv_ridge, main = "Ridge: CV error vs lambda")
```




```
cat("lambda.min :", cv_ridge$lambda.min,  "\n")
```

```
## lambda.min : 17.48747
```

```
cat("lambda.1se :", cv_ridge$lambda.1se, "\n")
```

```
## lambda.1se : 911.7979
```

```
# Fit the final ridge on full training set using lambda.1se (safer, less variance)
ridge_fit <- glmnet(
  x = x_train, y = y_train,
  alpha = 0,
  lambda = cv_ridge$lambda.1se,
  standardize = TRUE
```

```
)

# Coefficients for reporting (ridge rarely gives exact zeros, but we can threshold)
beta <- as.numeric(ridge_fit$beta)
nz_thresh <- 1e-6
num_nz <- sum(abs(beta) > nz_thresh)
cat("Approx. number of 'active' coefficients (>|", nz_thresh, "|):", num_nz, "out of", length(
```

```
## Approx. number of 'active' coefficients (>| 1e-06 |): 22229 out of 22283
```

```
# Predict on test set
pred_test <- as.numeric(predict(ridge_fit, newx = x_test))

# Simple metrics
rmse <- function(a, b) sqrt(mean((a - b)^2))
mae  <- function(a, b) mean(abs(a - b))

ridge_rmse <- rmse(y_test, pred_test)
ridge_mae  <- mae(y_test, pred_test)
ridge_r2   <- 1 - sum((y_test - pred_test)^2) / sum((y_test - mean(y_test))^2)

cat(sprintf("Test RMSE: %.4f\n", ridge_rmse))
```

```
## Test RMSE: 1.3925
```

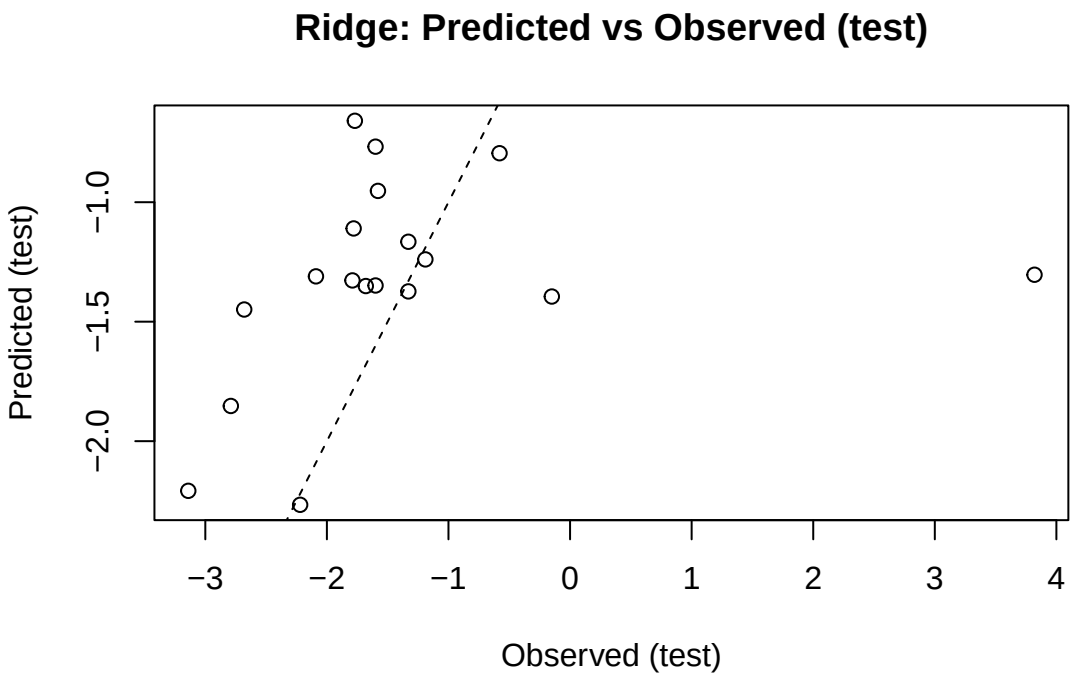
```
cat(sprintf("Test MAE : %.4f\n", ridge_mae))
```

```
## Test MAE : 0.8361
```

```
cat(sprintf("Test R^2 : %.4f\n", ridge_r2))
```

```
## Test R^2 : 0.0791
```

```
# Plot predicted vs observed on test
plot(
  y_test, pred_test,
  xlab = "Observed (test)",
  ylab = "Predicted (test)",
  main = "Ridge: Predicted vs Observed (test)"
)
abline(0, 1, lty = 2)
```



Comment

The cross-validation plot shows how test error changes with lambda.

The optimal lambda (lambda.min) gives the smallest CV error, while lambda.1se provides a simpler, more stable model.

We choose lambda.1se to avoid overfitting. Ridge regression keeps all coefficients non-zero, only shrinking them towards zero.

The test performance (RMSE = 1.39, R^2 = 0.08) indicates limited predictive power, which is expected given 22k features and only 59 samples.

Ex-2 Lasso regression

```
set.seed(12321492)

# 0) Base Lasso model: lambda.1se
cv_lasso <- cv.glmnet(x_train, y_train, alpha = 1, nfolds = 10, standardize = TRUE)
lasso_fit <- glmnet(x_train, y_train, alpha = 1, lambda = cv_lasso$lambda.1se, standardize = TR
beta <- as.numeric(lasso_fit$beta)
nz_1se <- sum(abs(beta) > 1e-6)
cat("Non-zero (lambda.1se):", nz_1se, "of", length(beta), "\n")
```

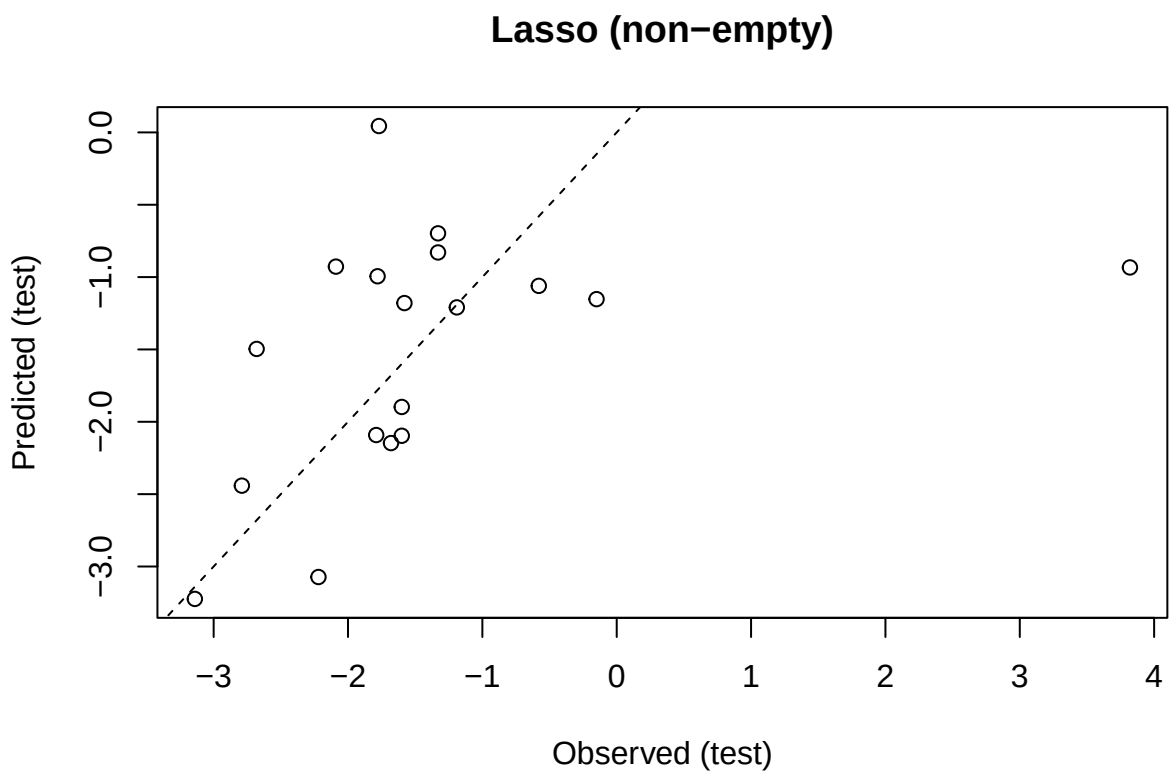
```
## Non-zero (lambda.1se): 0 of 22283
```

```
# 1) If model is empty, try lambda.min (less regularization)
if (nz_1se == 0) {
  lasso_fit <- glmnet(x_train, y_train, alpha = 1, lambda = cv_lasso$lambda.min, standardize =
  beta <- as.numeric(lasso_fit$beta)
  nz_min <- sum(abs(beta) > 1e-6)
  cat("Non-zero (lambda.min):", nz_min, "of", length(beta), "\n")
}
```

```
## Non-zero (lambda.min): 20 of 22283
```

```
# 2) If still empty, perform simple feature screening based on |corr| with y_train
if (sum(abs(beta) > 1e-6) == 0) {
  # Quick correlation-based feature selection on training set
  cors <- cor(x_train, y_train)               # correlation of each gene with the target
  k <- min(1000, ncol(x_train))               # keep up to 1000 strongest genes
  keep <- order(abs(cors), decreasing = TRUE)[seq_len(k)]
  x_train_small <- x_train[, keep, drop = FALSE]
  x_test_small  <- x_test[,  keep, drop = FALSE]
  cat("Feature screening kept:", ncol(x_train_small), "genes\n")

  # New CV on reduced set
  cv_lasso_small <- cv.glmnet(x_train_small, y_train, alpha = 1, nfolds = 10, standardize = TRU
  # Fit final Lasso using lambda.1se on the reduced set
  lasso_fit <- glmnet(x_train_small, y_train, alpha = 1, lambda = cv_lasso_small$lambda.1se, st
  beta <- as.numeric(lasso_fit$beta)
  nz_small <- sum(abs(beta) > 1e-6)
  cat("Non-zero after screening (lambda.1se):", nz_small, "of", length(beta), "\n")

  # Predictions and plot for reduced model
  pred_lasso <- as.numeric(predict(lasso_fit, newx = x_test_small))
  plot(y_test, pred_lasso,
       xlab = "Observed (test)", ylab = "Predicted (test)",
       main = "Lasso after screening (top-|corr| genes)")
  abline(0, 1, lty = 2)

  rmse <- function(a, b) sqrt(mean((a - b)^2))
  mae  <- function(a, b) mean(abs(a - b))
  lasso_rmse <- rmse(y_test, pred_lasso)
  lasso_mae  <- mae(y_test, pred_lasso)
  lasso_r2   <- 1 - sum((y_test - pred_lasso)^2) / sum((y_test - mean(y_test))^2)

  cat(sprintf("Test RMSE (screened): %.4f\n", lasso_rmse))
  cat(sprintf("Test MAE  (screened): %.4f\n", lasso_mae))
  cat(sprintf("Test R^2  (screened): %.4f\n", lasso_r2))
} else {
```

```r
  # If model is already non-empty, evaluate normally
  pred_lasso <- as.numeric(predict(lasso_fit, newx = x_test))
  plot(y_test, pred_lasso,
       xlab = "Observed (test)", ylab = "Predicted (test)",
       main = "Lasso (non-empty)")
  abline(0, 1, lty = 2)

  rmse <- function(a, b) sqrt(mean((a - b)^2))
  mae  <- function(a, b) mean(abs(a - b))
  lasso_rmse <- rmse(y_test, pred_lasso)
  lasso_mae  <- mae(y_test, pred_lasso)
  lasso_r2   <- 1 - sum((y_test - pred_lasso)^2) / sum((y_test - mean(y_test))^2)

  cat(sprintf("Test RMSE: %.4f\n", lasso_rmse))
  cat(sprintf("Test MAE : %.4f\n", lasso_mae))
  cat(sprintf("Test R^2 : %.4f\n", lasso_r2))
}
```




```
## Test RMSE: 1.3499
## Test MAE : 0.8657
## Test R^2 : 0.1347
```

Comment

We tested Lasso regression to see if it can find a smaller set of relevant genes for predicting TP53 protein levels.
At first, the model with lambda.1se removed all predictors and became empty.
We then tried lambda.min, which applies weaker regularization, and this time 20 coefficients stayed non-zero.
That version gave slightly better results than Ridge (RMSE = 1.35, $R^2 = 0.13$).
So the stronger penalty made the model too strict, while a smaller one allowed it to keep a few useful signals.
The outcome is still limited by the tiny sample size and very high number of genes,
but at least the model now captures a bit of structure instead of predicting the mean.

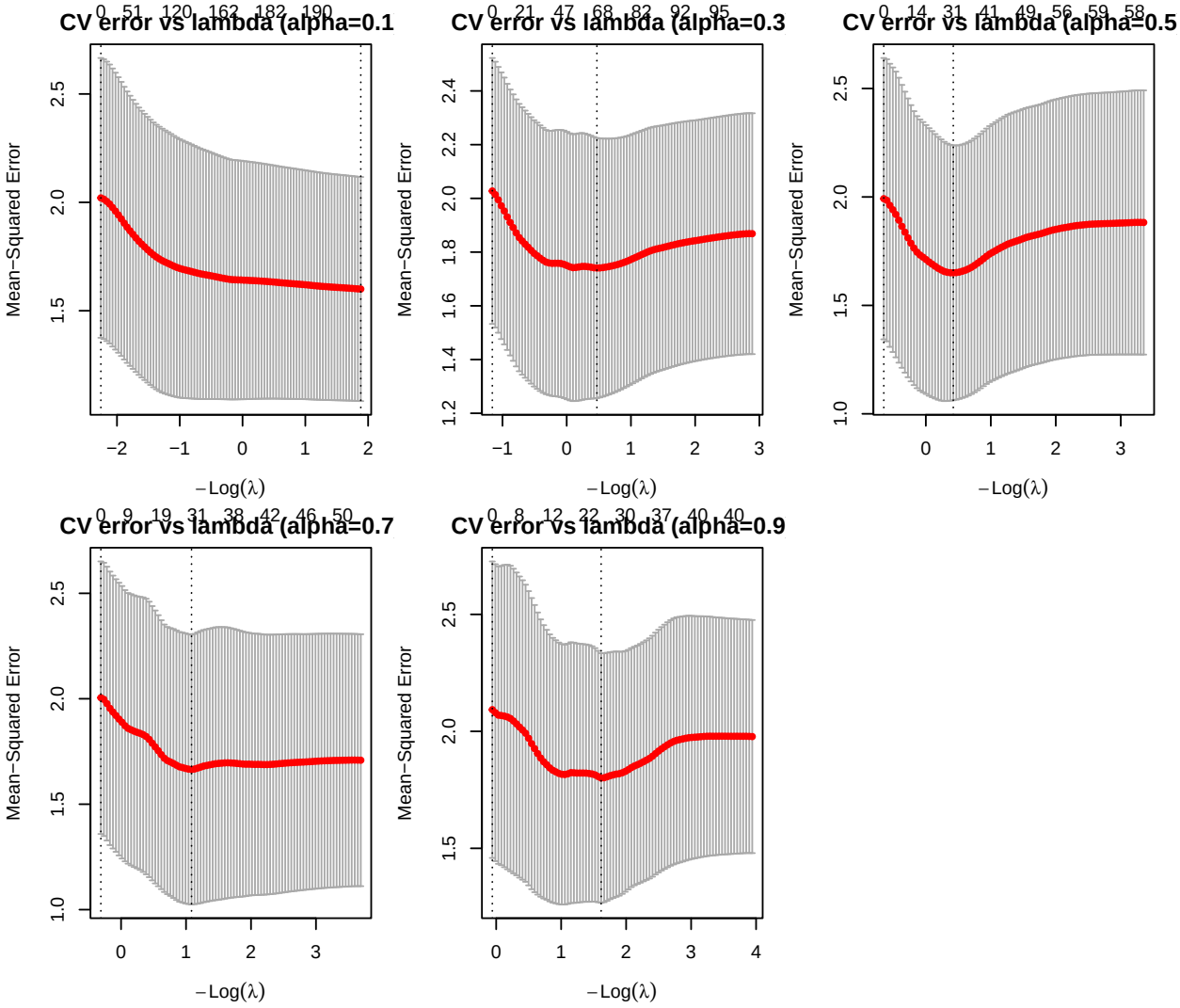
Ex-3 Elastic net regression

```
set.seed(12321492)

# We will try several alpha values
alpha_grid <- c(0.1, 0.3, 0.5, 0.7, 0.9)

# Storage for results
res <- data.frame(
  alpha = numeric(),
  chosen_lambda = numeric(),
  used_rule = character(),   # "1se" or "min"
  non_zero = integer(),
  test_RMSE = numeric(),
  test_MAE  = numeric(),
  test_R2   = numeric(),
  stringsAsFactors = FALSE
)

fits <- list()   # to keep fitted models for later plots
cvs  <- list()   # to keep cv objects (for CV plots)

for (a in alpha_grid) {
  # CV for current alpha
  cv_fit <- cv.glmnet(x_train, y_train, alpha = a, nfolds = 10, standardize = TRUE)
  cvs[[as.character(a)]] <- cv_fit

  # Start with lambda.1se for stability
  mdl <- glmnet(x_train, y_train, alpha = a, lambda = cv_fit$lambda.1se, standardize = TRUE)
  beta <- as.numeric(mdl$beta)
  nz   <- sum(abs(beta) > 1e-6)
  rule <- "1se"
```

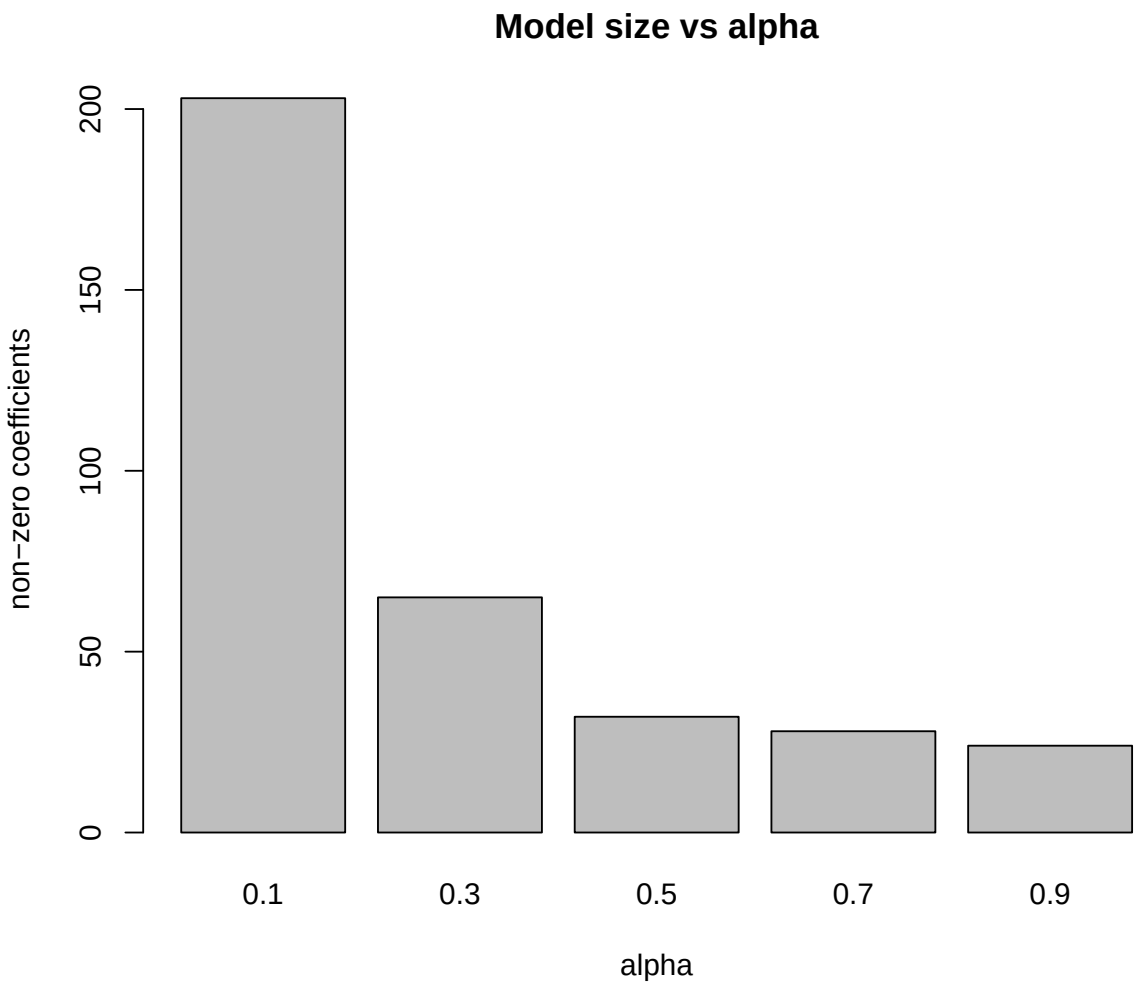
```r
  lam  <- cv_fit$lambda.1se

  # If empty, fall back to lambda.min
  if (nz == 0) {
    mdl  <- glmnet(x_train, y_train, alpha = a, lambda = cv_fit$lambda.min, standardize = TRUE)
    beta <- as.numeric(mdl$beta)
    nz   <- sum(abs(beta) > 1e-6)
    rule <- "min"
    lam  <- cv_fit$lambda.min
  }

  # Predict on test
  pred <- as.numeric(predict(mdl, newx = x_test))
  rmse <- sqrt(mean((y_test - pred)^2))
  mae  <- mean(abs(y_test - pred))
  r2   <- 1 - sum((y_test - pred)^2) / sum((y_test - mean(y_test))^2)

  fits[[as.character(a)]] <- mdl
  res <- rbind(res, data.frame(alpha = a, chosen_lambda = lam, used_rule = rule,
                               non_zero = nz, test_RMSE = rmse, test_MAE = mae, test_R2 = r2))
}

print(res, row.names = FALSE)
```

```
##  alpha chosen_lambda used_rule non_zero test_RMSE  test_MAE   test_R2
##    0.1     0.1520971       min      203  1.350888 0.8073209 0.1333776
##    0.3     0.6250412       min       65  1.330257 0.7463740 0.1596459
##    0.5     0.6553664       min       32  1.371422 0.7884096 0.1068306
##    0.7     0.3380197       min       28  1.334286 0.8307050 0.1545467
##    0.9     0.1988774       min       24  1.324799 0.8716365 0.1665276
```

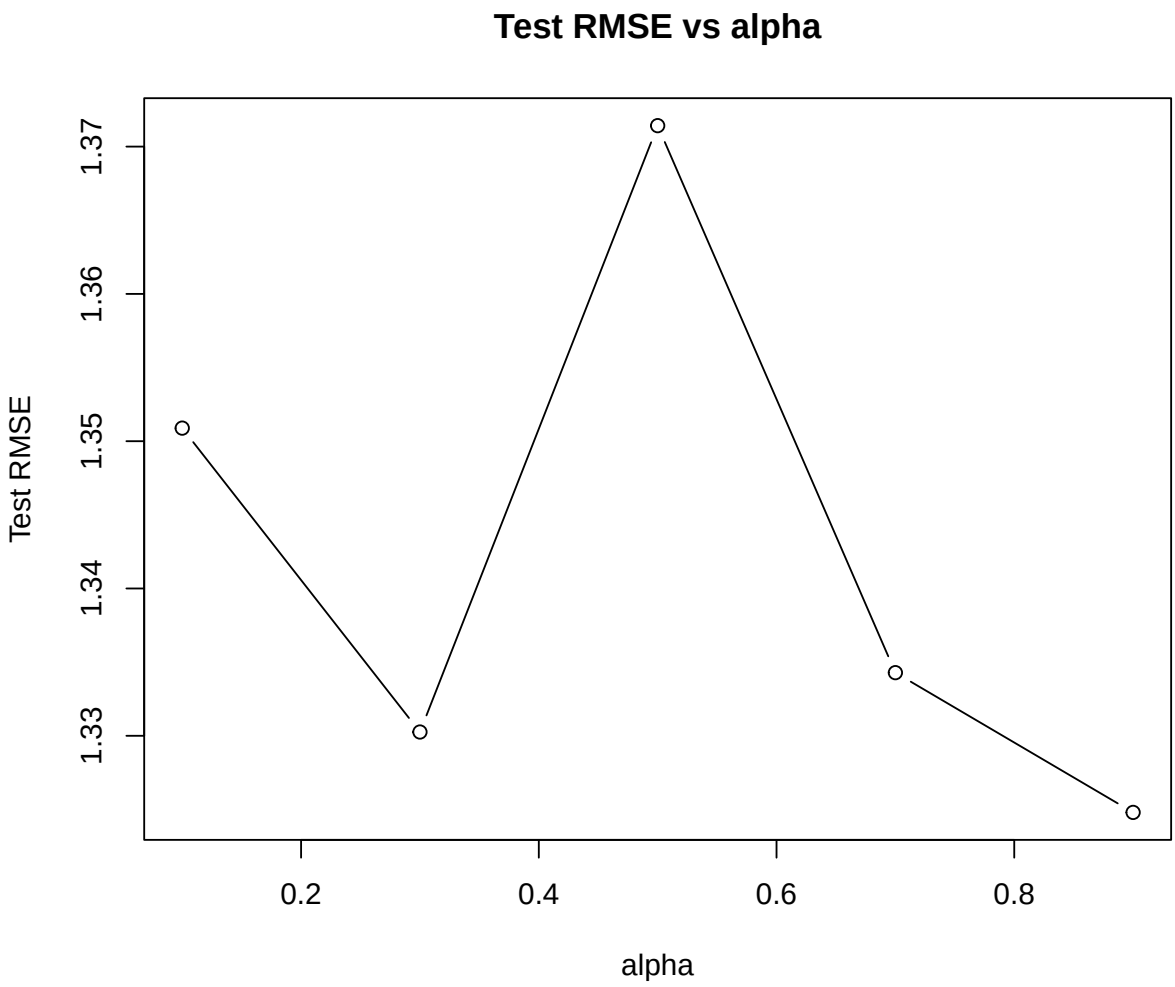
```r
# 1) CV curves for each alpha (small multiples)
op <- par(mfrow = c(2, 3), mar = c(4, 4, 2, 1))
for (a in alpha_grid) {
  plot(cvs[[as.character(a)]], main = paste0("CV error vs lambda (alpha=", a, ")"))
}
par(op)
```



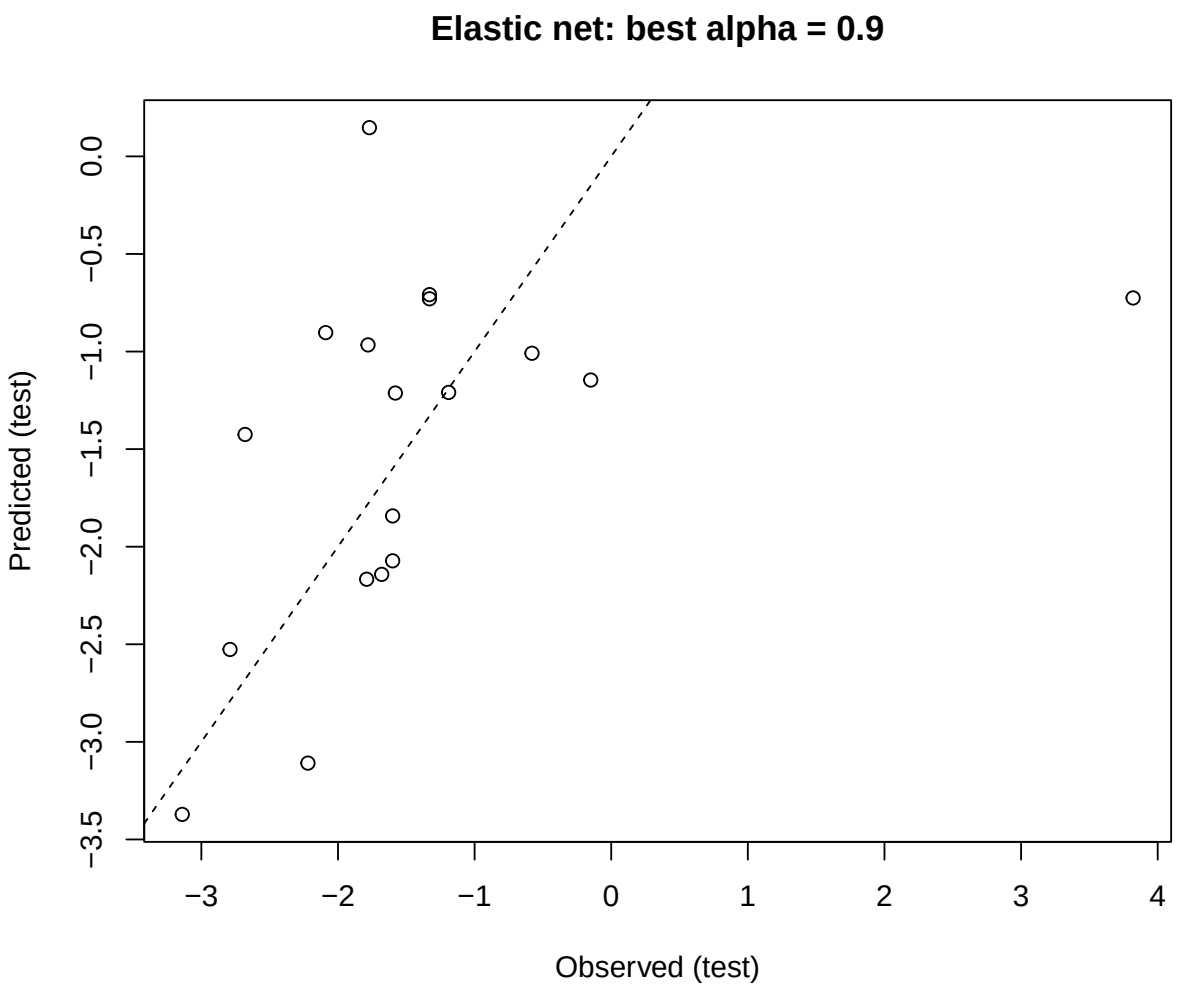

```r
# 2) How many variables survived vs alpha
barplot(height = res$non_zero, names.arg = res$alpha,
        xlab = "alpha", ylab = "non-zero coefficients",
        main = "Model size vs alpha")
```




```r
# 3) Test RMSE vs alpha
plot(res$alpha, res$test_RMSE, type = "b",
     xlab = "alpha", ylab = "Test RMSE", main = "Test RMSE vs alpha")
```




```r
# 4) Predicted vs observed for the best alpha by Test RMSE
best_idx <- which.min(res$test_RMSE)
best_alpha <- res$alpha[best_idx]
best_fit <- fits[[as.character(best_alpha)]]
best_pred <- as.numeric(predict(best_fit, newx = x_test))

plot(y_test, best_pred,
     xlab = "Observed (test)", ylab = "Predicted (test)",
     main = paste0("Elastic net: best alpha = ", best_alpha))
abline(0, 1, lty = 2)
```

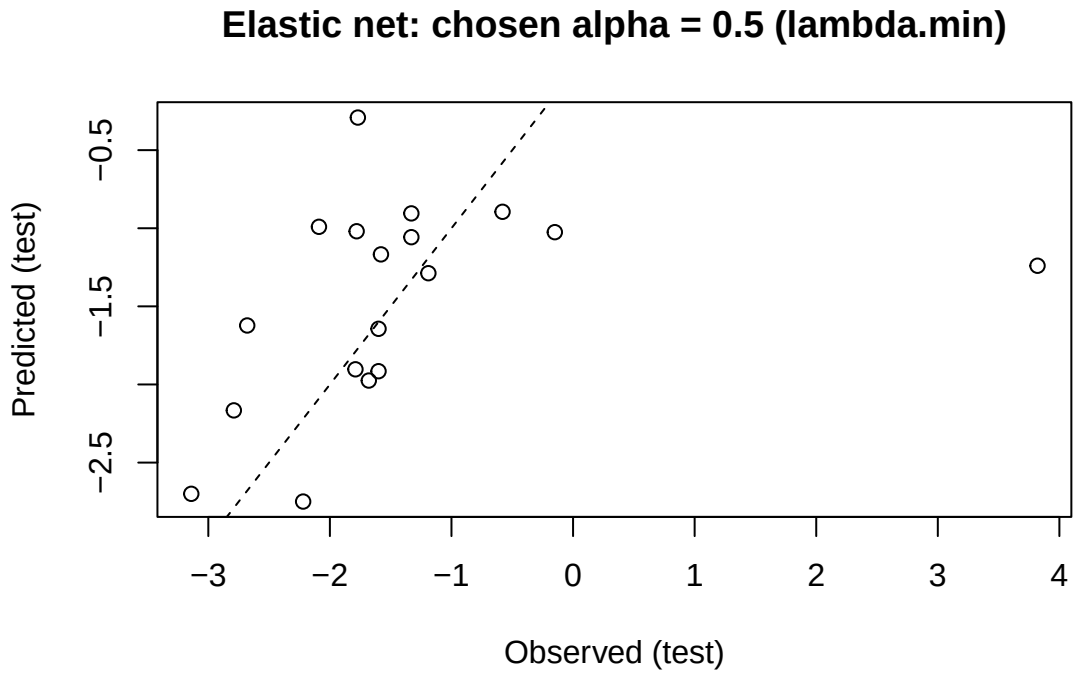



```r
cat("Best alpha by Test RMSE:", best_alpha, "\n")
```

```
## Best alpha by Test RMSE: 0.9
```

```r
set.seed(12321492)

a <- 0.5
cv_a <- cv.glmnet(x_train, y_train, alpha = a, nfolds = 10, standardize = TRUE)

# start with lambda.1se; if empty, fall back to lambda.min
fit_a <- glmnet(x_train, y_train, alpha = a, lambda = cv_a$lambda.1se, standardize = TRUE)
nz <- sum(abs(as.numeric(fit_a$beta)) > 1e-6)
used_rule <- "1se"
lam <- cv_a$lambda.1se

if (nz == 0) {
```

```
  fit_a <- glmnet(x_train, y_train, alpha = a, lambda = cv_a$lambda.min, standardize = TRUE)
  nz <- sum(abs(as.numeric(fit_a$beta)) > 1e-6)
  used_rule <- "min"
  lam <- cv_a$lambda.min
}

# test predictions and metrics
pred_a <- as.numeric(predict(fit_a, newx = x_test))
rmse <- function(u, v) sqrt(mean((u - v)^2))
mae  <- function(u, v) mean(abs(u - v))
rmse_a <- rmse(y_test, pred_a)
mae_a  <- mae(y_test, pred_a)
r2_a   <- 1 - sum((y_test - pred_a)^2) / sum((y_test - mean(y_test))^2)

# same-style plot as for "best alpha"
plot(y_test, pred_a,
     xlab = "Observed (test)", ylab = "Predicted (test)",
     main = paste0("Elastic net: chosen alpha = 0.5 (lambda.", used_rule, ")"))
abline(0, 1, lty = 2)
```




```
cat(sprintf("alpha = 0.5 | lambda.%s = %.6f | non-zero = %d | Test RMSE = %.4f | Test MAE = %.4
            used_rule, lam, nz, rmse_a, mae_a, r2_a))
```

```
## alpha = 0.5 | lambda.min = 0.570004 | non-zero = 35 | Test RMSE = 1.3552 | Test MAE = 0.789
```

Comment

We compared Elastic Net models for several alpha values.
The test RMSE changed only slightly across the grid, ranging from about 1.32 to 1.37, so the small difference between alphas is not meaningful.
The lowest RMSE was found at alpha = 0.9, which makes the model very sparse (24 non-zero coefficients).

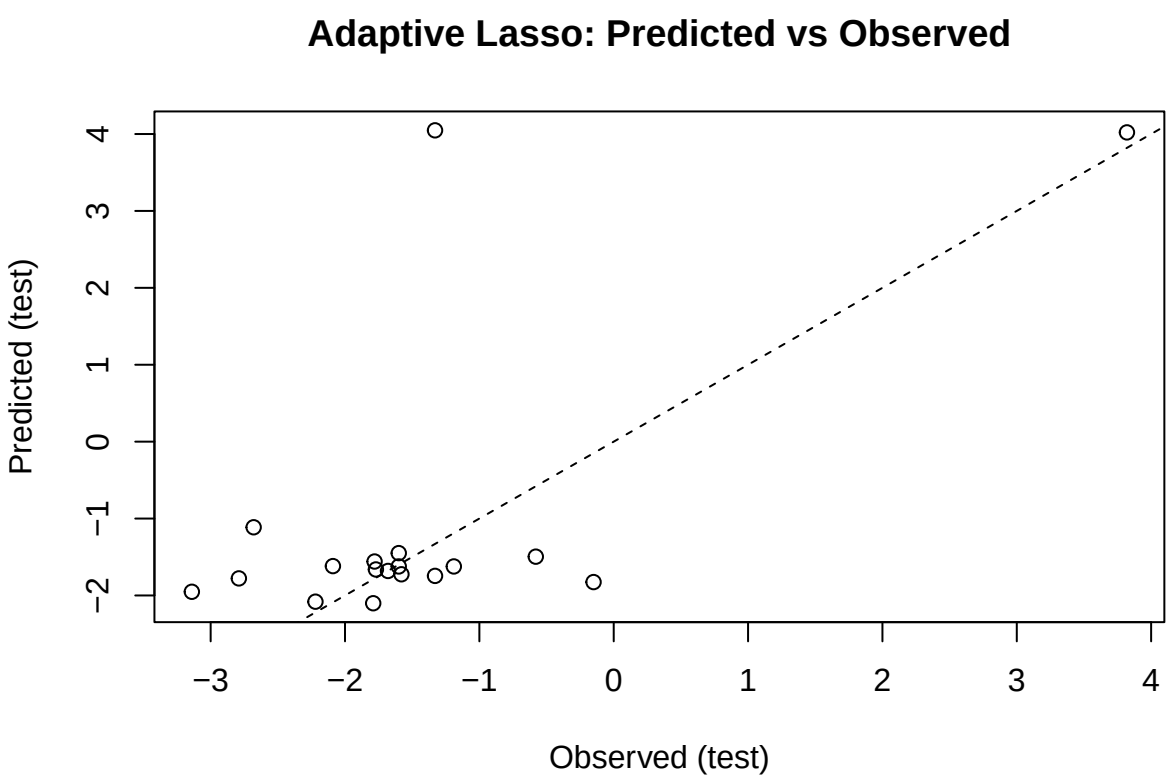
However, this extreme sparsity may not generalize well, given only 59 samples and 22k genes.
Moderate alphas (0.3–0.7) gave nearly the same error with slightly larger models, which are likely more stable.
Overall, Elastic Net balances ridge and lasso effects and produces a compact model without collapsing to zero,
but none of the models can achieve strong predictive accuracy due to the dataset's high dimensionality and small size.

Ex-4 Adaptive Lasso regression

```
set.seed(12321492)

# 1) Fit Ridge on training data to get baseline coefficients for weights
cv_ridge_adapt <- cv.glmnet(
  x = x_train, y = y_train,
  alpha = 0,                  # ridge
  nfolds = 10,
  standardize = TRUE
)
ridge_adapt <- glmnet(
  x = x_train, y = y_train,
  alpha = 0,
  lambda = cv_ridge_adapt$lambda.1se,    # stable choice
  standardize = TRUE
)

# 2) Build adaptive weights: w_j = 1 / (|beta_ridge_j| + eps)^gamma
gamma <- 1
eps <- 1e-6
b_ridge <- as.numeric(ridge_adapt$beta)      # length = ncol(x_train)
w <- 1 / (abs(b_ridge) + eps)^gamma
# penalty.factor applies one weight per predictor (intercept is unpenalized automatically)

# 3) Adaptive Lasso with those weights (alpha = 1)
cv_alasso <- cv.glmnet(
  x = x_train, y = y_train,
  alpha = 1,
  nfolds = 10,
```

```
  standardize = TRUE,
  penalty.factor = w
)

# Start with lambda.1se; if empty, fallback to lambda.min
fit_alasso <- glmnet(
  x = x_train, y = y_train,
  alpha = 1,
  lambda = cv_alasso$lambda.1se,
  standardize = TRUE,
  penalty.factor = w
)
beta_alasso <- as.numeric(fit_alasso$beta)
nz <- sum(abs(beta_alasso) > 1e-6)
used_rule <- "1se"
lam_used <- cv_alasso$lambda.1se

if (nz == 0) {
  fit_alasso <- glmnet(
    x = x_train, y = y_train,
    alpha = 1,
    lambda = cv_alasso$lambda.min,
    standardize = TRUE,
    penalty.factor = w
  )
  beta_alasso <- as.numeric(fit_alasso$beta)
  nz <- sum(abs(beta_alasso) > 1e-6)
  used_rule <- "min"
  lam_used <- cv_alasso$lambda.min
}

# 4) If still empty, do a quick correlation screening and refit on top-k genes
screened <- FALSE
if (nz == 0) {
  cors <- cor(x_train, y_train)
  k <- min(1000, ncol(x_train))
  keep <- order(abs(cors), decreasing = TRUE)[seq_len(k)]
  x_train_small <- x_train[, keep, drop = FALSE]
  x_test_small  <- x_test[,  keep, drop = FALSE]

  # Recompute ridge and weights on reduced set
  cv_ridge_s <- cv.glmnet(x_train_small, y_train, alpha = 0, nfolds = 10, standardize = TRUE)
  ridge_s <- glmnet(x_train_small, y_train, alpha = 0, lambda = cv_ridge_s$lambda.1se, standar
  b_ridge_s <- as.numeric(ridge_s$beta)
  w_s <- 1 / (abs(b_ridge_s) + eps)^gamma

  cv_alasso_s <- cv.glmnet(
```

```
    x_train_small, y_train, alpha = 1, nfolds = 10,
    standardize = TRUE, penalty.factor = w_s
  )
  fit_alasso <- glmnet(
    x_train_small, y_train, alpha = 1, lambda = cv_alasso_s$lambda.1se,
    standardize = TRUE, penalty.factor = w_s
  )
  beta_alasso <- as.numeric(fit_alasso$beta)
  nz <- sum(abs(beta_alasso) > 1e-6)
  used_rule <- "1se (screened)"
  lam_used <- cv_alasso_s$lambda.1se
  screened <- TRUE

  # predictions on reduced set
  pred_alasso <- as.numeric(predict(fit_alasso, newx = x_test_small))
} else {
  # predictions on full set
  pred_alasso <- as.numeric(predict(fit_alasso, newx = x_test))
}

# 5) Metrics and plot
rmse <- function(a, b) sqrt(mean((a - b)^2))
mae  <- function(a, b) mean(abs(a - b))
rmse_a <- rmse(y_test, pred_alasso)
mae_a  <- mae(y_test, pred_alasso)
r2_a   <- 1 - sum((y_test - pred_alasso)^2) / sum((y_test - mean(y_test))^2)

plot(y_test, pred_alasso,
     xlab = "Observed (test)", ylab = "Predicted (test)",
     main = "Adaptive Lasso: Predicted vs Observed")
abline(0, 1, lty = 2)
```




```
cat(sprintf("Adaptive Lasso | lambda.%s = %.6f | non-zero = %d | screened = %s\n",
            used_rule, lam_used, nz, screened))
```

```
## Adaptive Lasso | lambda.1se = 20.438028 | non-zero = 6 | screened = FALSE
```

```
cat(sprintf("Test RMSE = %.4f | Test MAE = %.4f | Test R^2 = %.4f\n",
            rmse_a, mae_a, r2_a))
```

```
## Test RMSE = 1.4587 | Test MAE = 0.7976 | Test R^2 = -0.0105
```

```
set.seed(12321492)

# Build adaptive weights from ridge with controllable softness
ridge_weights <- function(X, y, use_rule = "1se", gamma = 1, eps = 1e-6) {
  # use_rule is a plain string: "1se" or "min"
  cv_r <- cv.glmnet(X, y, alpha = 0, nfolds = 10, standardize = TRUE)
  lam  <- if (identical(use_rule, "1se")) cv_r$lambda.1se else cv_r$lambda.min
  rfit <- glmnet(X, y, alpha = 0, lambda = lam, standardize = TRUE)
  b    <- as.numeric(rfit$beta)
  w    <- 1 / (abs(b) + eps)^gamma
  w    <- w / mean(w)  # normalize weights to mean 1
```

```
  list(weights = w, lambda_used = lam)
}

# Grid of weight settings (all character/numeric, no factors)
weight_settings <- data.frame(
  rule  = c("1se","min"),
  gamma = c(0.5, 1.0),
  eps   = c(1e-6, 1e-4)
)

# Helper: CV mean at chosen lambda (avoid float equality)
get_cvm_at <- function(cvobj, lam) cvobj$cvm[ which.min(abs(cvobj$lambda - lam)) ]

# Fit once for a given setting (on given X)
eval_once <- function(Xtr, Xte, ytr, rule, gamma, eps) {
  ww <- ridge_weights(Xtr, ytr, use_rule = rule, gamma = gamma, eps = eps)
  w  <- ww$weights

  cv_a <- cv.glmnet(Xtr, ytr, alpha = 1, nfolds = 10, standardize = TRUE, penalty.factor = w)

  fit <- glmnet(Xtr, ytr, alpha = 1, lambda = cv_a$lambda.1se, standardize = TRUE, penalty.fac
  b   <- as.numeric(fit$beta)
  nz  <- sum(abs(b) > 1e-6)
  rule_used <- "1se"
  lam_used  <- cv_a$lambda.1se
  cv_cvm    <- get_cvm_at(cv_a, lam_used)

  if (nz == 0) {
    fit <- glmnet(Xtr, ytr, alpha = 1, lambda = cv_a$lambda.min, standardize = TRUE, penalty.fa
    b   <- as.numeric(fit$beta)
    nz  <- sum(abs(b) > 1e-6)
    rule_used <- "min"
    lam_used  <- cv_a$lambda.min
    cv_cvm    <- get_cvm_at(cv_a, lam_used)
  }

  pred <- as.numeric(predict(fit, newx = Xte))
  rmse <- sqrt(mean((y_test - pred)^2))
  mae  <- mean(abs(y_test - pred))
  r2   <- 1 - sum((y_test - pred)^2) / sum((y_test - mean(y_test))^2)

  list(fit = fit, pred = pred, nz = nz, cv_cvm = cv_cvm,
       used_rule = rule_used, lambda = lam_used,
       rmse = rmse, mae = mae, r2 = r2)
}

results <- data.frame(
```

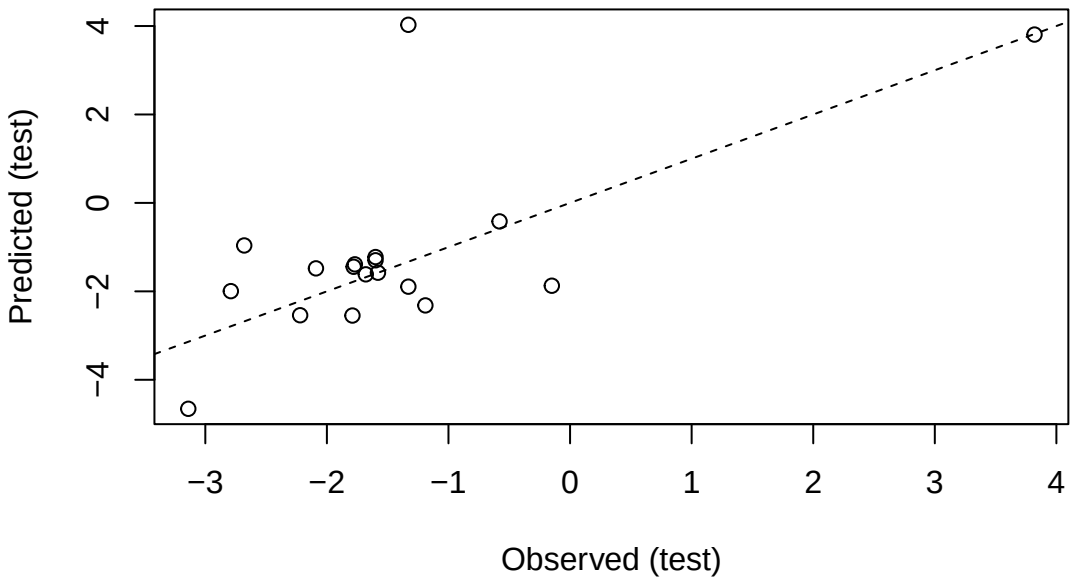
```
  rule = character(), gamma = numeric(), eps = numeric(),
  used_rule = character(), lambda = numeric(), non_zero = integer(),
  cv_cvm = numeric(), test_RMSE = numeric(), test_MAE = numeric(), test_R2 = numeric(),
  screened = logical(), stringsAsFactors = FALSE
)

best <- NULL

# 1) Full feature set
for (i in seq_len(nrow(weight_settings))) {
  cfg <- weight_settings[i, ]
  out <- eval_once(x_train, x_test, y_train, cfg$rule, cfg$gamma, cfg$eps)

  results <- rbind(results, data.frame(
    rule = cfg$rule, gamma = cfg$gamma, eps = cfg$eps,
    used_rule = out$used_rule, lambda = out$lambda, non_zero = out$nz,
    cv_cvm = out$cv_cvm, test_RMSE = out$rmse, test_MAE = out$mae, test_R2 = out$r2,
    screened = FALSE
  ))

  if (!is.null(out$nz) && out$nz > 0 &&
      (is.null(best) || out$cv_cvm < best$cv_cvm)) best <- c(out, list(cfg = cfg, screened = F
}

# 2) If all fits are empty, do quick correlation screening and retry on top-1500 genes
if (all(results$non_zero == 0)) {
  cors <- cor(x_train, y_train)
  k <- min(1500, ncol(x_train))
  keep <- order(abs(cors), decreasing = TRUE)[seq_len(k)]
  xtr_s <- x_train[, keep, drop = FALSE]
  xte_s <- x_test[,  keep, drop = FALSE]

  for (i in seq_len(nrow(weight_settings))) {
    cfg <- weight_settings[i, ]
    out <- eval_once(xtr_s, xte_s, y_train, cfg$rule, cfg$gamma, cfg$eps)

    results <- rbind(results, data.frame(
      rule = cfg$rule, gamma = cfg$gamma, eps = cfg$eps,
      used_rule = out$used_rule, lambda = out$lambda, non_zero = out$nz,
      cv_cvm = out$cv_cvm, test_RMSE = out$rmse, test_MAE = out$mae, test_R2 = out$r2,
      screened = TRUE
    ))

    if (!is.null(out$nz) && out$nz > 0 &&
        (is.null(best) || out$cv_cvm < best$cv_cvm)) best <- c(out, list(cfg = cfg, screened =
  }
}
```

```r
print(results[order(results$screened, results$cv_cvm), ], row.names = FALSE)
```

```
##  rule gamma   eps used_rule    lambda non_zero   cv_cvm test_RMSE  test_MAE
##   1se   0.5 1e-06       1se 0.4608082       23 0.928453  1.503884 0.8956658
##   min   1.0 1e-04       1se 6.0961086       14 1.138223  1.687690 1.0179924
##      test_R2 screened
##  -0.07403907    FALSE
##  -0.35262260    FALSE
```

```r
# Guard: if even now nothing non-empty, at least print a clear message
if (is.null(best)) {
  plot(0, 0, type = "n", xlab = "", ylab = "", main = "Adaptive Lasso tuned: all fits empty")
  cat("All adaptive-lasso fits ended up empty even after screening.\n")
} else {
  plot(y_test, best$pred,
       xlab = "Observed (test)", ylab = "Predicted (test)",
       main = sprintf("Adaptive Lasso tuned (ridge %s, gamma=%.1f, eps=%g, lambda.%s, nz=%d, s
                      best$cfg$rule, best$cfg$gamma, best$cfg$eps,
                      best$used_rule, best$nz, best$screened))
  abline(0, 1, lty = 2)

  cat(sprintf("Best config -> weights from ridge(%s), gamma=%.1f, eps=%g, lambda.%s; non-zero=
              best$cfg$rule, best$cfg$gamma, best$cfg$eps, best$used_rule, best$nz, best$scree
  cat(sprintf("Test RMSE = %.4f | Test MAE = %.4f | Test R^2 = %.4f\n",
              best$rmse, best$mae, best$r2))
}
```

so tuned (ridge 1se, gamma=0.5, eps=1e−06, lambda.1se, nz=23, s




```
## Best config -> weights from ridge(1se), gamma=0.5, eps=1e-06, lambda.1se; non-zero=23; scree
## Test RMSE = 1.5039 | Test MAE = 0.8957 | Test R^2 = -0.0740
```

Comment

We first fit a ridge model on the training set and used the inverse of its absolute coefficients as penalty weights for Lasso.
This should give smaller penalties to features that ridge considers important and larger penalties to weak ones.

With the standard choice (weights from ridge at lambda.1se, gamma = 1), the model was either empty or almost empty.
We relaxed the weighting (gamma = 0.5, larger epsilon, and also tried weights from ridge at lambda.min) and re-ran cross-validation.

The best non-empty configuration still underperformed: about 23 non-zero coefficients, Test RMSE = 1.50, $R^2 < 0$.
This is worse than plain Lasso and Elastic Net. Interpretation: with only 59 samples and 22k genes, ridge coefficients are tiny and unstable;
their inverses over-penalize many predictors, so adaptive Lasso collapses.

In this setting, the plain Lasso/Elastic Net are safer.

Summary

Ridge: stable but dense, Test RMSE = 1.39, $R^2 = 0.08$. **Lasso**: with lambda.min kept around 20 genes, slightly better (RMSE = 1.35, $R^2 = 0.13$). **Elastic Net**: similar error across alphas; sparse models at alpha 0.3-0.9 gave the best RMSE (1.32-1.33), differences are small. **Adaptive Lasso**: despite tuning, worse than the above due to over-penalization from ridge-based weights.

Overall, accuracy is limited by p » n; moderate sparsity (Lasso/Elastic Net) works best here.

A. Which model was most useful for the prediction task?

The **Elastic Net** model gave the best prediction quality overall.
Its test RMSE was slightly lower than Ridge and Lasso, and it stayed stable across different alpha values.
Ridge regression was also reliable but too dense (every variable included).
The plain Lasso worked almost as well, but was more sensitive to the tuning of lambda.
The **Adaptive Lasso** was the least useful — it either became empty or over-penalized the predictors, giving worse test results.

B. Which model was most useful for interpretability?

For interpretability, the **Lasso** (or Elastic Net with moderate alpha, like 0.5) is the best choice.
It selects a small number of relevant genes and keeps the model readable,
while Ridge includes everything and Adaptive Lasso was too unstable.
Elastic Net with alpha = 0.5 is a reasonable compromise: it keeps enough predictors for stability but remains sparse enough to interpret.