Exercise 4 (2025) — Advanced Methods for Regression and Classification

Olesia Galynskaia 12321492

2025-11-05

Loading and observing data

```
load("building.RData")
stopifnot(exists("df"), is.data.frame(df))
attributes(df)
```

```
## $names
##   [1] "y"                  "START.YEAR"         "START.QUARTER"
##   [4] "COMPLETION.YEAR"    "COMPLETION.QUARTER" "PhysFin1"
##   [7] "PhysFin2"           "PhysFin3"           "PhysFin4"
##  [10] "PhysFin5"           "PhysFin6"           "PhysFin7"
##  [13] "PhysFin8"           "Econ1"              "Econ2"
##  [16] "Econ3"              "Econ4"              "Econ5"
##  [19] "Econ6"              "Econ7"              "Econ8"
##  [22] "Econ9"              "Econ10"             "Econ11"
##  [25] "Econ12"             "Econ13"             "Econ14"
##  [28] "Econ15"             "Econ16"             "Econ17"
##  [31] "Econ18"             "Econ19"             "Econ1.lag1"
##  [34] "Econ2.lag1"         "Econ3.lag1"         "Econ4.lag1"
##  [37] "Econ5.lag1"         "Econ6.lag1"         "Econ7.lag1"
##  [40] "Econ8.lag1"         "Econ9.lag1"         "Econ10.lag1"
##  [43] "Econ11.lag1"        "Econ12.lag1"        "Econ13.lag1"
##  [46] "Econ14.lag1"        "Econ15.lag1"        "Econ16.lag1"
##  [49] "Econ17.lag1"        "Econ18.lag1"        "Econ19.lag1"
##  [52] "Econ1.lag2"         "Econ2.lag2"         "Econ3.lag2"
##  [55] "Econ4.lag2"         "Econ5.lag2"         "Econ6.lag2"
##  [58] "Econ7.lag2"         "Econ8.lag2"         "Econ9.lag2"
##  [61] "Econ10.lag2"        "Econ11.lag2"        "Econ12.lag2"
##  [64] "Econ13.lag2"        "Econ14.lag2"        "Econ15.lag2"
##  [67] "Econ16.lag2"        "Econ17.lag2"        "Econ18.lag2"
##  [70] "Econ19.lag2"        "Econ1.lag3"         "Econ2.lag3"
##  [73] "Econ3.lag3"         "Econ4.lag3"         "Econ5.lag3"
##  [76] "Econ6.lag3"         "Econ7.lag3"         "Econ8.lag3"
```

```
##  [79] "Econ9.lag3"         "Econ10.lag3"        "Econ11.lag3"
##  [82] "Econ12.lag3"        "Econ13.lag3"        "Econ14.lag3"
##  [85] "Econ15.lag3"        "Econ16.lag3"        "Econ17.lag3"
##  [88] "Econ18.lag3"        "Econ19.lag3"        "Econ1.lag4"
##  [91] "Econ2.lag4"         "Econ3.lag4"         "Econ4.lag4"
##  [94] "Econ5.lag4"         "Econ6.lag4"         "Econ7.lag4"
##  [97] "Econ8.lag4"         "Econ9.lag4"         "Econ10.lag4"
## [100] "Econ11.lag4"        "Econ12.lag4"        "Econ13.lag4"
## [103] "Econ14.lag4"        "Econ15.lag4"        "Econ16.lag4"
## [106] "Econ17.lag4"        "Econ18.lag4"        "Econ19.lag4"
##
## $class
## [1] "data.frame"
##
## $row.names
##   [1]   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18
##  [19]  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36
##  [37]  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54
##  [55]  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72
##  [73]  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90
##  [91]  91  92  93  94  95  96  97  98  99 100 101 102 103 104 105 106 107 108
## [109] 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126
## [127] 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144
## [145] 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161 162
## [163] 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179 180
## [181] 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198
## [199] 199 200 201 202 203 204 205 206 207 208 209 210 211 212 213 214 215 216
## [217] 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233 234
## [235] 235 236 237 238 239 240 241 242 243 244 245 246 247 248 249 250 251 252
## [253] 253 254 255 256 257 258 259 260 261 262 263 264 265 266 267 268 269 270
## [271] 271 272 273 274 275 276 277 278 279 280 281 282 283 284 285 286 287 288
## [289] 289 290 291 292 293 294 295 296 297 298 299 300 301 302 303 304 305 306
## [307] 307 308 309 310 311 312 313 314 315 316 317 318 319 320 321 322 323 324
## [325] 325 326 327 328 329 330 331 332 333 334 335 336 337 338 339 340 341 342
## [343] 343 344 345 346 347 348 349 350 351 352 353 354 355 356 357 358 359 360
## [361] 361 362 363 364 365 366 367 368 369 370 371 372
```

```
str(attributes(df))
```

```
## List of 3
##  $ names    : chr [1:108] "y" "START.YEAR" "START.QUARTER" "COMPLETION.YEAR" ...
##  $ class    : chr "data.frame"
##  $ row.names: int [1:372] 1 2 3 4 5 6 7 8 9 10 ...
```

```
head(sapply(df, function(x) attr(x, "label")), 100)
```

```
## $y
```

```
## NULL
##
## $START.YEAR
## NULL
##
## $START.QUARTER
## NULL
##
## $COMPLETION.YEAR
## NULL
##
## $COMPLETION.QUARTER
## NULL
##
## $PhysFin1
## NULL
##
## $PhysFin2
## NULL
##
## $PhysFin3
## NULL
##
## $PhysFin4
## NULL
##
## $PhysFin5
## NULL
##
## $PhysFin6
## NULL
##
## $PhysFin7
## NULL
##
## $PhysFin8
## NULL
##
## $Econ1
## NULL
##
## $Econ2
## NULL
##
## $Econ3
## NULL
##
## $Econ4
```

```
## NULL
##
## $Econ5
## NULL
##
## $Econ6
## NULL
##
## $Econ7
## NULL
##
## $Econ8
## NULL
##
## $Econ9
## NULL
##
## $Econ10
## NULL
##
## $Econ11
## NULL
##
## $Econ12
## NULL
##
## $Econ13
## NULL
##
## $Econ14
## NULL
##
## $Econ15
## NULL
##
## $Econ16
## NULL
##
## $Econ17
## NULL
##
## $Econ18
## NULL
##
## $Econ19
## NULL
##
## $Econ1.lag1
```

```
## NULL
##
## $Econ2.lag1
## NULL
##
## $Econ3.lag1
## NULL
##
## $Econ4.lag1
## NULL
##
## $Econ5.lag1
## NULL
##
## $Econ6.lag1
## NULL
##
## $Econ7.lag1
## NULL
##
## $Econ8.lag1
## NULL
##
## $Econ9.lag1
## NULL
##
## $Econ10.lag1
## NULL
##
## $Econ11.lag1
## NULL
##
## $Econ12.lag1
## NULL
##
## $Econ13.lag1
## NULL
##
## $Econ14.lag1
## NULL
##
## $Econ15.lag1
## NULL
##
## $Econ16.lag1
## NULL
##
## $Econ17.lag1
```

```
## NULL
##
## $Econ18.lag1
## NULL
##
## $Econ19.lag1
## NULL
##
## $Econ1.lag2
## NULL
##
## $Econ2.lag2
## NULL
##
## $Econ3.lag2
## NULL
##
## $Econ4.lag2
## NULL
##
## $Econ5.lag2
## NULL
##
## $Econ6.lag2
## NULL
##
## $Econ7.lag2
## NULL
##
## $Econ8.lag2
## NULL
##
## $Econ9.lag2
## NULL
##
## $Econ10.lag2
## NULL
##
## $Econ11.lag2
## NULL
##
## $Econ12.lag2
## NULL
##
## $Econ13.lag2
## NULL
##
## $Econ14.lag2
```

```
## NULL
##
## $Econ15.lag2
## NULL
##
## $Econ16.lag2
## NULL
##
## $Econ17.lag2
## NULL
##
## $Econ18.lag2
## NULL
##
## $Econ19.lag2
## NULL
##
## $Econ1.lag3
## NULL
##
## $Econ2.lag3
## NULL
##
## $Econ3.lag3
## NULL
##
## $Econ4.lag3
## NULL
##
## $Econ5.lag3
## NULL
##
## $Econ6.lag3
## NULL
##
## $Econ7.lag3
## NULL
##
## $Econ8.lag3
## NULL
##
## $Econ9.lag3
## NULL
##
## $Econ10.lag3
## NULL
##
## $Econ11.lag3
```

```
## NULL
##
## $Econ12.lag3
## NULL
##
## $Econ13.lag3
## NULL
##
## $Econ14.lag3
## NULL
##
## $Econ15.lag3
## NULL
##
## $Econ16.lag3
## NULL
##
## $Econ17.lag3
## NULL
##
## $Econ18.lag3
## NULL
##
## $Econ19.lag3
## NULL
##
## $Econ1.lag4
## NULL
##
## $Econ2.lag4
## NULL
##
## $Econ3.lag4
## NULL
##
## $Econ4.lag4
## NULL
##
## $Econ5.lag4
## NULL
##
## $Econ6.lag4
## NULL
##
## $Econ7.lag4
## NULL
##
## $Econ8.lag4
```

```
## NULL
##
## $Econ9.lag4
## NULL
##
## $Econ10.lag4
## NULL
##
## $Econ11.lag4
## NULL
```

Ex-1

Train/test split

```r
df <- df[order(1:nrow(df)), ]    # already in time order, this is a no-op

n <- nrow(df)
cut <- floor(0.8 * n)

train <- df[1:cut, ]
test  <- df[(cut+1):n, ]
```

Comment

If we split the data randomly, we would mix future observations into the training set.
That would give the model information about the future that it would not have in reality.
In other words, the model would "cheat" and we would get over-optimistic results.

To avoid this leakage, we use a time-based split:

the earlier 80% of the observations are used for training

the last 20% are used as the test set

This way, the model only learns from the past and we evaluate it on the future, which matches the real-world forecasting scenario.

Ex-2

(a) PLS with 10-fold CV, time-aware segments

```r
library(pls)

set.seed(12321492)  # reproducibility for CV folds
```

```r
# 1) Keep only numeric variables; y must be numeric
is_num <- vapply(train, is.numeric, logical(1))
stopifnot("y" %in% names(train), is_num["y"])
train_num <- train[, is_num, drop = FALSE]

# 2) Drop zero-variance columns (defensive)
nzv <- vapply(train_num, function(x) sd(x, na.rm = TRUE) > 0, logical(1))
train_num <- train_num[, nzv, drop = FALSE]

# 3) Fit PLS with consecutive segments to respect time
pls_fit <- plsr(
  y ~ .,
  data         = train_num,
  scale        = TRUE,
  validation   = "CV",
  segments     = 10,
  segment.type = "consecutive"
)

# 4) CV diagnostics
summary(pls_fit)
```

```
## Data:    X dimension: 297 107
##  Y dimension: 297 1
## Fit method: kernelpls
## Number of components considered: 107
##
## VALIDATION: RMSEP
## Cross-validated using 10 consecutive segments.
##        (Intercept)  1 comps  2 comps  3 comps  4 comps  5 comps  6 comps
## CV          0.8735   0.5668   0.3908   0.3381   0.3144   0.2845   0.2823
## adjCV       0.8735   0.5653   0.3892   0.3362   0.3115   0.2816   0.2790
##        7 comps  8 comps  9 comps  10 comps  11 comps  12 comps  13 comps
## CV      0.2859   0.3018   0.3042    0.3041    0.3062    0.3068    0.3053
## adjCV   0.2827   0.2975   0.2993    0.2987    0.3007    0.3012    0.2996
##        14 comps  15 comps  16 comps  17 comps  18 comps  19 comps  20 comps
## CV       0.3076    0.3104    0.3139    0.3157    0.3182    0.3194    0.3214
## adjCV    0.3017    0.3044    0.3075    0.3092    0.3113    0.3125    0.3143
##        21 comps  22 comps  23 comps  24 comps  25 comps  26 comps  27 comps
## CV       0.3218    0.3217    0.3193    0.3188    0.3164    0.3154    0.3165
## adjCV    0.3146    0.3145    0.3121    0.3116    0.3094    0.3085    0.3095
##        28 comps  29 comps  30 comps  31 comps  32 comps  33 comps  34 comps
## CV       0.3180    0.3184    0.3171    0.3167    0.3179    0.3174    0.3183
## adjCV    0.3108    0.3112    0.3101    0.3096    0.3107    0.3102    0.3110
##        35 comps  36 comps  37 comps  38 comps  39 comps  40 comps  41 comps
## CV       0.3174    0.3182    0.3199    0.3209    0.3224    0.3219    0.3218
## adjCV    0.3102    0.3109    0.3125    0.3134    0.3147    0.3143    0.3142
```

```
##        42 comps  43 comps  44 comps  45 comps  46 comps  47 comps  48 comps
## CV       0.3214    0.3213    0.3217    0.3215    0.3221    0.3218    0.3213
## adjCV    0.3138    0.3138    0.3141    0.3139    0.3145    0.3143    0.3138
##        49 comps  50 comps  51 comps  52 comps  53 comps  54 comps  55 comps
## CV       0.3214    0.3216    0.3217    0.3217    0.3216    0.3212    0.3216
## adjCV    0.3139    0.3140    0.3141    0.3141    0.3141    0.3136    0.3140
##        56 comps  57 comps  58 comps  59 comps  60 comps  61 comps  62 comps
## CV       0.3218    0.3218    0.3214    0.3213    0.3210    0.3213    0.3211
## adjCV    0.3142    0.3142    0.3139    0.3137    0.3135    0.3137    0.3135
##        63 comps  64 comps  65 comps  66 comps  67 comps  68 comps  69 comps
## CV       0.3208    0.3208    0.3207    0.3207    0.3208    0.3211    0.3211
## adjCV    0.3133    0.3132    0.3132    0.3132    0.3133    0.3135    0.3136
##        70 comps  71 comps  72 comps  73 comps  74 comps  75 comps  76 comps
## CV       0.3211    0.3212   8886609   8895189  12354349  12354386  12354375
## adjCV    0.3135    0.3136   8425845   8433980  11714012  11714047  11714037
##        77 comps  78 comps  79 comps  80 comps  81 comps  82 comps  83 comps
## CV     12354381  12354374  12354381  12354375  12354378  12354376  12354378
## adjCV  11714043  11714036  11714043  11714038  11714040  11714038  11714040
##        84 comps  85 comps  86 comps  87 comps  88 comps  89 comps  90 comps
## CV     12354380  12354382  12354379  12354376  12354381  12354379  12354375
## adjCV  11714042  11714043  11714041  11714038  11714043  11714041  11714037
##        91 comps  92 comps  93 comps  94 comps  95 comps  96 comps  97 comps
## CV     12354373  12354377  12354377  12354377  12354377  12354375  12354387
## adjCV  11714036  11714039  11714039  11714039  11714040  11714037  11714049
##        98 comps  99 comps  100 comps  101 comps  102 comps  103 comps
## CV     12354385  12354380   12354381   12354381   12354387   12354384
## adjCV  11714046  11714042   11714043   11714043   11714049   11714046
##        104 comps  105 comps  106 comps  107 comps
## CV      12354382   12354387   12354374   12354381
## adjCV   11714044   11714049   11714036   11714043
##
## TRAINING: % variance explained
##    1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps  8 comps
## X    64.92    69.77    73.87    78.03    80.63    81.88    84.96    87.65
## y    62.28    83.21    88.00    90.60    92.43    93.20    93.40    93.56
##    9 comps  10 comps  11 comps  12 comps  13 comps  14 comps  15 comps
## X    89.56     91.05     91.98     93.00     93.59     94.17     94.75
## y    93.77     93.94     94.05     94.15     94.31     94.36     94.40
##    16 comps  17 comps  18 comps  19 comps  20 comps  21 comps  22 comps
## X     95.23     95.97     96.33     97.00     97.48     97.80     98.01
## y     94.44     94.48     94.56     94.58     94.61     94.66     94.72
##    23 comps  24 comps  25 comps  26 comps  27 comps  28 comps  29 comps
## X     98.14     98.31     98.47     98.65     98.80     98.92     99.02
## y     94.79     94.83     94.86     94.89     94.93     94.97     95.00
##    30 comps  31 comps  32 comps  33 comps  34 comps  35 comps  36 comps
## X     99.17     99.23     99.29     99.38     99.44     99.48     99.52
## y     95.01     95.04     95.06     95.08     95.10     95.11     95.12
##    37 comps  38 comps  39 comps  40 comps  41 comps  42 comps  43 comps
```

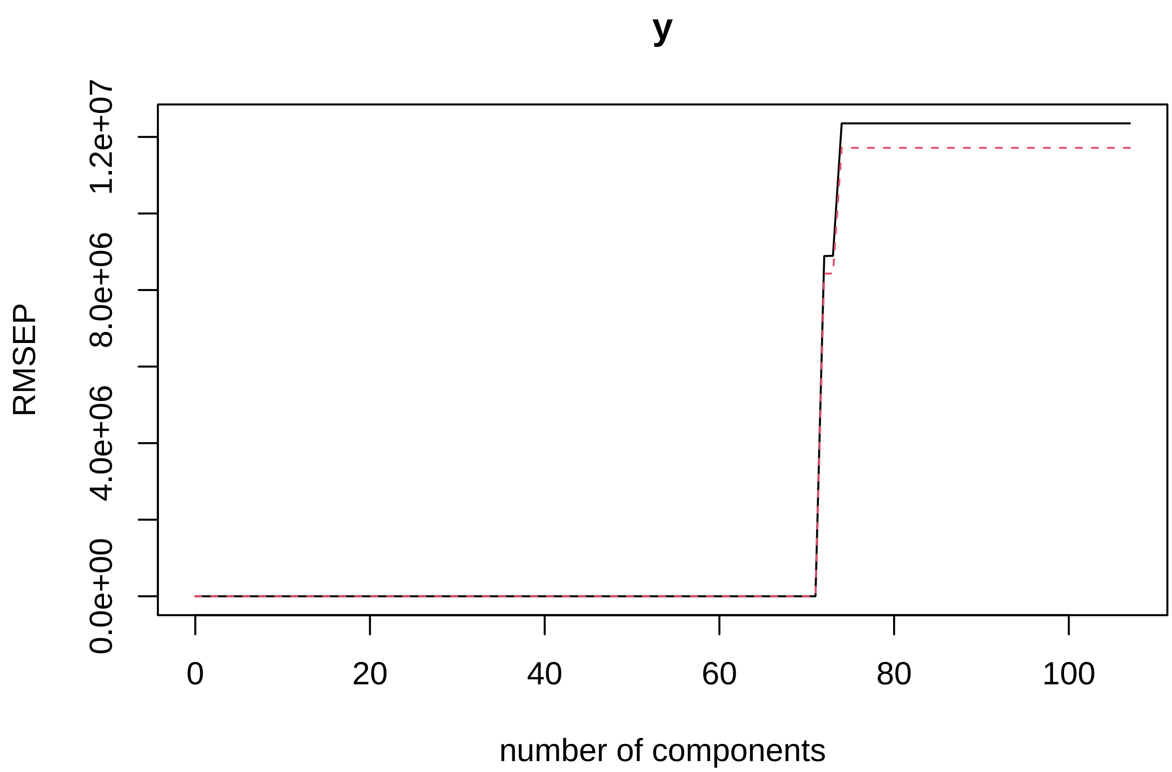
```
## X     99.56     99.63     99.66     99.67     99.70     99.72     99.74
## y     95.13     95.14     95.15     95.16     95.17     95.17     95.18
##    44 comps  45 comps  46 comps  47 comps  48 comps  49 comps  50 comps
## X     99.77     99.79     99.81     99.83     99.85     99.87     99.88
## y     95.18     95.18     95.18     95.19     95.19     95.19     95.19
##    51 comps  52 comps  53 comps  54 comps  55 comps  56 comps  57 comps
## X     99.89     99.91     99.92     99.94     99.95     99.95     99.96
## y     95.19     95.20     95.20     95.20     95.20     95.20     95.20
##    58 comps  59 comps  60 comps  61 comps  62 comps  63 comps  64 comps
## X     99.97     99.97     99.98     99.98     99.98     99.99     99.99
## y     95.20     95.20     95.20     95.20     95.20     95.20     95.20
##    65 comps  66 comps  67 comps  68 comps  69 comps  70 comps  71 comps
## X     99.99     99.99     100.0     100.0     100.0     100.0     100.0
## y     95.20     95.20      95.2      95.2      95.2      95.2      95.2
##    72 comps  73 comps  74 comps  75 comps  76 comps  77 comps  78 comps
## X     100.0     100.0     100.0     100.0     100.0     100.0     100.0
## y      95.2      95.2      95.2      95.2      95.2      95.2      95.2
##    79 comps  80 comps  81 comps  82 comps  83 comps  84 comps  85 comps
## X     100.0     100.0     100.0     100.0     100.0     100.0     100.0
## y      95.2      95.2      95.2      95.2      95.2      95.2      95.2
##    86 comps  87 comps  88 comps  89 comps  90 comps  91 comps  92 comps
## X     100.0     100.0     100.0     100.0     100.0     100.0     100.0
## y      95.2      95.2      95.2      95.2      95.2      95.2      95.2
##    93 comps  94 comps  95 comps  96 comps  97 comps  98 comps  99 comps
## X     100.0     100.0     100.0     100.0     100.0     100.0     100.0
## y      95.2      95.2      95.2      95.2      95.2      95.2      95.2
##    100 comps  101 comps  102 comps  103 comps  104 comps  105 comps  106 comps
## X      100.0      100.0      100.0      100.0      100.0      100.0      100.0
## y       95.2       95.2       95.2       95.2       95.2       95.2       95.2
##    107 comps
## X      100.0
## y       95.2
```

```r
cv_res <- RMSEP(pls_fit)    # CV errors for all components (incl. intercept)
cv_mat <- cv_res$val        # 3D array: [metric, response, component]

# Robust extraction: use indices, not names (names may differ or be NULL)
#  - 1st dim index 1 = "CV" RMSE
#  - 2nd dim index 1 = first/only response
#  - 3rd dim = components (slot 1 is intercept-only)
rmse_all  <- drop(cv_mat[1, 1, ])     # includes intercept
rmse_vals <- rmse_all[-1]             # drop intercept-only
ncomp_seq <- seq_along(rmse_vals)

# Choose optimal number of components by min CV-RMSE
opt_ncomp <- which.min(rmse_vals)
cat("Optimal ncomp by min CV-RMSE:", opt_ncomp, "\n")
```

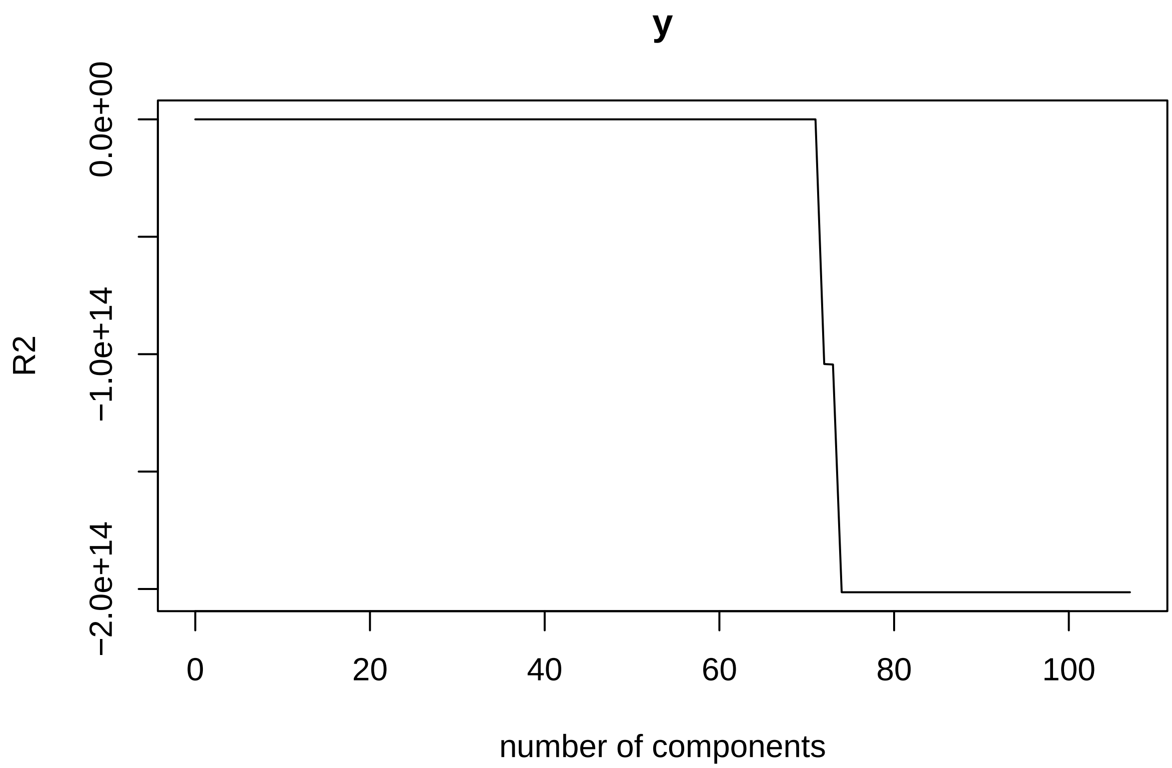
```
## Optimal ncomp by min CV-RMSE: 6
```

(b) Plotting and choosing numbers of components

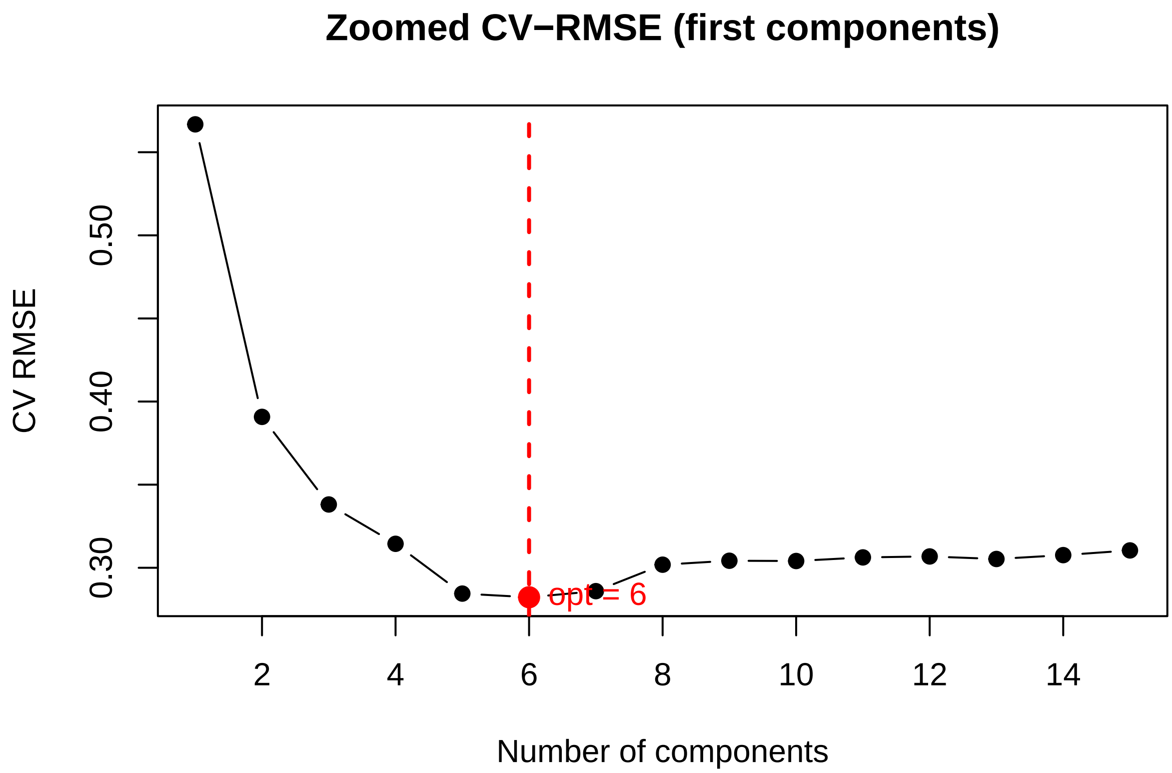
```r
validationplot(pls_fit, val.type = "RMSEP")  # full CV-RMSE curve
```




```r
validationplot(pls_fit, val.type = "R2")     # CV-R^2 curve
```




```r
# Zoom into first 15 components for readability
upper <- min(15, length(rmse_vals))
plot(ncomp_seq[1:upper], rmse_vals[1:upper], type = "b", pch = 19,
     xlab = "Number of components", ylab = "CV RMSE",
     main = "Zoomed CV-RMSE (first components)")
abline(v = opt_ncomp, col = "red", lwd = 2, lty = 2)
points(opt_ncomp, rmse_vals[opt_ncomp], col = "red", pch = 19, cex = 1.4)
text(opt_ncomp, rmse_vals[opt_ncomp], paste0("opt = ", opt_ncomp),
     pos = 4, col = "red")
```



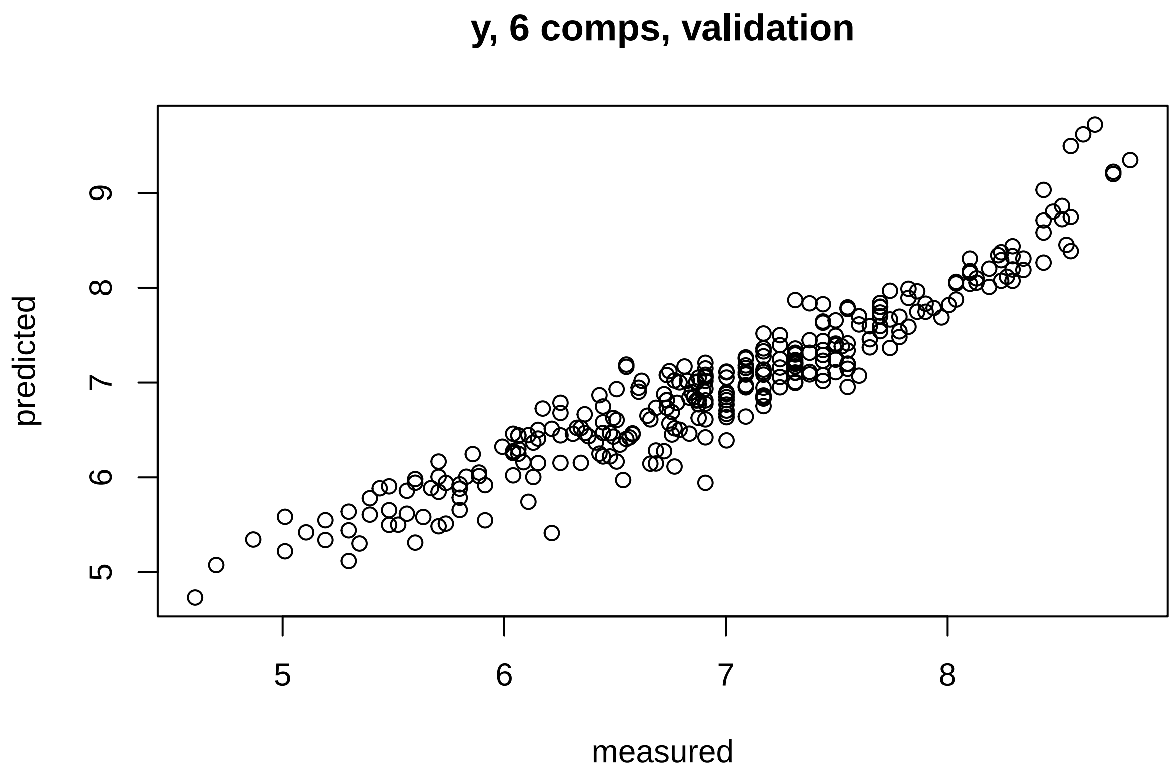
Comment

The cross-validated RMSE decreases rapidly as we add the first few components and reaches its minimum at 6 components.
Beyond this point, the error starts to increase again, indicating the beginning of overfitting.

Therefore, the optimal number of PLS components is 6.

(c) Predplot

```r
predplot(pls_fit, ncomp = opt_ncomp)
```



Comment

The predplot() shows the cross-validated predicted values versus the measured values using 6 components.

Most points lie close to the diagonal, indicating that the model captures the main structure in the data.
There is some scatter, especially at extreme values, but overall the cross-validated fit looks reasonable.

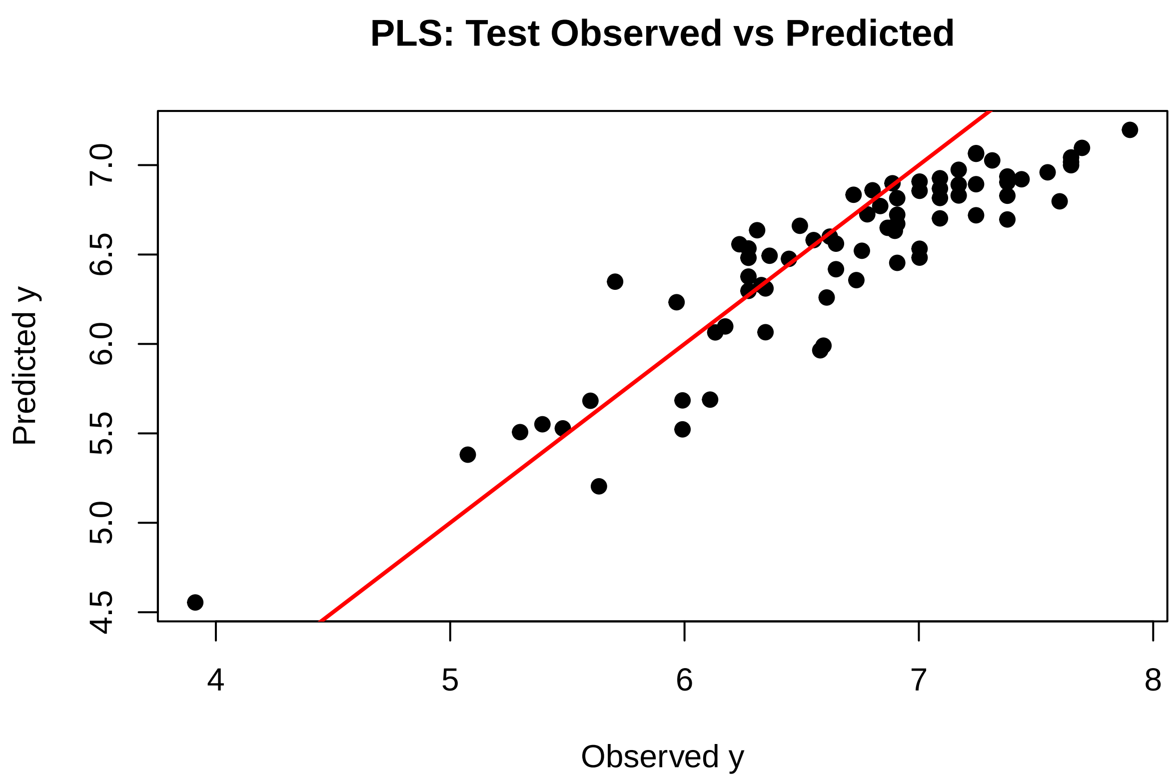
(d) Predicted versus observed values

```
# Prepare test numeric data
test_num <- test[, is_num, drop = FALSE]

# Predict
y_pred <- predict(pls_fit, newdata = test_num, ncomp = opt_ncomp)
y_true <- test_num$y

# RMSE
```

```r
rmse <- sqrt(mean((y_true - y_pred)^2))
rmse
```

```
## [1] 0.3638673
```

```r
plot(y_true, y_pred,
     xlab = "Observed y",
     ylab = "Predicted y",
     pch = 19,
     main = "PLS: Test Observed vs Predicted")
abline(0, 1, col = "red", lwd = 2)
```



Comment

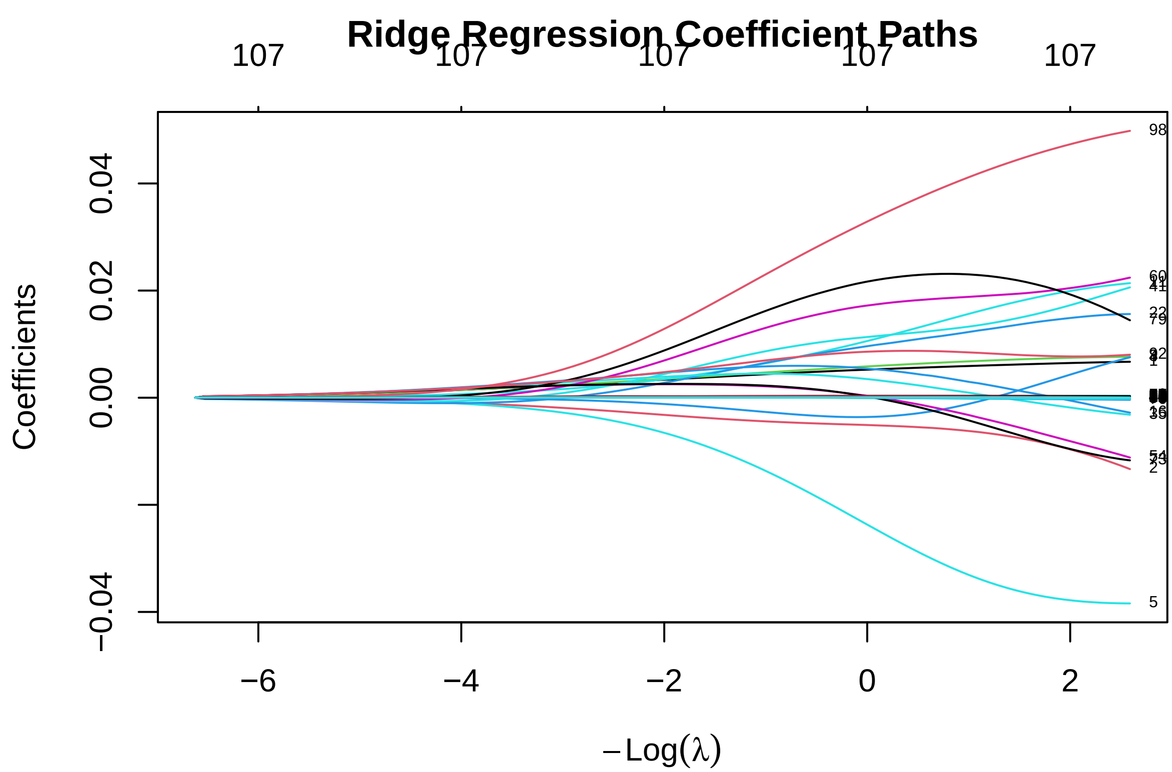
The PLS model achieves a test RMSE of approximately 0.36, which is higher than the RMSE obtained in the previous exercise (about 0.25). This indicates that, for this dataset, PLS does not outperform the simpler linear model.
A likely reason is that the dataset already has a structured set of lagged predictors and does not suffer heavily from multicollinearity issues, so the advantage of PLS is limited in this case.

The test scatter plot confirms that PLS predictions generally follow the observed values but exhibit more deviation from the diagonal compared to the previous model.

Ex-3

(a) Ridge Regression

```
library(glmnet)

# prepare X and y matrices for glmnet
X_train <- as.matrix(train_num[, setdiff(names(train_num), "y")])
y_train <- train_num$y

# Fit ridge regression (alpha = 0)
ridge_fit <- glmnet(
  X_train, y_train,
  alpha = 0,              # ridge penalty
  standardize = TRUE      # glmnet scales by default
)

# plot the coefficient paths vs lambda
plot(ridge_fit, xvar = "lambda", label = TRUE,
     main = "Ridge Regression Coefficient Paths")
```



Comment

- What does the plot show?

It shows the coefficient paths for ridge regression as the penalty parameter lambda varies.
For large lambda, the coefficients are shrunk strongly towards zero.
As lambda decreases, the coefficients gradually increase in magnitude, approaching the ordinary least squares solution.

- Which default parameters are used for lambda?

glmnet() uses a default logarithmic grid of lambda values (around 100 values), automatically chosen based on data scale.
It starts from a lambda large enough to shrink all coefficients almost to zero and decreases to a small value near the unregularized model.

- What is the meaning of alpha?

alpha = 0 means ridge regression (L2 penalty). Values:

alpha = 0 → Ridge (L2)

alpha = 1 → LASSO (L1)

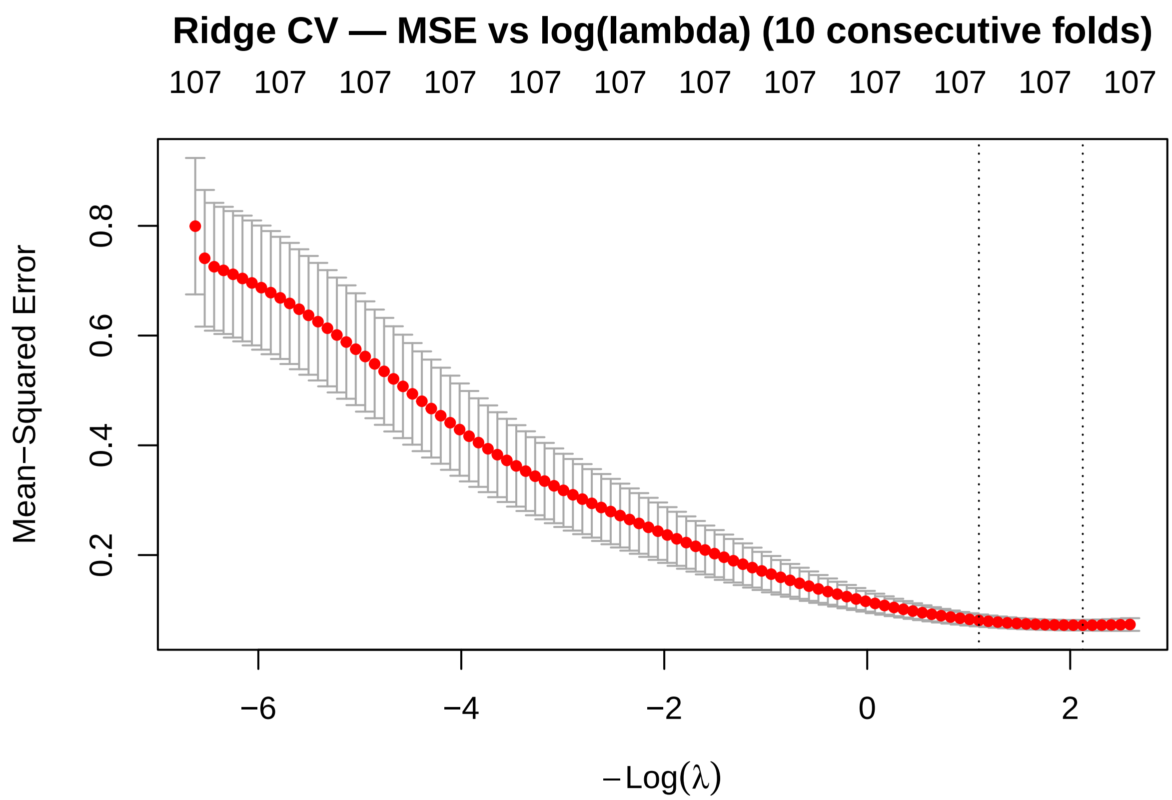
0 < alpha < 1 → Elastic Net

Ridge shrinks coefficients but does not set them exactly to zero.

(b) Ridge Regression

```r
X_train <- as.matrix(train_num[, setdiff(names(train_num), "y")])
y_train <- train_num$y

K <- 10
n_tr <- nrow(X_train)
fold_sizes <- rep(floor(n_tr / K), K)
fold_sizes[1:(n_tr %% K)] <- fold_sizes[1:(n_tr %% K)] + 1
foldid <- inverse.rle(list(lengths = fold_sizes, values = seq_len(K)))

# CV ridge (alpha=0)
set.seed(12321492)
cv_ridge <- cv.glmnet(
  X_train, y_train,
  alpha = 0,                # ridge
  standardize = TRUE,       # glmnet scales by default
  nfolds = K,               # will be overridden by foldid, but keep for clarity
  foldid = foldid,          # consecutive blocks, no time leakage
  type.measure = "mse",     # for Gaussian -> MSE; we'll sqrt later for RMSE
  family = "gaussian"
)
```

```r
# Plot CV curve
plot(cv_ridge)
title("Ridge CV - MSE vs log(lambda) (10 consecutive folds)", line = 2.5)
```




```r
# Pick lambdas and RMSEs
lam_min <- cv_ridge$lambda.min       # minimizes mean CV error
lam_1se <- cv_ridge$lambda.1se       # 1-SE rule (simpler model)

mse_min <- min(cv_ridge$cvm)         # mean CV MSE at lambda.min
rmse_min <- sqrt(mse_min)

# Also compute RMSE at lambda.1se for reporting
mse_1se <- cv_ridge$cvm[which(cv_ridge$lambda == lam_1se)]
rmse_1se <- sqrt(mse_1se)

cat(sprintf("lambda.min = %.5g; CV RMSE (min) = %.4f\n", lam_min, rmse_min))
```

```
## lambda.min = 0.11958; CV RMSE (min) = 0.2685
```

```r
cat(sprintf("lambda.1se = %.5g; CV RMSE (1SE) = %.4f\n", lam_1se, rmse_1se))
```

```
## lambda.1se = 0.33274; CV RMSE (1SE) = 0.2844
```

```r
# Coefficients at lambda.min (optional preview)
coef_min <- coef(cv_ridge, s = "lambda.min")
nnz <- sum(coef_min != 0)
cat("Nonzero coefficients at lambda.min:", nnz, "\n")
```

```
## Nonzero coefficients at lambda.min: 108
```

```r
head(as.matrix(coef_min)[,1], 15)
```

```
##        (Intercept)          START.YEAR       START.QUARTER     COMPLETION.YEAR
##       2.308854e+00        6.532715e-03       -1.032039e-02        7.504879e-03
## COMPLETION.QUARTER           PhysFin1            PhysFin2            PhysFin3
##       5.019807e-03       -3.805561e-02        2.198618e-05        2.954967e-05
##           PhysFin4            PhysFin5            PhysFin6            PhysFin7
##      -5.116027e-05        3.325872e-05        1.494950e-04        2.030161e-02
##           PhysFin8               Econ1               Econ2
##       2.682164e-04        8.919271e-06        8.630951e-05
```

Comment

We fitted ridge regression with cv.glmnet using 10 consecutive folds to respect time order.

The CV curve decreases smoothly as lambda decreases and reaches its minimum at lambda_min 0.12, with CV RMSE 0.27.

As a more conservative choice, the 1-SE rule selects lambda_1se 0.33 with a slightly higher CV RMSE (0.28) and stronger shrinkage.

The corresponding ridge coefficients are obtained via coef(cv_ridge, s = "lambda.min") (or "lambda.1se"). Ridge shrinks magnitudes but does not set coefficients exactly to zero.

(c) Ridge — test predictions, plots, RMSE

```r
stopifnot(exists("cv_ridge"))

# Ensure test has the same numeric columns as training (after our NZV drop)
test_num <- test[, colnames(train_num), drop = FALSE]

# Matrices for glmnet
X_test <- as.matrix(test_num[, setdiff(colnames(test_num), "y")])
y_true <- test_num$y

# Predict with lambda.min and lambda.1se
y_pred_min  <- drop(predict(cv_ridge, newx = X_test, s = "lambda.min"))
y_pred_1se  <- drop(predict(cv_ridge, newx = X_test, s = "lambda.1se"))
```

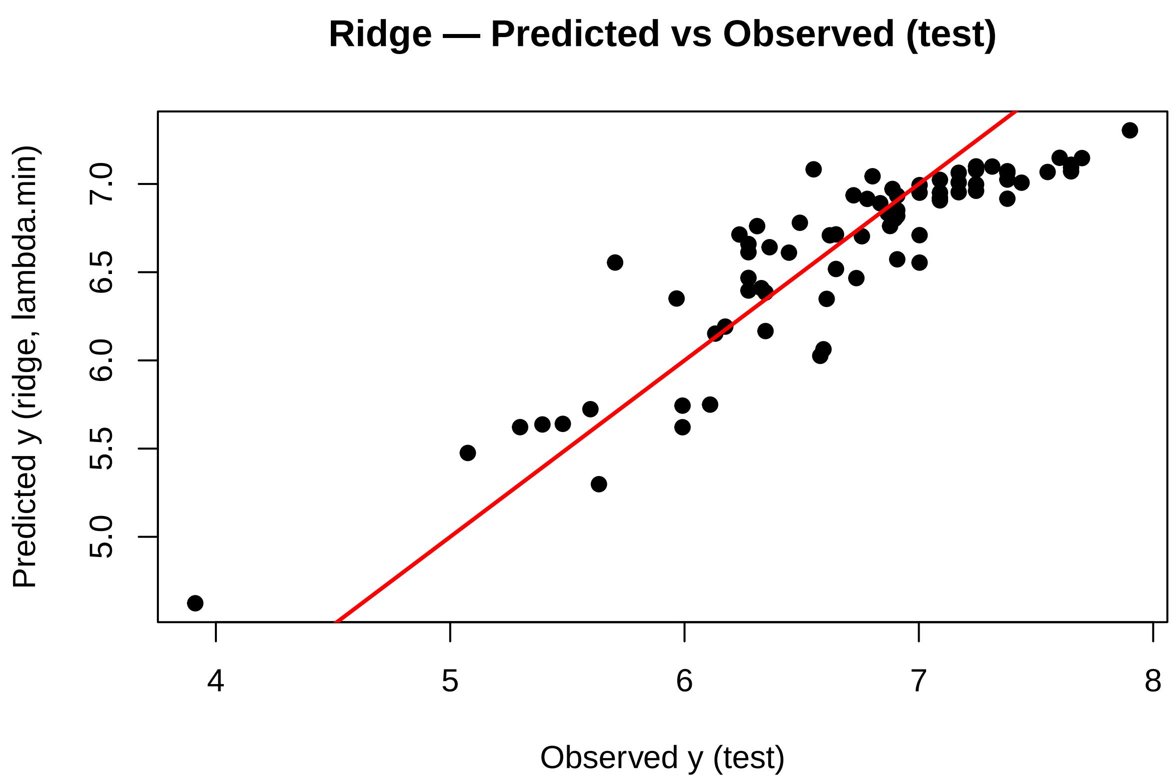
```r
# RMSE helper
rmse <- function(y, yp) sqrt(mean((y - yp)^2))

rmse_min <- rmse(y_true, y_pred_min)
rmse_1se <- rmse(y_true, y_pred_1se)

cat(sprintf("Test RMSE (ridge, lambda.min):  %.4f\n", rmse_min))
```

```
## Test RMSE (ridge, lambda.min):  0.3251
```

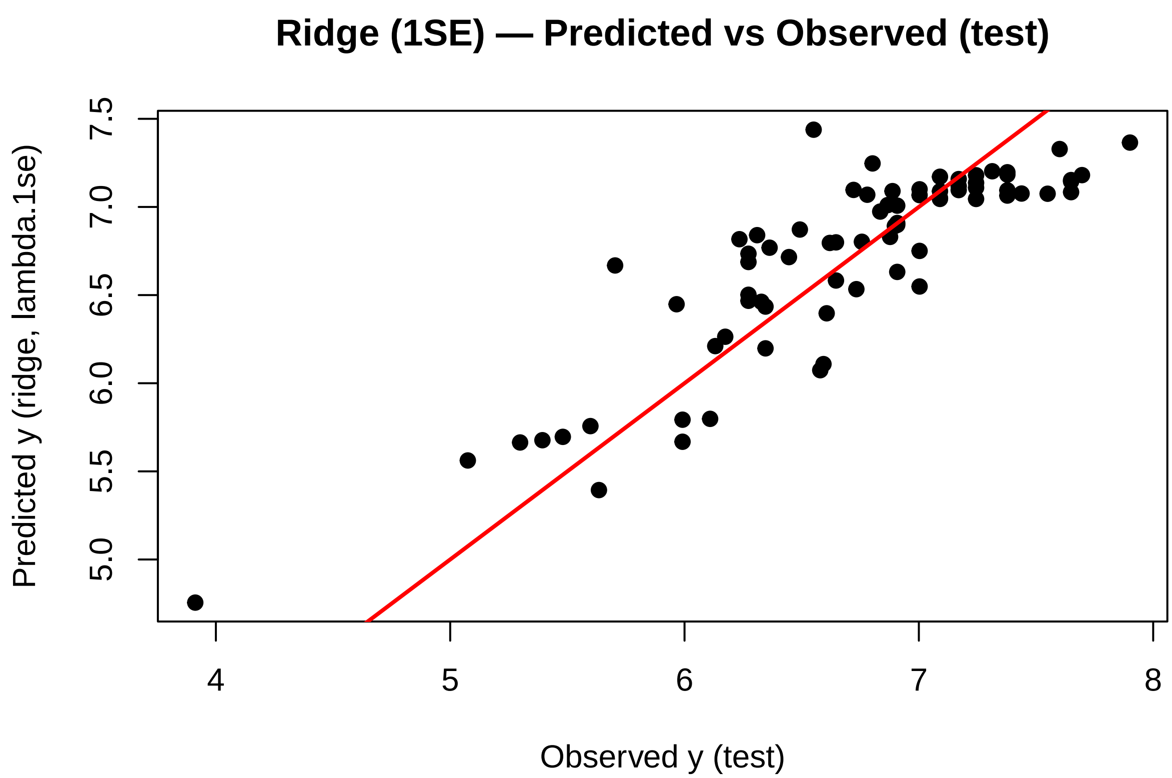
```r
cat(sprintf("Test RMSE (ridge, lambda.1se):  %.4f\n", rmse_1se))
```

```
## Test RMSE (ridge, lambda.1se):  0.3404
```

```r
# Plot: observed vs predicted (lambda.min)
plot(y_true, y_pred_min,
     xlab = "Observed y (test)",
     ylab = "Predicted y (ridge, lambda.min)",
     pch = 19, main = "Ridge - Predicted vs Observed (test)")
abline(0, 1, col = "red", lwd = 2)
```



```r
# Second plot for 1SE
plot(y_true, y_pred_1se,
     xlab = "Observed y (test)",
     ylab = "Predicted y (ridge, lambda.1se)",
     pch = 19, main = "Ridge (1SE) - Predicted vs Observed (test)")
abline(0, 1, col = "red", lwd = 2)
```



Comment

Using the optimal ridge model, we predicted the response on the test set and compared the results against the observed values.
The scatter plot shows a clear positive linear relationship, with most points lying close to the 45-degree line, indicating good predictive performance.

The test RMSE for the model with lambda_min is approximately 0.33, and about 0.34 under the 1-SE rule.
Compared to the previous models, ridge performs better than PLS (0.36) but does not outperform the standard linear regression model from Exercise 3 (0.25).

This is consistent with the idea that ridge improves stability through coefficient shrinkage, but may not produce the lowest error when multicollinearity is moderate and the baseline model already fits well.

(d)

Comment

Even though ridge penalization shrinks the overall L2-norm of the coefficient vector as lambda increases, individual coefficients do not have to move strictly in one direction.
With correlated predictors, the coefficients influence each other, so some paths can bend slightly rather than decrease smoothly.

In addition, glmnet computes the solution path numerically, which can introduce small non-monotonic wiggles.
In other words, the total shrinkage is monotonic, but individual coefficient curves may show small reversals, and this is expected behavior rather than a problem.