Lab 3 RM

Viktor Sjoberg

2023-11-28

```
library(readr)
library(ggplot2)
library(gt)
library(boot)
library(glmnet)
```

```
## Loading required package: Matrix
```

```
## Loaded glmnet 4.1-8
```

```
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
```

```
library(tidyr)
```

```
##
## Attaching package: 'tidyr'
```

```
## The following objects are masked from 'package:Matrix':
##
##     expand, pack, unpack
```

```
setwd("/Users/viktorsjoberg/Desktop/high-dimensional/Assignment 3")
data <-  load('Lab3.Rdata')
```

Task 1

a)

```
standardize_gene_expression <- function(data) {
  # Apply the standardization function to each column (gene)
  standardized_data <- apply(data, 2, function(x) {
    mean_x <- mean(x)
    sd_x <- sd(x)
    (x - mean_x) / sd_x + mean_x
```

```
  })
  return(as.data.frame(standardized_data))
}

xx_standardized <- standardize_gene_expression(xx)
```

b)

```
xx_adjusted <- xx_standardized - 10
```

c)

We'll use the first five individuals to estimate the vector of the means. This involves using the maximum likelihood estimator and both James-Stein estimators

```
# Using first five individuals
first_five <- xx_adjusted[1:5,]
# Maximum Likelihood Estimator
mle_estimate <- apply(first_five,2,'mean')
sigma2 <-  apply(first_five,2,'sd')
m_sigma <- mean(sigma2)

# James-Stein Estimators
# Shrinkage towards zero
js_shrink_zero <- function(estimate) {
  shrinkage_factor <- (1 - (p - 2) / sum((estimate^2))*m_sigma/5)*mle_estimate
  return(shrinkage_factor)
}
# James-Stein Estimator - Shrinkage towards the common mean
p <- 300
common_mean <- (p-3)/(p-1)*m_sigma/5/var(mle_estimate)
js_shrink_common <- function(estimate) {
  shrinkage_factor <- (1 - common_mean)*mle_estimate + common_mean*mean(mle_estimate)
  return(shrinkage_factor)
}

# Apply the James-Stein estimators
js_zero_estimate <- js_shrink_zero(mle_estimate)
js_common_estimate <- js_shrink_common(mle_estimate)

# Now the lengths should match
length(js_zero_estimate) # Should be equal to the number of genes
```

```
## [1] 300
```

```
length(js_common_estimate) # Should be equal to the number of genes
```

```
## [1] 300
```

In this case we know that sigma is equal to one however in a real scenario we often dont know what sigma is so i have estimeted it whit the first 5 rows. The first five rows will thus be a smaple and the data set will be seen as the population.

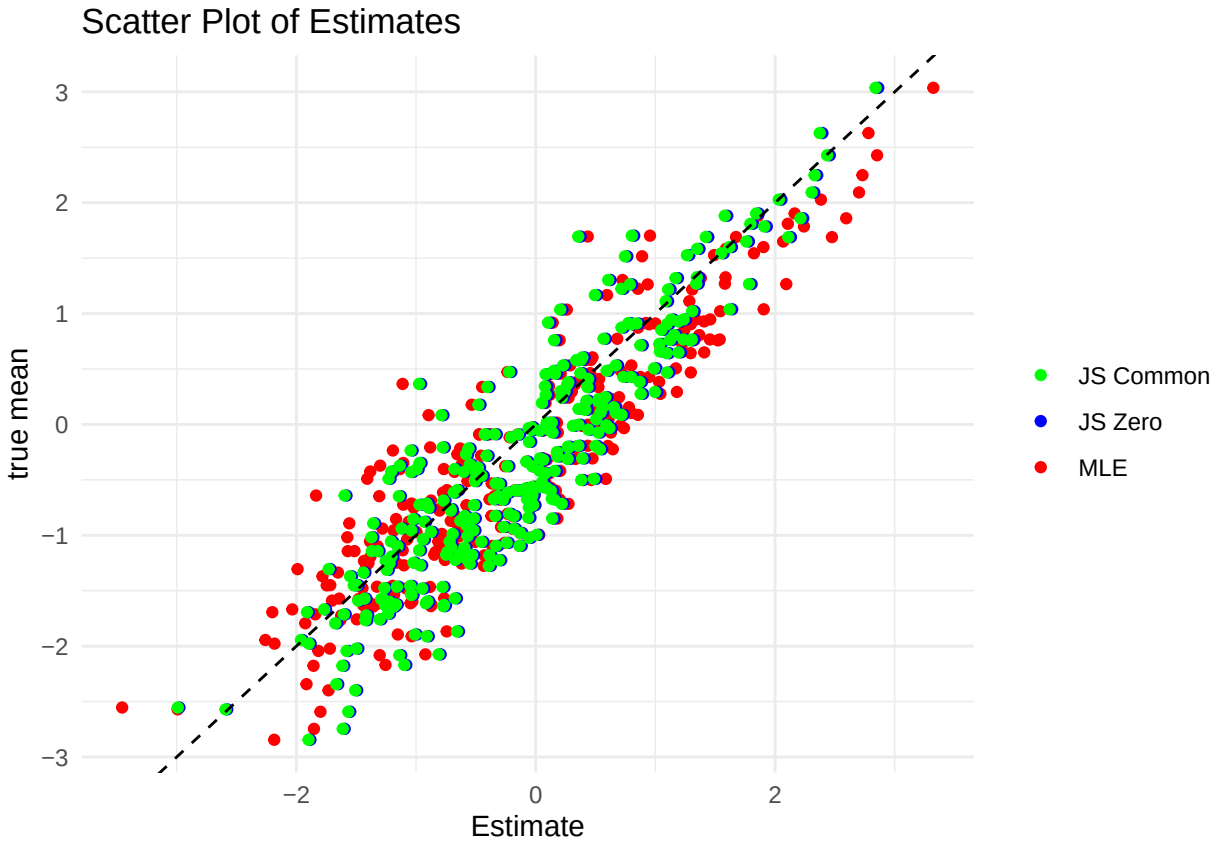
d)

i)

```
xx_rest <- xx_adjusted[6:nrow(xx_adjusted),]
average_rest <- colMeans(xx_rest)


# Create a data frame for ggplot
data_to_plot <- data.frame(
  Average_Expression = rep(average_rest, times = 3),
  Estimate = c(mle_estimate, js_zero_estimate, js_common_estimate),
  Method = factor(rep(c("MLE", "JS Zero", "JS Common"), each = length(average_rest)))
)

# Plot with ggplot
ggplot(data_to_plot, aes(x = Estimate, y = Average_Expression, color = Method)) +
  geom_point() +
  geom_abline(intercept = 0, slope = 1, linetype = "dashed", color = "black") +
  labs(x = "Estimate", y = "true mean", title = "Scatter Plot of Estimates") +
  scale_color_manual(values = c("MLE" = "red", "JS Zero" = "blue", "JS Common" = "green")) +
  theme_minimal() +
  theme(legend.title = element_blank())
```



Since the estimeted sigma is close to one it does not change the outcome that much.

ii)

```r
# Function to calculate squared error
squared_error <- function(estimate, true_value) {
  sum((estimate - true_value)^2)
}

# Calculating squared errors
se_mle <- squared_error(mle_estimate, average_rest)
se_js_zero <- squared_error(js_zero_estimate, average_rest)
se_js_common <- squared_error(js_common_estimate, average_rest)

squared_errors <- data.frame(
  Method = c("MLE", "JS Zero", "JS Common"),
  Squared_Error = c(se_mle, se_js_zero, se_js_common)
)

squared_errors_gt <- gt(squared_errors)
gt_output <- "gt img/squared_errors_gt.png"
gtsave(squared_errors_gt, gt_output)
```

Method	Squared_Error
MLE	83.61966
JS Zero	77.43922
JS Common	75.16283

Task 2

```r
set.seed(123)  # For reproducibility
n <- 1000  # Number of observations
p <- 950   # Number of variables
```

```r
sigma <- 1/sqrt(1000)  # Standard deviation

# Generate the design matrix X
X <- matrix(rnorm(n*p,0,sigma), nrow = n, ncol = p)
dim(X)
```

```
## [1] 1000  950
```

```r
# Generate the response variable Y
beta <- c(rep(3, 5), rep(0, p-5))  # Coefficients
epsilon <- rnorm(n, mean = 0, sd = 1)  # Noise vector
mu <- X%*%beta
Y <- X %*% beta + epsilon
dim(Y)
```

```
## [1] 1000    1
```

```r
# Function to perform the analysis
perform_analysis <- function(X, Y, num_vars, sigma, n) {
  # Subset the design matrix to the first num_vars variables
  X_sub <- X[, 1:num_vars]
  # Convert to data frame for use in lm()
  df <- as.data.frame(X_sub)
  names(df) <- paste0("V", 1:num_vars) # Assign column names for the formula
  df$Y <- Y  # Add the response variable to the dataframe

  # Fit the linear model using the dataframe and a formula interface
  formula <- as.formula(paste("Y ~ ", paste(names(df)[1:num_vars], collapse=" + "), " - 1", sep=""))
  fit <- lm(formula, data = df)

  # a) Estimate beta and calculate RSS and PE
  beta_hat <- coef(fit)
  muhat <- X_sub%*%beta_hat
  rss <- sum(resid(fit)^2)
  pe <- sum((muhat-mu)^2)+n
  rss_adjust <- rss + 2*num_vars

  # c) Leave-one-out cross-validation to estimate PE
  fit_glm <- glm(df$Y ~ X_sub, family=gaussian())
  loo <- cv.glm(df, fit_glm, K = n)
  pe_loo <- mean(loo$delta)

  return(list(beta_hat = beta_hat, RSS = rss, rss_adjust = rss_adjust, pe = pe,  PE_loo = pe_loo))
}
```

```r
# Perform the analyses for each number of variables
#vars_2 <- perform_analysis(X, Y, 2, sigma, n)
#vars_5 <- perform_analysis(X, Y, 5, sigma, n)
#vars_10 <- perform_analysis(X, Y, 10, sigma, n)
#vars_100 <- perform_analysis(X, Y, 100, sigma, n)
#vars_500 <- perform_analysis(X, Y, 500, sigma, n)
#vars_950 <- perform_analysis(X, Y, p, sigma, n)
#write_rds(vars_2, "fit/vars_2.rds")
#write_rds(vars_5, "fit/vars_5.rds")
#write_rds(vars_10, "fit/vars_10.rds")
#write_rds(vars_100, "fit/vars_100.rds")
```

```r
#write_rds(vars_500, "fit/vars_500.rds")
#write_rds(vars_950, "fit/vars_950.rds")
```

```r
vars_2 <- read_rds("fit/vars_2.rds")
vars_5 <- read_rds("fit/vars_5.rds")
vars_10 <- read_rds("fit/vars_10.rds")
vars_100 <- read_rds("fit/vars_100.rds")
vars_500 <- read_rds("fit/vars_500.rds")
vars_950 <- read_rds("fit/vars_950.rds")
```

```r
var_names <- c("vars_2", "vars_5", "vars_10", "vars_100", "vars_500", "vars_950")
results_df <- data.frame("Number of Variables" = numeric(),
                         RSS = numeric(),
                         rss_adjust = numeric(),
                         PE = numeric(),
                         PE_loo = numeric())

for (var_name in var_names) {
  # Retrieve the variable from the environment
  var_value <- get(var_name)

  # Append the values to the results data frame
  results_df <- rbind(results_df, data.frame("Number of Variables" = as.numeric(gsub("vars_", "", var_na
                                             RSS = var_value$RSS,
                                             RSS_adjust = var_value$rss_adjust,
                                             PE = var_value$pe,
                                             PE_loo = var_value$PE_loo
                                             ))
}


min_values <- sapply(results_df[, 2:5], min)
results_gt <- gt(results_df)
for (i in seq_along(min_values)) {
  results_gt <- results_gt %>%
    text_transform(
      locations = cells_body(
        columns = i+1,
        rows = results_df[[i+1]] == min_values[[i]]
      ),
      fn = function(x) {
        html("<span style='color: red;'>", x, "</span>")
      }
    )
}
gt_output2 <- "gt img/results_gt.png"
gtsave(results_gt, gt_output2)
```

Number.of.Variables	RSS	RSS_adjust	PE	PE_loo
2	1012.94651	1016.9465	1030.172	1.048200
5	975.50314	985.5031	1006.064	1.085599
10	970.29381	990.2938	1011.273	1.090805
100	870.40886	1070.4089	1111.158	1.191656
500	458.74345	1458.7435	1522.824	1.602427
950	60.20503	1960.2050	1921.362	2.002739

The more variables you have the less will RSS be. So we don't get much information just by looking att RSS. Both RSS_adjust and pe suggests that 5 variables is the optimal number of variables. The prediction error using leave-one-out cross validation is best when only two varibales where used.