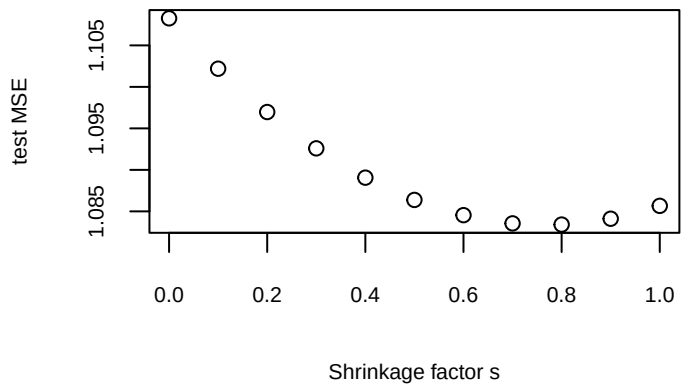
Statistical Learning week 1 - Answers to exercises

Exercise 1: Bias can be beneficial

```
set.seed(1)
x <- runif(50, min = -3, max = 3)
epsilon <- rnorm(50)
y <- 0.1*x + epsilon
train_dat <- data.frame(x, y)
beta_OLS <- coef(lm(y ~ 0 + x, data = train_dat))
beta_OLS
```

```
##         x
## 0.1184196
```

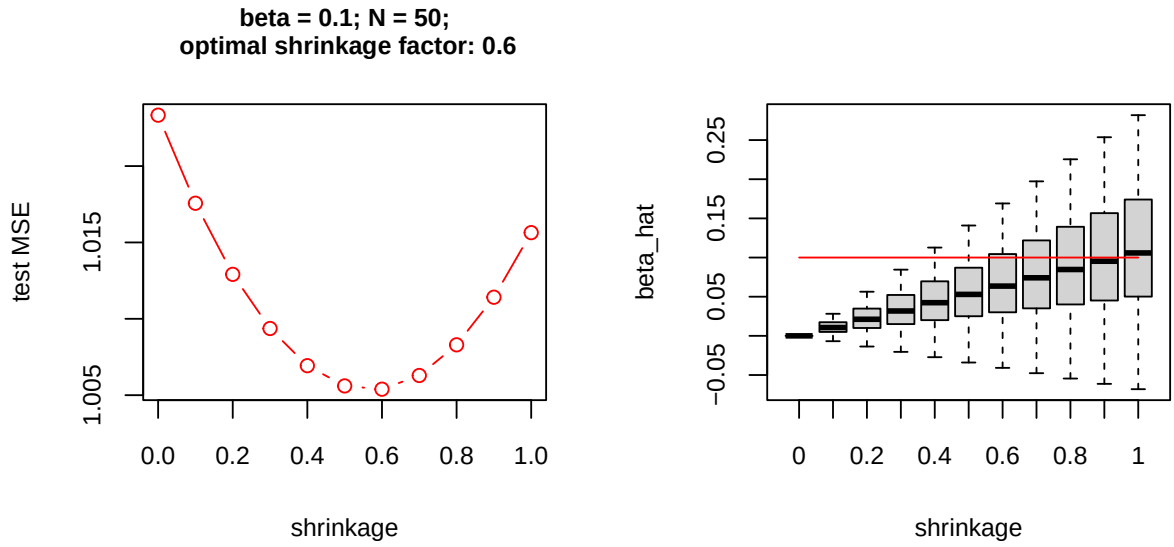
```
x <- runif(1000, min = -3, max = 3)
epsilon <- rnorm(1000)
y <- 0.1*x + epsilon
test_dat <- data.frame(x, y)
s <- seq(0, 1, by = .1)
test_MSE <- numeric(length(s))
for (i in 1:length(s)) {
  y_hat <- test_dat$x * s[i] * beta_OLS
  test_MSE[i] <- mean((y_hat - test_dat$y)^2)
}
plot(s, test_MSE, xlab = "Shrinkage factor s", ylab = "test MSE",
     cex.lab = .7, cex.axis = .7)
```



We see the optimal value of the shrinkage factor is less than 1, so the OLS coefficient (which has $s = 1$) is unbiased, but not optimal in terms of prediction.

Not every value of the random seed will yield this result, so we repeat the experiment with 100 replications:

```
beta <- 0.1
n <- 50
n_reps <- 100
shrinkage <- seq(0, 1, by = 0.1)
mse <- beta_hats <- matrix(0, nrow = n_reps, ncol = length(shrinkage))
colnames(mse) <- colnames(beta_hats) <- shrinkage
set.seed(1234)
for (i in 1:n_reps) {
  ## generate training data
  x <- runif(n, min = -3, max = 3)
  y <- beta*x + rnorm(n)
  ## fit OLS and get parameter estimates
  fit <- lm(y ~ 0 + x)
  b_ols <- coef(fit)
  ## generate test data:
  xtest <- runif(1000, min = -3, max = 3)
  ytest <- beta*xtest + rnorm(1000)
  ## apply shrinkage and obtain predictions
  for (s in 1:length(shrinkage)) {
    ## generate predictions for test observations
    ypred <- xtest * shrinkage[s] * b_ols
    mse[i, s] <- mean((ytest - ypred)^2)
    beta_hats[i, s] <- shrinkage[s] * b_ols
  }
}
par(mfrow = c(1, 2))
min_id <- which(colMeans(mse) == min(colMeans(mse)))
## Plot MSE versus shrinkage
plot(x = shrinkage, y = colMeans(mse), type = 'b',
     col = "red", xlab = "shrinkage", ylab = "test MSE",
     main = paste0("beta = ", beta, "; N = ", n,
                   ";\n optimal shrinkage factor: ",
                   shrinkage[min_id]), cex.main = .8,
     cex.lab = .8, cex.axis = .8)
## Plot distributions of beta estimates (add line for true value)
boxplot(beta_hats, xlab = "shrinkage", ylab = "beta_hat", outline = FALSE,
        cex.lab = .8, cex.axis = .8)
lines(c(1, 11), c(beta, beta), col = "red")
```

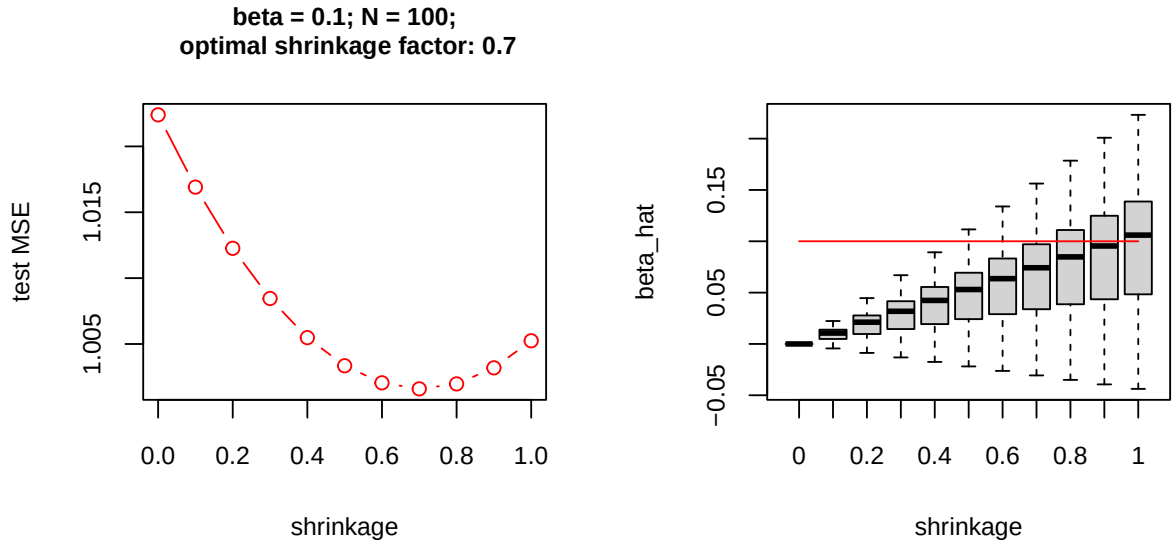


With shrinkage, we see that the estimated coefficient $\hat{\beta}$ becomes biased downwards (the red line in the middle plot indicates the true value β), but the variance of the estimate also gets (much) smaller. A shrinkage factor of 0.6 was optimal. The red line in the right plot indicates the irreducible error.

What if we increase training sample size?

```
n <- 100
set.seed(1234)
for (i in 1:n_reps) {
  ## generate training data
  x <- runif(n, min = -3, max = 3)
  y <- beta*x + rnorm(n)
  ## fit OLS and get parameter estimates
  fit <- lm(y ~ 0 + x)
  b_ols <- coef(fit)
  ## generate test data
  xtest <- runif(1000, min = -3, max = 3)
  ytest <- beta*xtest + rnorm(1000)
  ## apply shrinkage and obtain predictions
  for (s in 1:length(shrinkage)) {
    ## generate predictions for test observations
    ypred <- xtest * shrinkage[s] * b_ols
    mse[i, s] <- mean((ytest - ypred)^2)
    beta_hats[i, s] <- shrinkage[s] * b_ols
  }
}
par(mfrow = c(1, 2))
min_id <- which(colMeans(mse) == min(colMeans(mse)))
## Plot MSE versus shrinkage
plot(x = shrinkage, y = colMeans(mse), type = 'b',
     col = "red", xlab = "shrinkage", ylab = "test MSE",
     main = paste0("beta = ", beta, "; N = ", n,
                   ";\n optimal shrinkage factor: ",
                   shrinkage[min_id]),
     cex.lab = .8, cex.axis = .8, cex.main = .8)
```

```r
## Plot distributions of beta estimates (add line for true value)
boxplot(beta_hats, xlab = "shrinkage", ylab = "beta_hat", outline = FALSE,
        cex.lab = .8, cex.axis = .8)
lines(c(1, 11), c(beta, beta), col = "red")
```

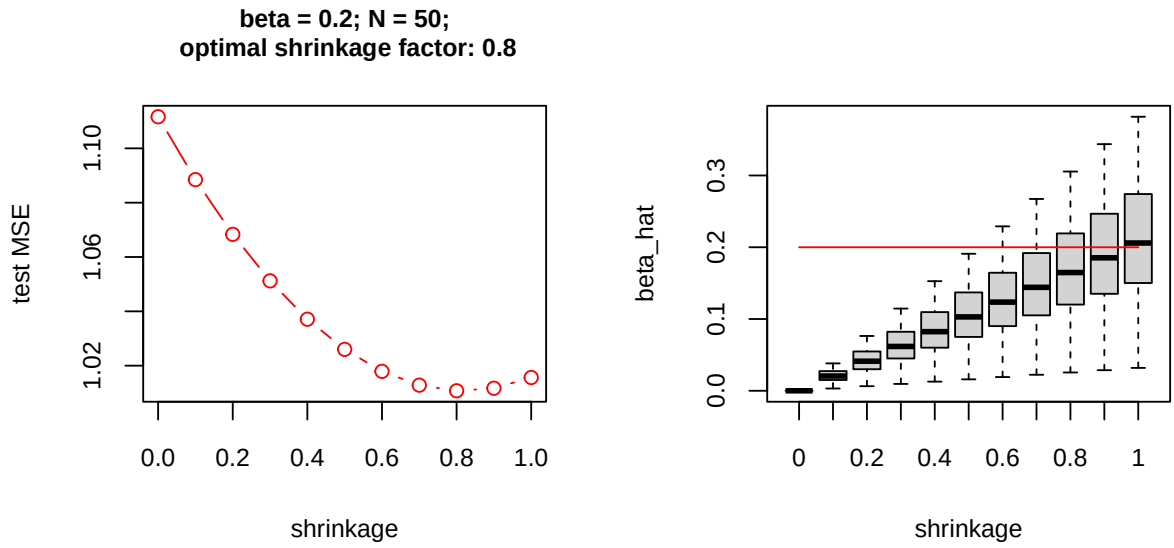


With larger sample size (i.e., more information in the sample), shrinkage is still beneficial. However, with larger sample size, variance of $\hat{\beta}$ is smaller, so we need less shrinkage (bias) to optimize prediction error. Now, a shrinkage factor of 0.7 was optimal.

What if we increase the effect of X?

```r
beta <- 0.2
n <- 50
set.seed(1234)
for (i in 1:n_reps) {
  ## generate training data
  x <- runif(n, min = -3, max = 3)
  y <- beta*x + rnorm(n)
  ## fit OLS and get parameter estimates
  fit <- lm(y ~ 0 + x)
  b_ols <- coef(fit)
  ## generate test data
  xtest <- runif(1000, min = -3, max = 3)
  ytest <- beta*xtest + rnorm(1000)
  ## apply shrinkage and obtain predictions
  for (s in 1:length(shrinkage)) {
    ## generate predictions for test observations
    ypred <- xtest * shrinkage[s] * b_ols
    mse[i, s] <- mean((ytest - ypred)^2)
    beta_hats[i, s] <- shrinkage[s] * b_ols
  }
}
par(mfrow = c(1, 2))
min_id <- which(colMeans(mse) == min(colMeans(mse)))
```

```r
## Plot MSE versus shrinkage factor
plot(x = shrinkage, y = colMeans(mse), type = 'b',
     col = "red", xlab = "shrinkage", ylab = "test MSE",
     main = paste0("beta = ", beta, "; N = ", n,
                   ";\n optimal shrinkage factor: ",
                   shrinkage[min_id]),
     cex.main = .8, cex.lab = .8, cex.axis = .8)
## Plot distributions of beta estimates (add line for true value)
boxplot(beta_hats, xlab = "shrinkage", ylab = "beta_hat", outline = FALSE,
        cex.lab = .8, cex.axis = .8)
lines(c(1, 11), c(beta, beta), col = "red")
```

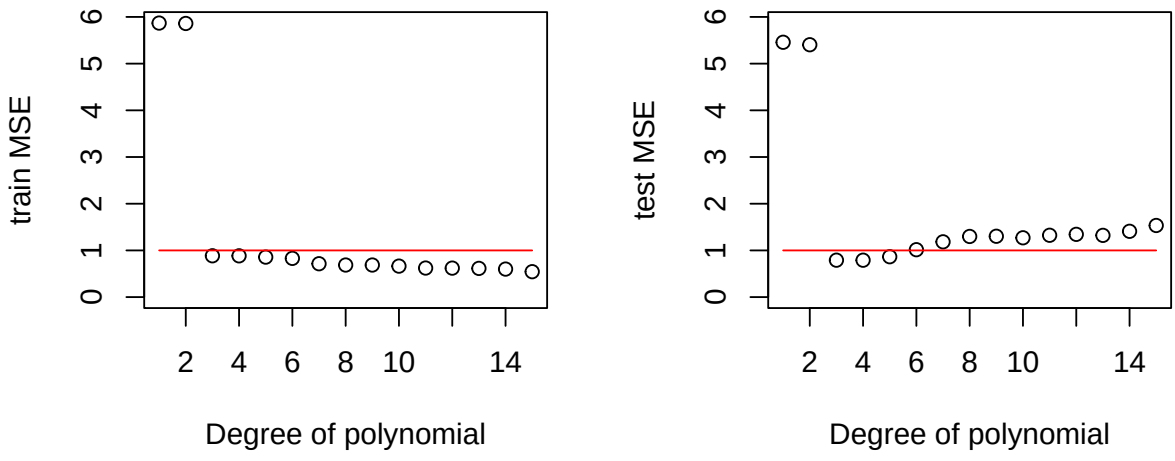


With larger effect size (i.e., larger β, so higher signal-to-noise ratio), shrinkage is still beneficial. Note that the variance of $\hat{\beta}$ does not change as a function of effect size. However, the shrinkage factor c has a stronger effect on larger coefficients, so we need less shrinkage (bias) to optimize prediction error. A shrinkage factor of 0.8 was optimal.

Conclusion: Shrinkage is beneficial for prediction. With higher signal-to-noise ratio and/or larger training sample size (i.e., there is more information in the training data), a lower amount of shrinkage is optimal.

Exercise 2: Under- and overfitting with polynomial regression

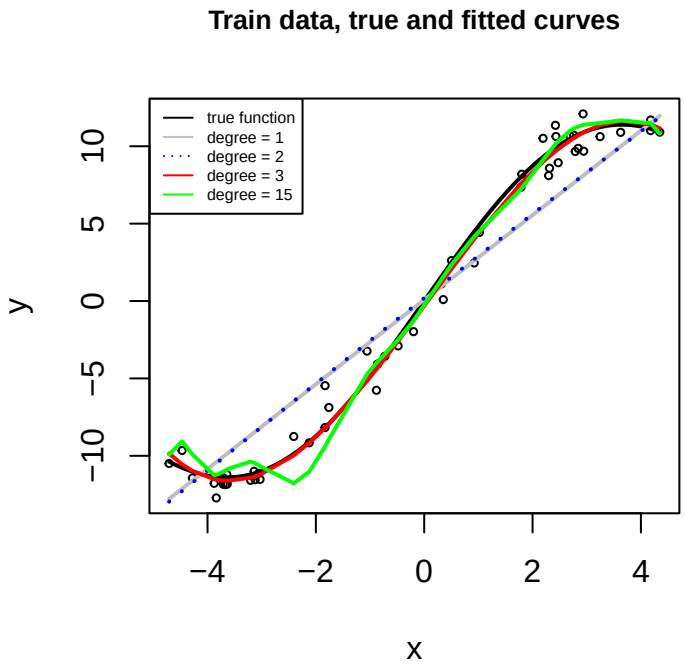
```r
set.seed(42)
n <- 50

## generate training data
x <- runif(n, min = -5, max = 5)
y <- x + 8*sin(x/2) + rnorm(n)
train <- data.frame(x, y)

## generate test data:
x <- runif(n, min = -5, max = 5)
y <- x + 8*sin(x/2) + rnorm(n)
test <- data.frame(x, y)

fit <- train_pred <- test_pred <- train_err <- test_err <- list()

for (d in 1:15) {
  fit[[d]] <- lm(y ~ poly(x, degree = d), data = train)
  train_pred[[d]] <- predict(fit[[d]], newdata = train)
  test_pred[[d]] <- predict(fit[[d]], newdata = test)
  test_err[[d]] <- mean((test$y - test_pred[[d]])^2)
  train_err[[d]] <- mean((train$y - train_pred[[d]])^2)
}
train_err <- unlist(train_err)
test_err <- unlist(test_err)
par(mfrow = c(1, 2))
plot(1:15, train_err, xlab = "Degree of polynomial",
     ylab = "train MSE", ylim = c(0, max(c(train_err, test_err))))
lines(c(1, 15), c(1, 1), col = "red")
plot(1:15, test_err, xlab = "Degree of polynomial",
     ylab = "test MSE", ylim = c(0, max(c(train_err, test_err))))
lines(c(1, 15), c(1, 1), col = "red")
```



The MSE is depicted against the degree of the polynomial. The red curve represents irreducible error. We see rather high train and test MSE for polynomials of degree 1 and 2, and then a sharp decrease afterwards. After degree 6, the overfitting begins and test MSE starts to increase, while train MSE continues decreasing. Note that, taking into account the irreducible error (which has variance of 1), the cubic (degree 3) model does pretty well.

We plot the fitted curves against the training observations:

```
plot(x, y, main = "Train data, true and fitted curves", cex = .5, cex.main = .8)
curve(x + 8*sin(x/2), add = TRUE, lwd = 2)
lines(sort(test$x), test_pred[[1]][order(test$x)], col = "grey", lwd = 2)
lines(sort(test$x), test_pred[[2]][order(test$x)], col = "blue", lwd = 2, lty = 3)
lines(sort(test$x), test_pred[[3]][order(test$x)], col = "red", lwd = 2)
lines(sort(test$x), test_pred[[15]][order(test$x)], col = "green", lwd = 2)
legend("topleft", legend = c("true function", paste("degree =", c(1, 2, 3, 15))),
       lty = c(1, 1, 3, 1, 1), col = c("black", "grey", "blue", "red", "green"),
       cex = .5)
```

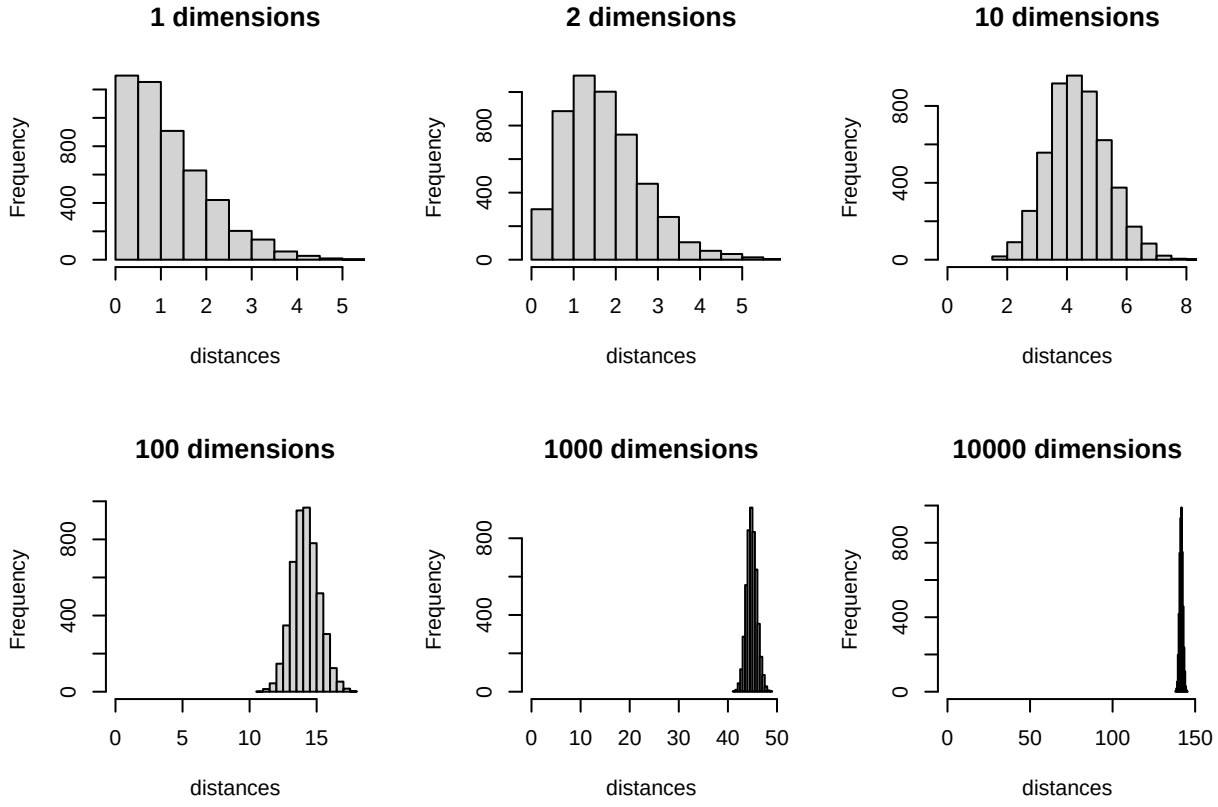


The linear and quadratic clearly stand out, failing to capture non-linearities. The other curves follow closely the true conditional means, although the higher-order polynomials show aberrant behaviour at the boundaries and may adjust to the training data too much. Note that for these data, the erratic behavior near the boundaries might not affect test MSE too much, because observations near the boundary are rare in a single dimension. But in increasingly higher dimension, observations will be increasingly closer to the boundaries of the space.

Exercise 3: Curse of dimensionality

```
p <- 10000
N <- 100
set.seed(42)
X <- matrix(rnorm(p*N), ncol = p, nrow = N)
par(mfrow = c(2, 3))
## L_2
for (p in c(1, 2, 10, 100, 1000, 10000)) {
  distances <- dist(X[ , 1:p])
  if (p == 1) print(head(distances))
  hist(distances, main = paste(p, "dimensions"), xlim = c(0, max(distances)))
}
```

```
## [1] 1.9356566 1.0078300 0.7380958 0.9666901 1.4770830 0.1405636
```



k nearest neighbours (and many other distance-based methods, too) assume that nearness is meaningful: that observations that are closer by are more similar than observations further apart.

In low dimensions, the Euclidian distances between observations indeed seem meaningful: Distances show quite some variability, there are many observation pairs very near (almost zero distance), and many observation pairs are further away (not neighbours).

With increasing dimension p, it is increasingly the case that all observations are far apart. With very high dimensions, distances between observations seem not so meaningful anymore: All observations are far apart, none are near. One could argue, among observations that are all at a large distance, there are no real neighbours. Being nearer by 1 or 2 is likely to reflect only chance fluctuation, compared to the average distance of 141.5624338 (SD = 0.9874754 in $p = 10,000$ dimensions. It is a bit like living in Leiden and that

you would call anyone from the eastern part of New York your neighbour, but anyone from the western part of New York would not be your neighbour.

Exercise 4: Flexibility and predictive performance

a) The sample size n is extremely large, and the number of predictors p is small.

Flexible method probably better. There is a low number of predictors, so no 'curse of dimensionality'; extremely large sample size, so probably enough information in the data to reliably estimate flexible model.

b) The number of predictors p is extremely large, and the number of observations N is small.

Inflexible method probably better. Extremely large number of predictors, so 'curse of dimensionality' applies; small sample size, so probably too little information in the data to reliably estimate a flexible / complex model.

c) The relationship between the predictors and response is highly non-linear.

Flexible method probably better: need flexible method to deal with non-linear association.

d) The variance of the error terms, i.e. $\sigma^2 = Var(\epsilon)$, is extremely high.

Inflexible method probably better. With a noisy data problem (i.e., a lot of irreducible error), a flexible method may overfit. However, if sample size is (very) large, a flexible method may still perform well, as long as it employs some kind of smoothing procedure to not overfit on individual errors.