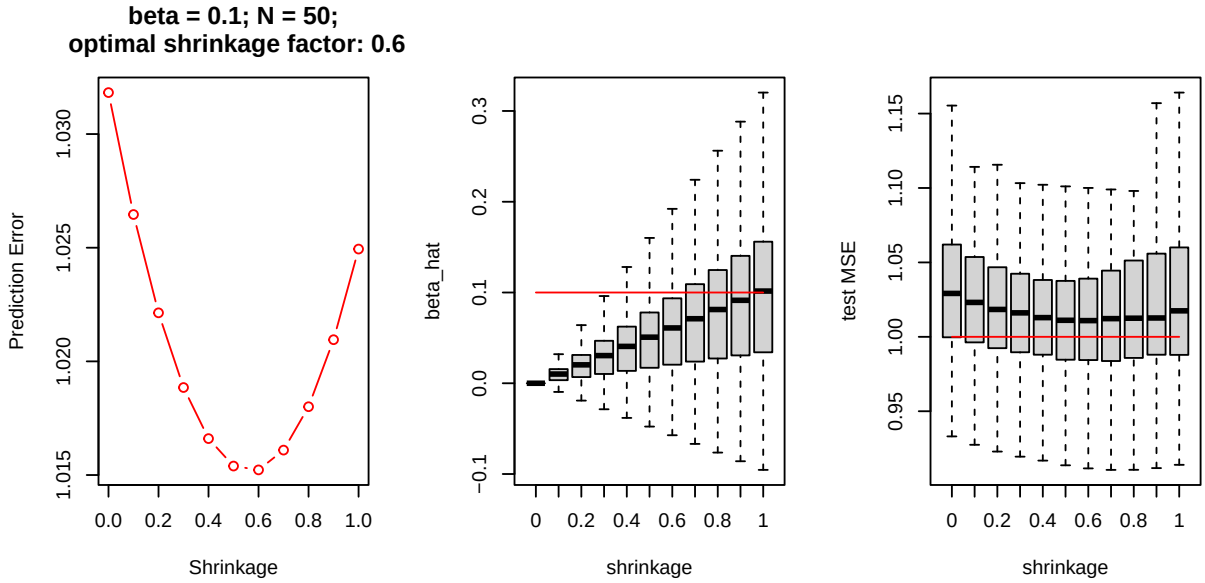
Statistical Learning week 1 - Answers to exercises

Exercise 1: Shrinkage

```r
beta <- 0.1 # effect size
n <- 50 # sample size
n_reps <- 100 # no. of replications
shrinkage <- seq(0, 1, by = 0.1) # values for shrinkage parameter
error <- beta_hats <- matrix(0, nrow = n_reps, ncol = length(shrinkage))
colnames(error) <- colnames(beta_hats) <- shrinkage # objects for savig results
set.seed(42)
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
    error[i, s] <- mean((ytest - ypred)^2)
    beta_hats[i, s] <- shrinkage[s] * b_ols
  }
}
par(mfrow = c(1,3))
min_id <- which(colMeans(error) == min(colMeans(error)))
## Plot MSE versus shrinkage factor
plot(x = shrinkage, y = colMeans(error), type = 'b',
     col = "red", xlab = "Shrinkage", ylab = "Prediction Error",
     main = paste0("beta = ", beta, "; N = ", n,
                   ";\n optimal shrinkage factor: ",
                   shrinkage[min_id]))
## Plot distributions of beta estimates (add line for true value)
boxplot(beta_hats, xlab = "shrinkage", ylab = "beta_hat", outline = FALSE)
lines(c(1, 11), c(0.1, 0.1), col = "red")
## Plot distributions of MSE (add line for irreducible error)
boxplot(error, ylab = "test MSE", xlab = "shrinkage", outline = FALSE)
lines(c(1, 11), c(1, 1), col = "red")
```




```
## Compute variance of the shrunken and OLS estimates
round(apply(beta_hats, 2, var), digits = 3)
```

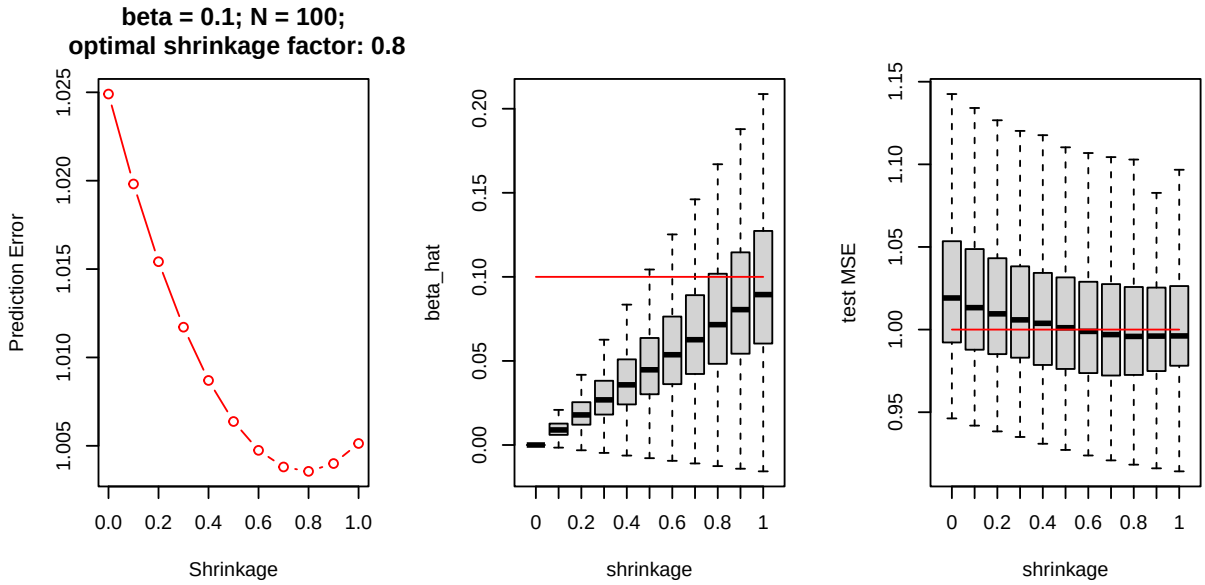
```
##     0   0.1   0.2   0.3   0.4   0.5   0.6   0.7   0.8   0.9     1
## 0.000 0.000 0.000 0.001 0.001 0.002 0.003 0.004 0.005 0.006 0.007
```

With shrinkage, we see that the estimated coefficient $\hat{\beta}$ are biased towards zero (the red line in the middle plot indicates the true value of β), but this also reduces the variance of the estimated coefficients, which is beneficial for prediction. For this specific data problem, a shrinkage factor of 0.6 seems optimal. The red line in the right plot indicates the irreducible error.

What if we use a larger sample size?

```
n <- 100 # adjust sample size
set.seed(42)
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
    error[i, s] <- mean((ytest - ypred)^2)
    beta_hats[i, s] <- shrinkage[s] * b_ols
  }
}
par(mfrow = c(1,3))
min_id <- which(colMeans(error) == min(colMeans(error)))
```

```r
## Plot MSE versus shrinkage factor
plot(x = shrinkage, y = colMeans(error), type = 'b',
     col = "red", xlab = "Shrinkage", ylab = "Prediction Error",
     main = paste0("beta = ", beta, "; N = ", n,
                   ";\n optimal shrinkage factor: ",
                   shrinkage[min_id]))
## Plot distributions of beta estimates (add line for true value)
boxplot(beta_hats, xlab = "shrinkage", ylab = "beta_hat", outline = FALSE)
lines(c(1, 11), c(0.1, 0.1), col = "red")
## Plot distributions of MSE (add line for irreducible error)
boxplot(error, ylab = "test MSE", xlab = "shrinkage", outline = FALSE)
lines(c(1, 11), c(1, 1), col = "red")
```




```r
## Compute variance of the shrunken and OLS estimates
round(apply(beta_hats, 2, var), digits = 3)
```

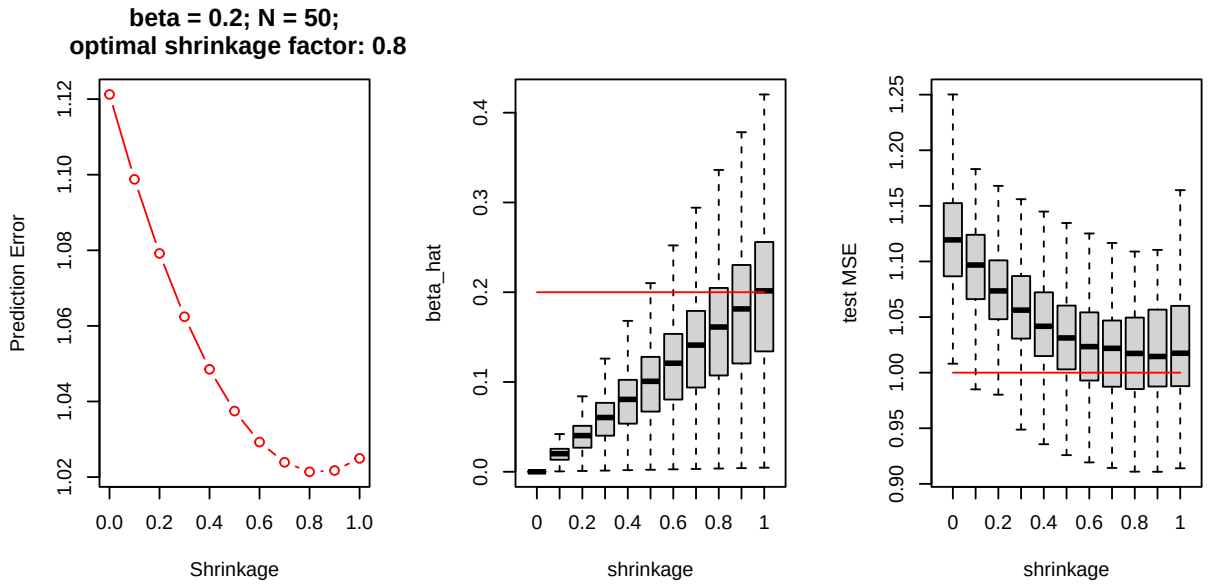
```
##     0   0.1   0.2   0.3   0.4   0.5   0.6   0.7   0.8   0.9     1
## 0.000 0.000 0.000 0.000 0.001 0.001 0.001 0.002 0.002 0.003 0.003
```

With larger sample size (i.e., more information in the sample), shrinkage is still beneficial. With larger sample size, however, the variance of $\hat{\beta}$ is lower, so we need less shrinkage (bias) to optimize prediction error. For this data problem, a shrinkage factor of 0.8 seems optimal.

What if we increase the effect size?

```r
beta <- 0.2 # adjust effect size
n <- 50 # set sample size to original value
set.seed(42)
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
    error[i, s] <- mean((ytest - ypred)^2)
    beta_hats[i, s] <- shrinkage[s] * b_ols
  }
}
par(mfrow = c(1,3))
min_id <- which(colMeans(error) == min(colMeans(error)))
## Plot MSE versus shrinkage factor
plot(x = shrinkage, y = colMeans(error), type = 'b',
     col = "red", xlab = "Shrinkage", ylab = "Prediction Error",
     main = paste0("beta = ", beta, "; N = ", n,
                   ";\n optimal shrinkage factor: ",
                   shrinkage[min_id]))
## Plot distributions of beta estimates (add line for true value)
boxplot(beta_hats, xlab = "shrinkage", ylab = "beta_hat", outline = FALSE)
lines(c(1, 11), c(0.2, 0.2), col = "red")
## Plot distributions of MSE (add line for irreducible error)
boxplot(error, ylab = "test MSE", xlab = "shrinkage", outline = FALSE)
lines(c(1, 11), c(1, 1), col = "red")
```




```
## Compute variance of the shrunken and OLS estimates
round(apply(beta_hats, 2, var), digits = 3)
```

```
##     0   0.1   0.2   0.3   0.4   0.5   0.6   0.7   0.8   0.9     1
## 0.000 0.000 0.000 0.001 0.001 0.002 0.003 0.004 0.005 0.006 0.007
```

With larger effect size (i.e., larger β, i.e., more information in the sample), shrinkage is still beneficial. With larger effect size, variance of $\hat{\beta}$ remains identical. However, the shrinkage factor has a stronger effect on larger

coefficients, so we need less shrinkage (bias) to optimize prediction error. For this data problem, a shrinkage factor of 0.8 seems optimal.

In conclusion: Shrinkage is beneficial for prediction. With higher effect and/or training sample size (i.e., more information in the training data), a lower amount of shrinkage is optimal.

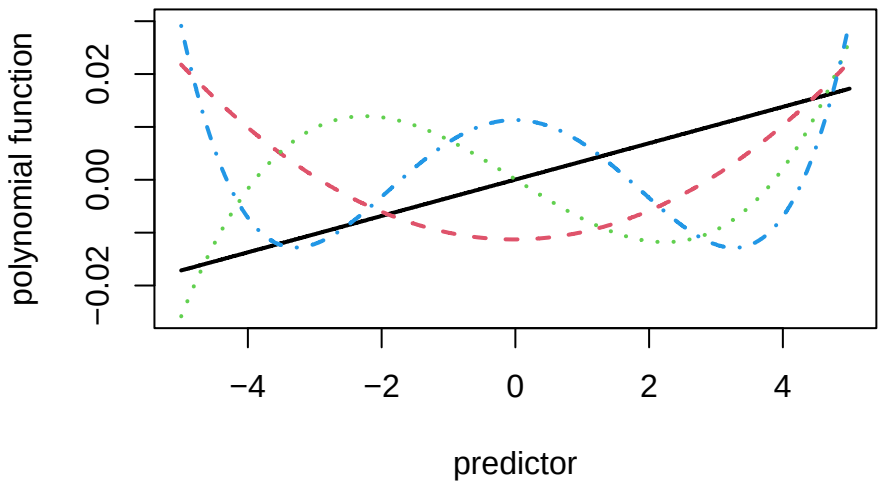
Exercise 2: Non-linear Regression

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

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)
}
```

Note that I used the `poly` function to set up the polynomial functions of predictor `x`. Argument `degree` specifies the degree of the polynomial. E.g., with `degree = 4`, we get a matrix with 4 columns: the 1st is a linear function of the predictor, the 2nd a quadratic function, the 3rd a cubic function, the 4th a quartic function. Note that `poly` sets up orthogonal contrasts; i.e., the columns of the design matrix are independent.

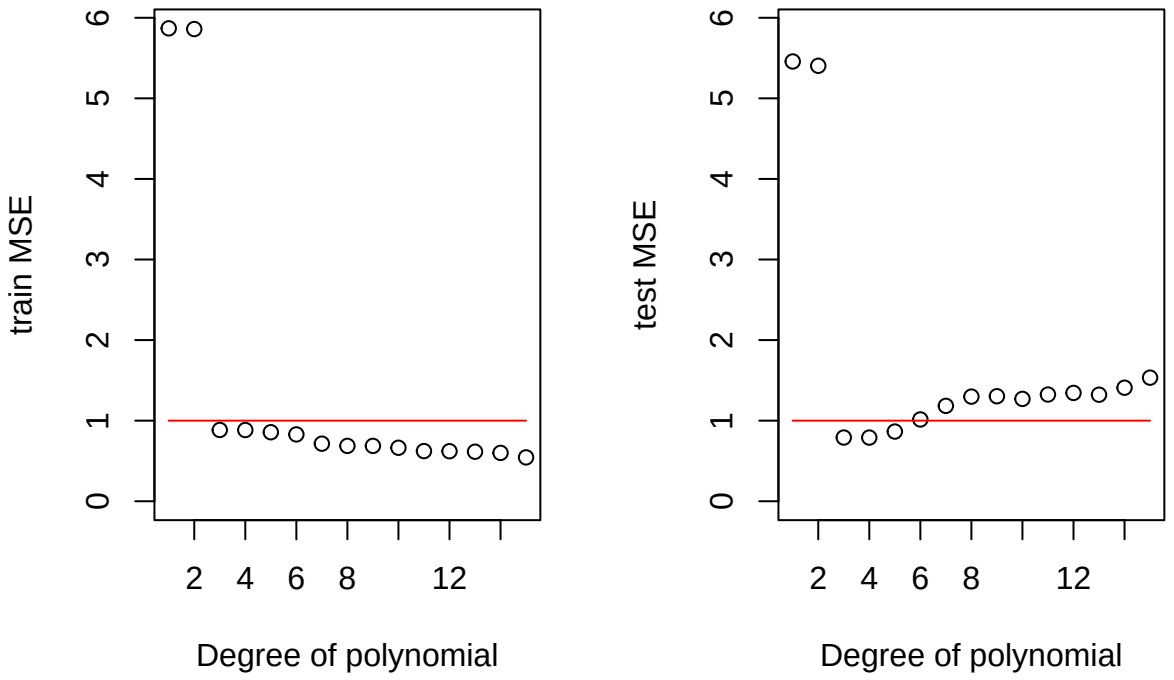
To illustrate, this is what the first four polynomial expansions (functions) look like:

```
set.seed(42)
x_tmp <- runif(10000, -5, 5)
pol <- poly(x_tmp, degree = 4)
matplot(x_tmp[order(x_tmp)], pol[order(x_tmp), ], type = "l", lwd = 2, xlab = "predictor",
        ylab = "polynomial function")
```



We now plot the training and test error of the fitted polynomial models:

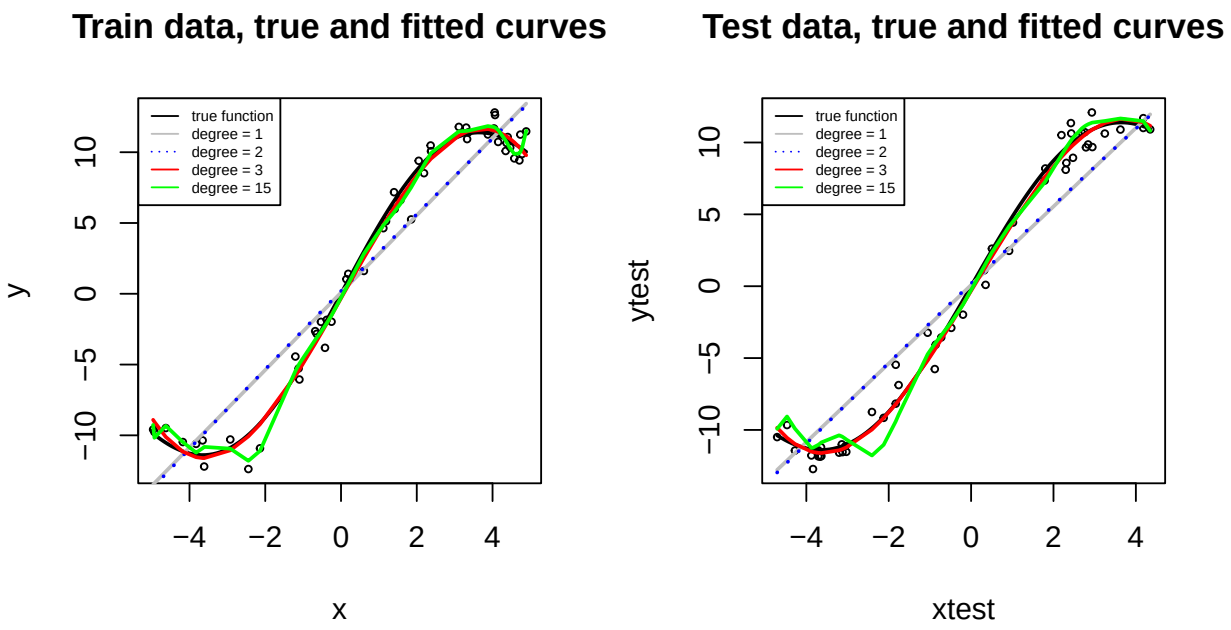
```
train_err <- unlist(train_err)
test_err <- unlist(test_err)
par(mfrow = c(1, 2))
plot(1:15, train_err, xlab = "Degree of polynomial",
     ylab = "train MSE", ylim = c(0, max(c(train_err, test_err))))
lines(c(1, 15), c(1, 1), col = "red")
plot(1:15, test_err, xlab = "Degree of polynomial",
     ylab = "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 that the train and test MSE are rather high for polynomials of degree 1 and 2, and shows a sharp decrease afterwards. After degree 6, the test MSE starts to increase again, reaching the maximum test MSE at degree 15. The train MSE only goes down with increasing complexity. Note that, taking into account the irreducible error (which has variance of 1), the cubic (degree 3) model does pretty well.

Let's inspect the fitted curves against the training and test observations:

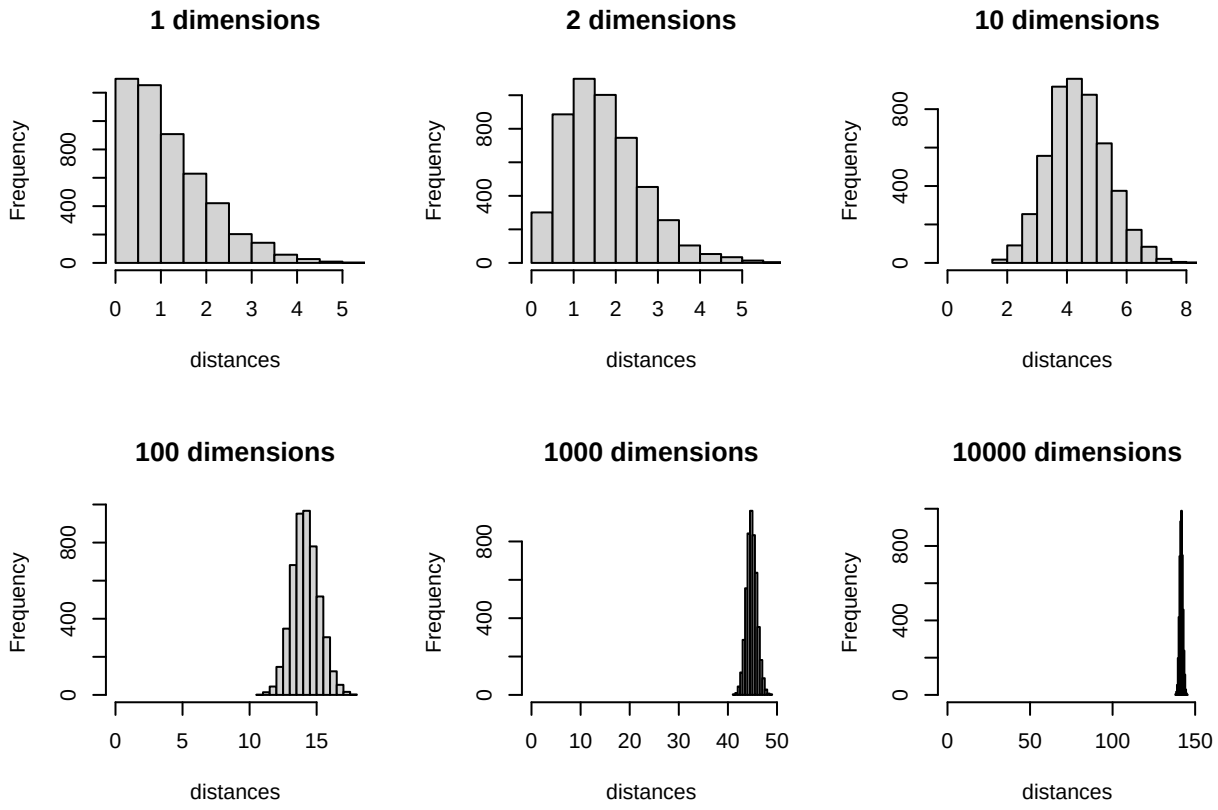
```
par(mfrow = c(1, 2))
plot(x, y, main = "Train data, true and fitted curves", cex = .5)
curve(x + 8*sin(x/2), add = TRUE, lwd = 2)
lines(sort(x), train_pred[[1]][order(x)], col = "grey", lwd = 2)
lines(sort(x), train_pred[[2]][order(x)], col = "blue", lwd = 2, lty = 3)
lines(sort(x), train_pred[[3]][order(x)], col = "red", lwd = 2)
lines(sort(x), train_pred[[15]][order(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)
plot(xtest, ytest, main = "Test data, true and fitted curves", cex = .5)
curve(x + 8*sin(x/2), add = TRUE, lwd = 2)
lines(sort(xtest), test_pred[[1]][order(xtest)], col = "grey", lwd = 2)
lines(sort(xtest), test_pred[[2]][order(xtest)], col = "blue", lwd = 2, lty = 3)
lines(sort(xtest), test_pred[[3]][order(xtest)], col = "red", lwd = 2)
lines(sort(xtest), test_pred[[15]][order(xtest)], 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 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_train <- matrix(rnorm(p*N), ncol = p, nrow = N)
par(mfrow = c(2, 3))
for (p in c(1, 2, 10, 100, 1000, 10000)) {
  distances <- dist(x_train[ , 1:p])
  hist(distances, main = paste(p, "dimensions"), xlim = c(0, max(distances)))
}
```



k nearest neighbours assumes that nearness, or distances between observations, 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, many observation pairs are very close (almost zero distance), but many observation pairs also have quite some distance between them. The larger the distance, the less common it is.

With increasing dimension, it is increasingly the case that all observations are far apart. With very high dimensions, the 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 distance of about 150, there are no real neighbours. Being nearer by 1 or 2 is likely to reflect only a chance fluctuation.

Exercise 4

a) If we assign to the majority class, all observations would be assigned to the red class, with $\hat{p}_{red} = \frac{2}{3}$. The test misclassification rate would be .5. The test MSE on predicted probabilities would be:

```
mean(c((2/3 - 0)^2, (2/3 - 1)^2))
```

```
## [1] 0.2777778
```

b)

```
x_train <- data.frame(x1 = c(0, 2, 0, 0, -1, 1),
x2 = c(3, 0, 1, 1, 0, 1),
x3 = c(0, 0, 3, 2, 1, 1))
y_train <- c("red (1)", "red (1)", "red (1)", "green (0)", "green (0)", "red (1)")
x_test <- data.frame(x1 = c(0, 2),
                     x2 = c(0, 2),
                     x3 = c(0, 0))
distances <- apply(x_test, 1, function(x)
apply(x_train, 1, function(y) sqrt(sum((x - y)^2))))
colnames(distances) <- c("test_obs_1", "test_obs_2")
data.frame(x_train, distances, y_train)
```

```
##   x1 x2 x3 test_obs_1 test_obs_2   y_train
## 1  0  3  0   3.000000   2.236068   red (1)
## 2  2  0  0   2.000000   2.000000   red (1)
## 3  0  1  3   3.162278   3.741657   red (1)
## 4  0  1  2   2.236068   3.000000 green (0)
## 5 -1  0  1   1.414214   3.741657 green (0)
## 6  1  1  1   1.732051   1.732051   red (1)
```

c) With $k = 1$, test obs. 1 will be assigned to the green class with $\hat{p}_{red}(x) = 0$; test obs. 2 will be assigned to the red class, with $\hat{p}_{red}(x) = 1$.

d) With $k = 3$, test observation 1 will be assigned to the red class, with $\hat{p}_{red} = \frac{2}{3}$; test observation 2 will be assigned to the red class, with $\hat{p}_{red} = 1$.

e) Test observation 1 is actually green (0), and test observation 2 is red (1). The errors when predicting the majority class are:

```
1/2 * (1 + 0) # MCR
```

```
## [1] 0.5
```

```
1/2 * ( (1 - 4/6)^2 + (0 - 4/6)^2 ) # Brier
```

```
## [1] 0.2777778
```

The misclassification error rate is 0 for $k = 1$; this is an improvement over assigning to the majority class. The misclassification rate for $k = 3$ is .5, this is not an improvement over assigning to the majority class.

The squared error loss on predicted probabilities is:

```
mean(c((0 - 0)^2, (1 - 1)^2)) # for k = 1
```

```
## [1] 0
```

```
mean(c((0 - 2/3)^2, (1 - 1)^2)) # for k = 3
```

```
## [1] 0.2222222
```

According to the Brier score, both $k = 1$ and $k = 3$ are an improvement over assigning to the majority class.

ISL Exercise 2.1

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.

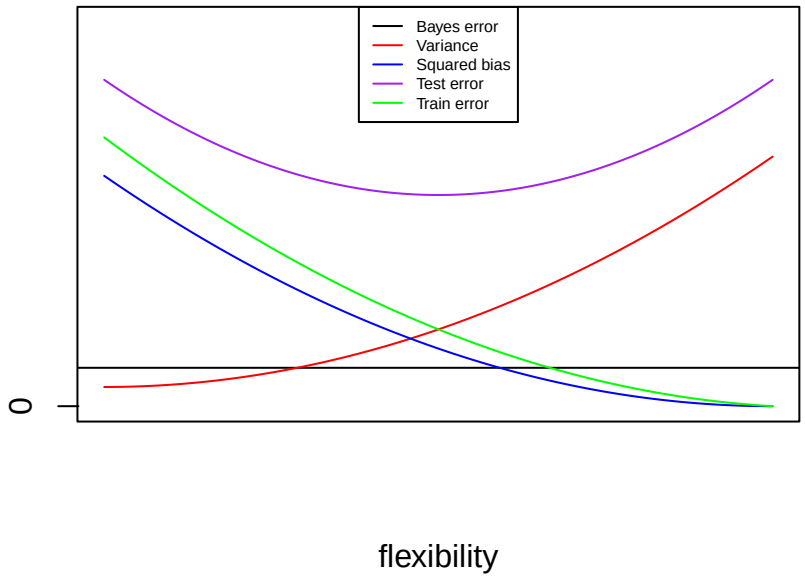
Flexible method probably better: need flexible method to deal with non-linear association.

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 enough, a flexible method may still perform well, as long as it employs some kind of smoothing procedure to not overfit on individual errors.

ISL Exercise 2.3

a) Note that the curves will have different relative heights, and may have somewhat different shapes, depending on the data problem.



b) Given a fixed-size sample from a fixed population (data problem):

- Bayes (irreducible) error remains constant
- Squared bias starts high and approaches 0 with increasing complexity
- Variance starts low (but > 0), and increases ever more strongly with increasing complexity
- Test error = irreducible error + variance + squared bias
- Training error is always lower than test error. With low flexibility, it is more or less the sum of the irreducible error and squared bias. Training error decreases with increasing flexibility and converges to the value of the squared bias. The difference between training and test error is mostly composed of variance (at least, in this example, where the irreducible error is generally low, compared to the variance and squared bias).

ISL Exercise 2.5

Advantage of flexible approach: Estimated prediction function $\hat{f}(x)$ can flexibly adjust to the data. Therefore, in principle a (very) flexible method can approximate / accommodate any function of the predictors, be it simple or complex.

Disadvantages of flexible approach: Estimated prediction function $\hat{f}(x)$ may too flexibly adapt to the data. If the sample size is not large enough, or the parameters of the model-fitting approach are not well chosen, the estimated $\hat{f}(x)$ may too closely adapt to noise / error / sampling fluctuations, and not generalize well to observations outside the training dataset. Also, a flexible model may be more difficult to interpret.

Flexible approaches may be preferred when the association between predictors and response is expected to be complex, sample size is large and the number of predictor variables is small. Inflexible approaches may be preferred if sample size is small, the association between predictors and response is expected to be relatively simple, the number of predictor variables is large, and/or the fitted model should be easy to interpret. should be easy to interpret.