Statistical Learning Week 1 - Solutions to exercises

8 February 2025

Exercises ISL2 book

Exercise ISL2 2.4/1

(a) 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) 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 and/or complex model.

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

(d) 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.

Exercise ISL2 2.4/2

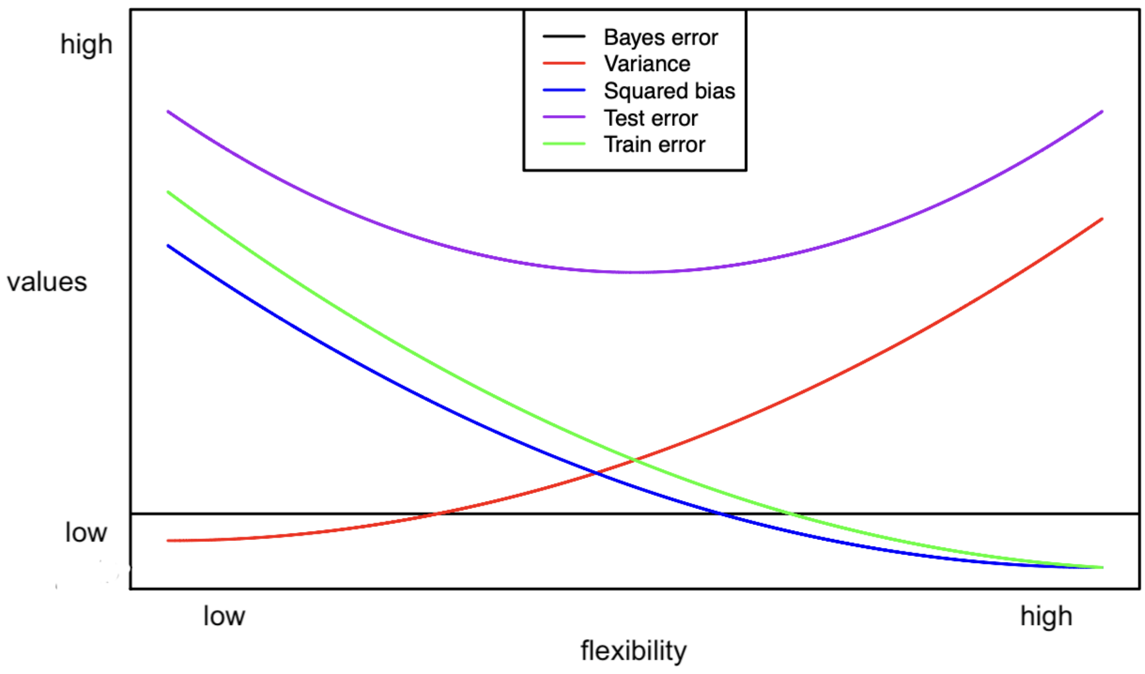
(a) The response is continuous, specifically the salary of a CEO. Hence, it is a regression problem. The goal is not to predict the salary of a CEO, but to understand the impact that certain predictors have on it. It is thus an inference task. The sample size for this problem is $n = 500$ and the predictors include profit, number of employees, and industry, thus, $p = 3$.

(b) In this case, the response is binary, specifically success or failure, and hence it is a classification problem. The aim here is to predict success or failure. $n = 20$ and $p = 13$.

(c) This problem involves a response that is continuous, specifically the % change in the USD/Euro exchange rate, and hence it is a regression problem. The goal is to predict the response, the % change in the USD/Euro exchange rate. $n = 52$ and $p = 3$

Exercise ISL2 2.4/3

(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 Hence decreases initially but increases later due to overfitting.
- 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).

Exercise ISL2 2.4/4

Answers will vary, some

(a) **Classification**

- **Predicting diabetes:** Given a set of risk factors (e.g., weight/BMI, family history of the disease, blood serum markers), predict if a person will develop diabetes in the next year. Note that if we wanted to quantify the risk for each risk factor, this data problem also could be an example for inference.
- **Fraud detection:** Classify whether a transaction is fraudulent, given data like the transaction amount, location, purchased item or service, previous customer transactions, etc. The response would be binary and our aim is to make a prediction.

(b) **Regression**

- **Sales:** Predict unit sales of a product given marketing data such as TV, Radio or Internet advert expenditure, and use it to infer the importance of each advertising method. The response is the unit sales of the product.
- **Driving Insurance premium:** Given a set of variables such as the drivers history, age, type of vehicle, expected yearly mileage and the premium as the response, we can train a model to predict the insurance premium for new customers.

(c) **Cluster analysis**

- **Patient classification:** : Clustering can be used to group similar patients with a specific disease into subgroups based on their clinical characteristics. This can be useful in sorting patients into categories into different treatment categories based on their needs.
- **Market research:** Sort people within a city into distinct market segments to increase marketing effectiveness. Demographic data such as age, sex, income, location, together with opinion polls or sales data could be used to identify similar groups.

Exercise ISL2 2.4/5

Inflexible methods are more interpretable, require fewer observations, but can have high bias when the underlying function is non-linear. They might be preferred if there are many predictors due to "curse of dimensionality". Less flexible (parametric) methods are useful if we assume or know that the underlying assumptions can be met. In such cases, the model needs to predict fewer parameters than a non-parametric method. They are preferred when we are interested in making inferences or when the interpretability of the results is important.

Flexible methods can deal with complex, non-linear patterns but typically require larger sample sizes and are less interpretable (black box models). Moreover, while they have a lower bias, they are also more prone to overfitting (high variance). Flexible (non-parametric) methods are preferable when we do not want to (or cannot) make assumptions about the function to be estimated or when we want to make accurate predictions, but care less about how those predictions were made.

Exercise ISL2 2.4/6

Parametric approaches use and estimate a known number of model parameters (e.g., $p + 1$ coefficients in a multiple linear regression) by assuming a specific form of f (i.e., linear in this example). Parametric models can be more interpretable as there is a model behind how data is generated. However, the disadvantage is that the model might not reflect reality. If the model is too far from the truth, estimates will be poor and more flexible models can fit many different forms and require more parameters (leading to overfitting).

Non-parametric approaches do not estimate a small number of parameters, so a large number of observations may be needed to obtain accurate estimates. We make no assumptions about the true form of f, which allows for larger variety of shapes and could accommodate more complex data patterns.

Exercises 2.1: Venn-diagram

Answers will vary, there are a couple of examples in the meeting slides. Note that statistical learning is missing from each.

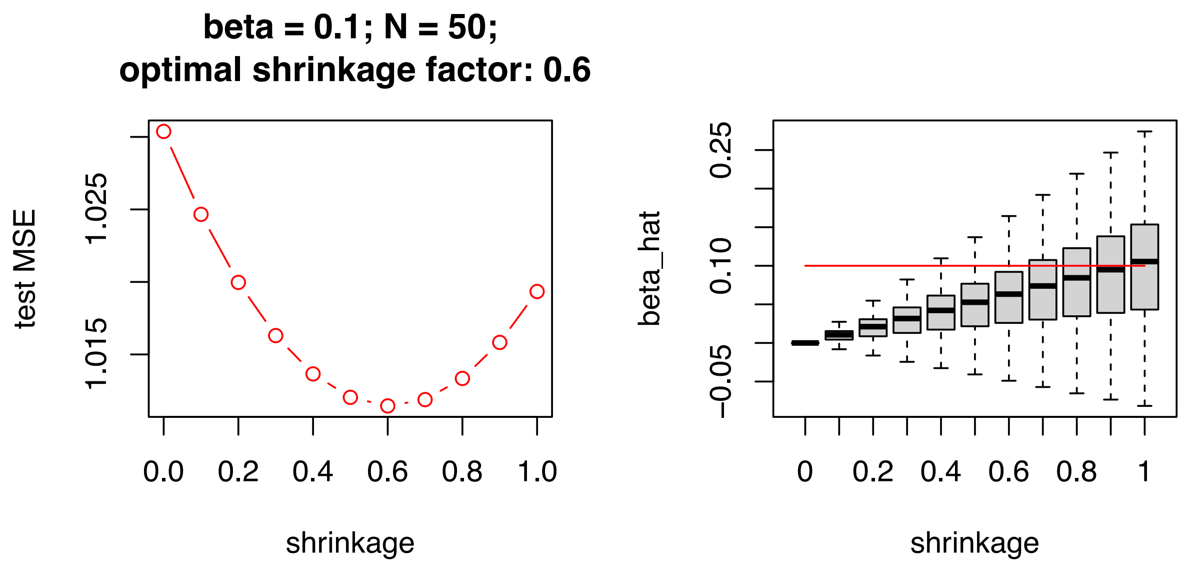
Exercises 2024

Exercise 2.1: Inference vs. Prediction?

See above 1.2

Exercise 2.2: Bias can be beneficial

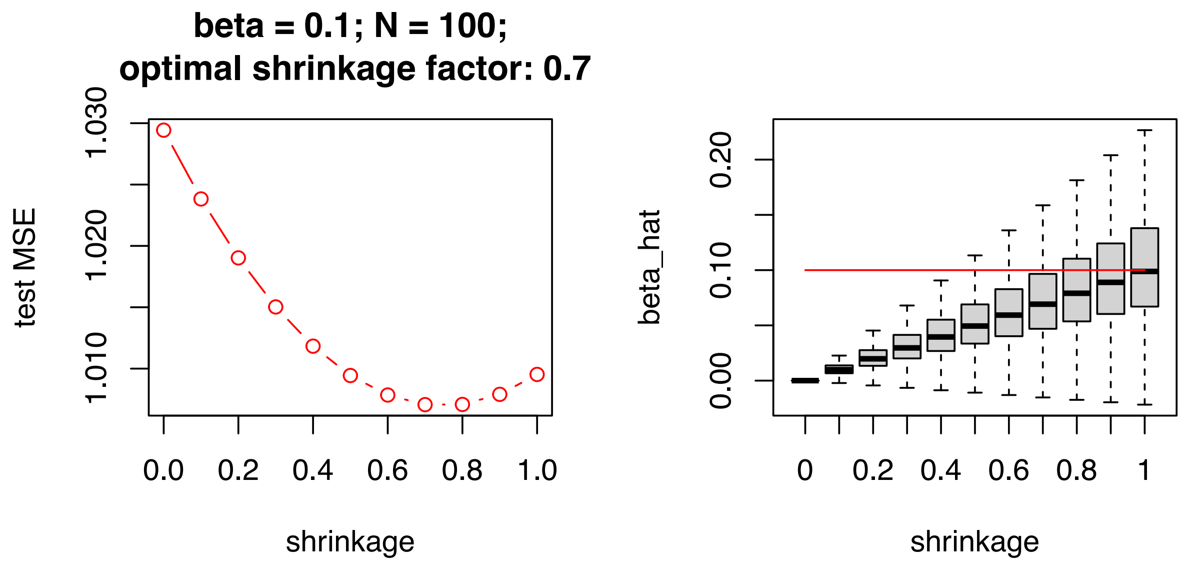
```r
make_ex_1 <- function(beta = .1, n_train = 50) {
  n_reps <- 100
  shrinkage <- seq(0, 1, by = 0.1)
  mse <- beta_hats <- matrix(0, nrow = n_reps, ncol = length(shrinkage))
  colnames(mse) <- colnames(beta_hats) <- shrinkage
  set.seed(123)
  for (i in 1:n_reps) {
    ## generate training data
    x <- runif(n_train, min = -3, max = 3)
    y <- beta * x + rnorm(n_train)

    ## fit OLS and get parameter estimates
    fit <- lm(y ~ 0 + x)
    b_ols <- coef(fit)

    ## generate test data:
    xtest <- runif(10000, min = -3, max = 3)
    ytest <- beta * xtest + rnorm(10000)
    ## 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_train,
      ";\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(beta, beta), col = "red")
  ## Compute variance of the estimates
  var_est <- round(apply(beta_hats, 2, var), digits = 3)
  return(list(optimal_s = shrinkage[min_id], var_est))
}

part1 <- make_ex_1()
```



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?

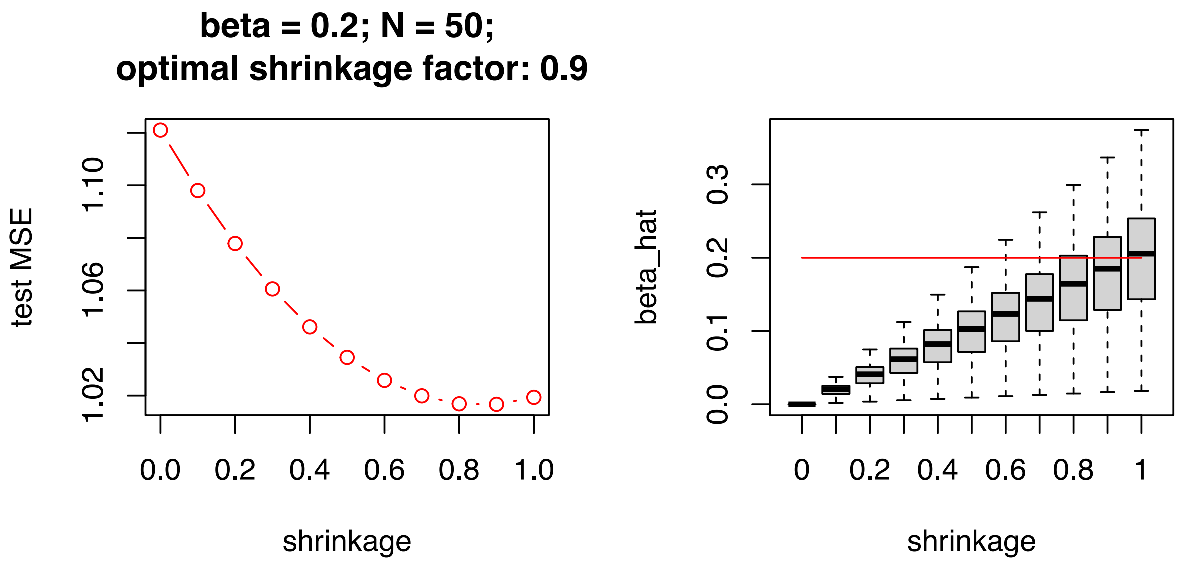
```
part2 <- make_ex_1(n_train = 100)
```



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 0.6 was optimal.

What if we increase the effect of X?

```r
part3 <- make_ex_1(beta = .2)
```

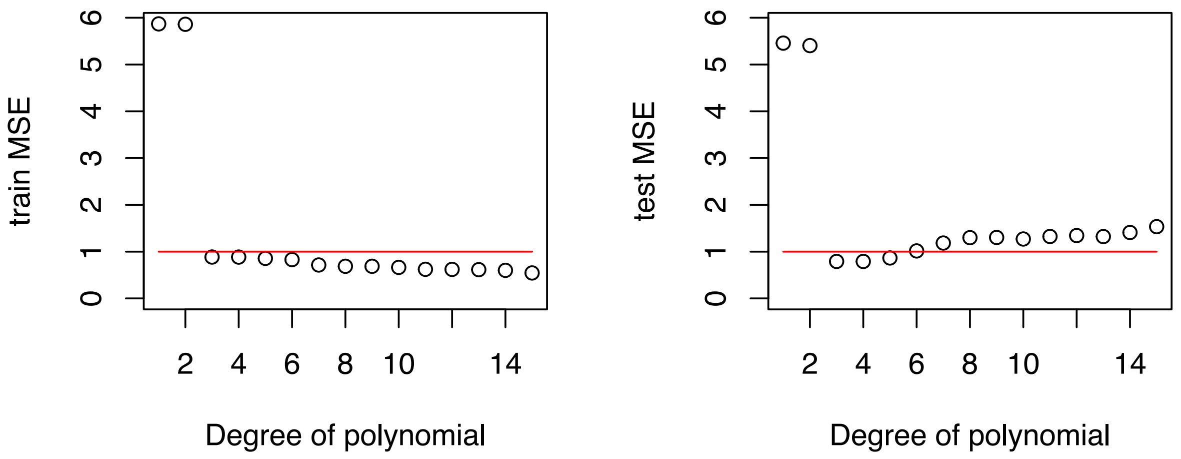


With larger effect size (i.e., larger β, so higher signal-to-noise ratio), shrinkage is still beneficial. Note that the variance of $\hat{\beta}$ does not change as a function of effect size. However, the shrinkage factor c has a stronger effect on larger coefficients, so we need less shrinkage (bias) to optimize prediction error. A shrinkage factor of 0.9 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 2.3: Under- and overfitting with polynomial regression

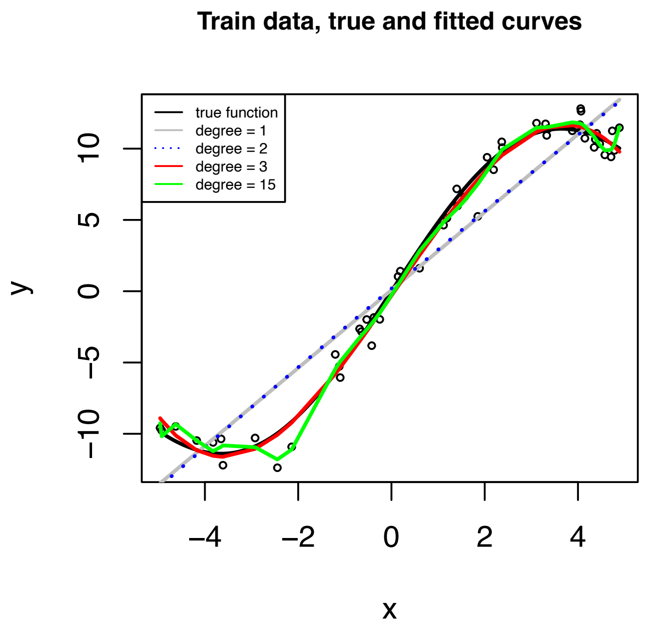
```r
set.seed(42)
n <- 50

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

degrees <- 1:15
train_err <- test_err <- rep(NA, length(degrees))
train_pred <- matrix(NA, nrow = length(degrees), ncol = n)

for (d in 1:15) {
  fit <- lm(y ~ poly(x, degree = d), data = train)
  train_pred[d,] <- predict(fit, newdata = train)
  test_pred <- predict(fit, newdata = test)
  test_err[d] <- mean((test$y - test_pred)^2)
  train_err[d] <- mean((train$y - train_pred[d,])^2)
}

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. The cubic (degree 3) seems to provide a good bias-variance trade-off: complex enough to capture the true function, not too flexible so it does not adjust to the sample at hand too much.

We plot the fitted curves against the training observations:

```r
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(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)
```

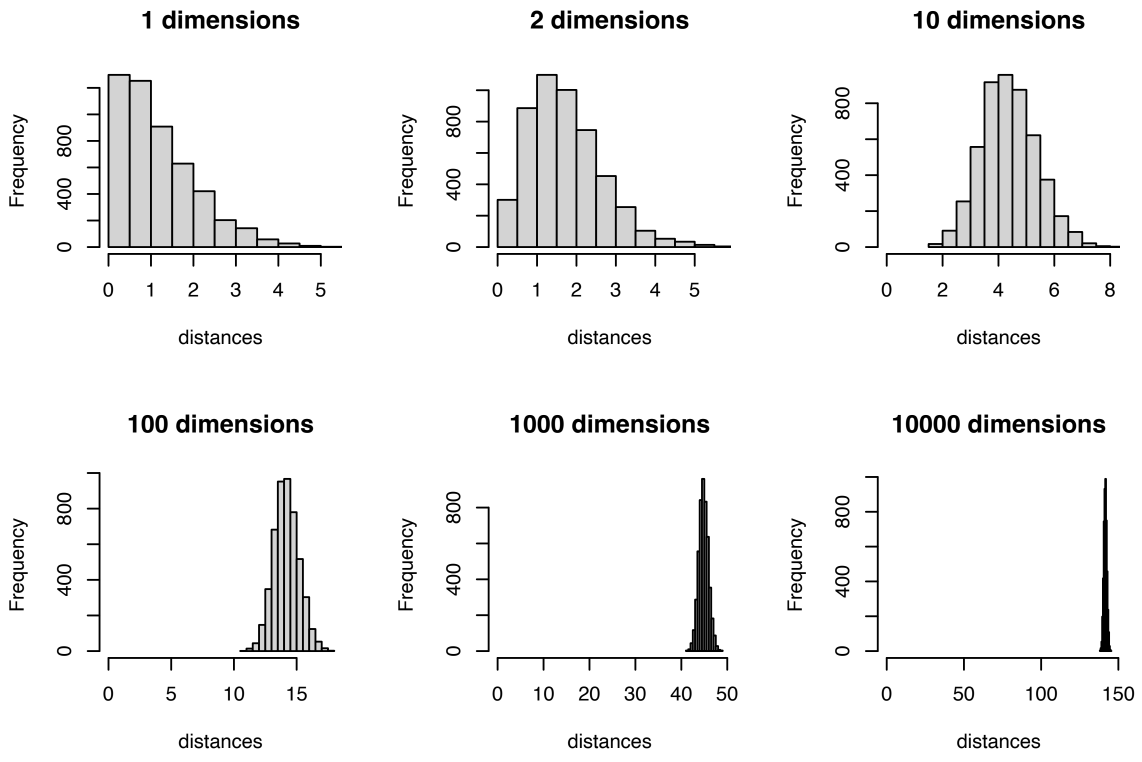


The linear and quadratic clearly stand out, failing to capture non-linearities. The other curves follow closely the true relationship, although the higher-order polynomials show aberrant behaviour at the boundaries and may adjust to the training data too much. Note that for these data, the erratic behavior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

Exercise 2.4: Curse of dimensionality

```r
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 assumes 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. It seems a bit like living in Leiden and that you would call anyone from the eastern part of New York would be your neighbour, but anyone from the western part of New York would not be your neighbour.

Exercise 2.5: Classifying digits

This exercise is about zipcode data; the aim is to recognize handwritten numbers 0 through 9. The observations are images of handwritten digits, automatically scanned from envelopes by the U.S. Postal Service. The images have been deslanted and size normalized, resulting in 16 x 16 grayscale images (Le Cun et al., 1990), yielding 256 predictor variables, reflecting the grayscale value for each pixel.

```
## Prepare training data
train <- read.table("zip.train", sep = " ")
dim(train)
```

```
## [1] 7291  258
```

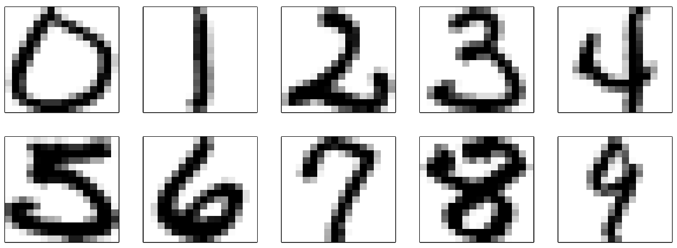
```
#head(train[1:10]) ## not run for space considerations
#colSums(is.na(train)) # last columns has only missings
train <- train[ , -258]

## Prepare test data
test <- read.table("zip.test", sep = " ")
#colSums(is.na(test)) ## not run for space considerations
dim(test)
```

```
## [1] 2007  257
```

```
#head(test[1:10]) ## not run for space considerations
```

This is what some of the images look like:

```
par(mfrow = c(2, 5), par(mar = c(1, 1, 0.1, 0.1))) # set graphical parameters
for (i in 0:9) {
  im <- matrix(as.numeric(train[train$V1 == i, 2:257][1, ]), nrow = 16, ncol = 16)
  image(t(apply(-im, 1, rev)), col = gray((0:63)/63), yaxt = "n", xaxt = "n")
  ## 63 is the minimum number of unique grayscale values observed for any variable
}
```



Select only 2s and 3s (`V1` is an indicator for digit):

```
train <- train[train$V1 %in% 2:3, ]
train$V1 <- train$V1 - 2
dim(train)
```

```
## [1] 1389  257
```

```r
test <- test[test$V1 %in% 2:3, ]
test$V1 <- test$V1 - 2
dim(test)
```

```
## [1] 364 257
```

```r
table(train$V1)
```

```
##
##   0   1
## 731 658
```

```r
table(test$V1)
```

```
##
##   0   1
## 198 166
```

Fit a GLM and generate predictions:

```r
lmod <- glm(V1 ~ ., data = train, family = "binomial")
```

```
## Warning: glm.fit: algorithm did not converge

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```r
#summary(lmod) ## not run for space considerations
y_hat_train <- predict(lmod, newdata = train, type = "response") > 0.5
y_hat_test <- predict(lmod, newdata = test, type = "response") > 0.5
train_mcr_lm <- mean((train$V1 != y_hat_train))
test_mcr_lm <- mean((test$V1 != y_hat_test))
train_mcr_lm
```

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

```r
test_mcr_lm
```

```
## [1] 0.05494505
```

Fit kNN and generate predictions:

```r
library("class")
ks <- 1:15
train_knn <- matrix(NA, nrow = nrow(train), ncol = length(ks))
test_knn <- matrix(NA, nrow = nrow(test), ncol = length(ks))
colnames(test_knn) <- colnames(train_knn) <- ks
for (i in 1:length(ks)) {
  train_knn[ , i] <- knn(train = train[ , -1], test = train[ , -1], cl = train$V1, k = i)
  test_knn[ , i] <- knn(train = train[ , -1], test = test[ , -1], cl = train$V1, k = i)
}
head(train_knn[ , 1]) ## predictions are generated as 1 and 2, need to subtract 1
```

```
## [1] 2 2 2 2 2 1
```

```
train_mcr_knn <- apply(train$V1 != train_knn-1, 2, mean)
test_mcr_knn <- apply(test$V1 != test_knn-1, 2, mean)
## Print train and test misclassification rates for each value of k
round(train_mcr_knn, digits = 4L)
```

```
##      1      2      3      4      5      6      7      8      9     10     11
## 0.0000 0.0065 0.0050 0.0065 0.0058 0.0072 0.0065 0.0086 0.0094 0.0094 0.0086
##     12     13     14     15
## 0.0079 0.0086 0.0094 0.0094
```

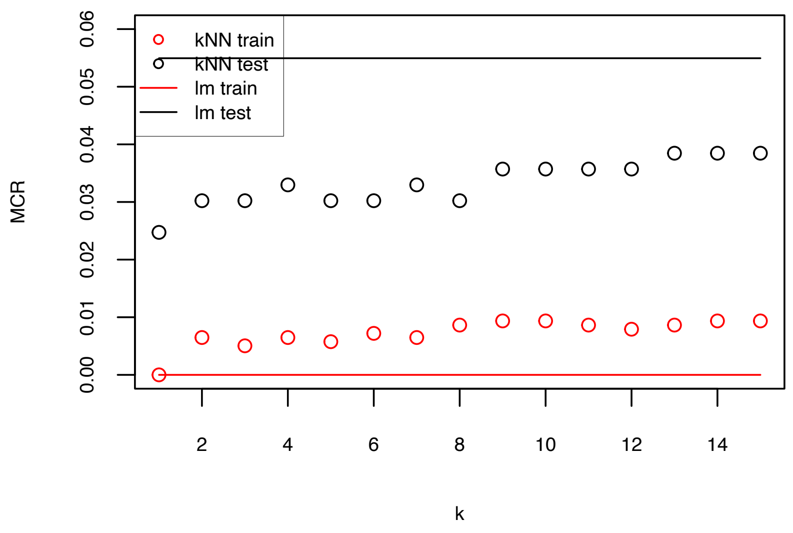
```
round(test_mcr_knn, digits = 4L)
```

```
##      1      2      3      4      5      6      7      8      9     10     11
## 0.0247 0.0302 0.0302 0.0330 0.0302 0.0302 0.0330 0.0302 0.0357 0.0357 0.0357
##     12     13     14     15
## 0.0357 0.0385 0.0385 0.0385
```

kNN outperforms logistic regression for all values of k on the test data.

Plot performance:

```
## Plot results
plot(ks, test_mcr_knn, ylim = c(0, 0.06), ylab = "MCR", xlab = "k", cex.lab = .7,
     cex.axis=.7)
legend("topleft", legend = c("kNN train", "kNN test", "lm train", "lm test"),
       col = c("red", "black", "red", "black"), cex = .7,
       pch = c(1, 1, NA, NA), lty = c(NA, NA, 1, 1), box.lwd = 0)
points(ks, train_mcr_knn, col = "red")
lines(x = ks, y = rep(test_mcr_lm, times = length(ks)))
lines(x = ks, y = rep(train_mcr_lm, times = length(ks)), col = "red")
```



The dots are for kNN, the lines are for the logistic regression. The training error (red) indicates logistic regression did better than kNN on training data, but it overfits: The difference in training (red) and test error (black) is largest for the logistic regression, and the test error (black) indicates kNN in fact generalizes better. For these data, the lowest test error is obtained for $k = 1$.