Module 2: Recommeded exercises

TMA4268 Statistical Learning V2020

Stefanie Muff, Department of Mathematical Sciences, NTNU

January xx and yy, 2020

Contents

Plan for the introductory lecture

First hour with all students:

14.15: getting to know group members, connecting to the screen, introduction
14.15-14.55: work with Problem 2 above, and if time look at Problem 1, Exam 2018 Problem 2, and Problem 3.
14.55-15.00: summing up, discussing in plenum.

15.00-15.15: break, refreshments

Second hour: divided content for three types of students

a) Students who have previously taken TMA4267:
   - may continue with Problem 1, Exam 2018 Problem 2 and Problem 3.
b) Students who have not taken before and will not take TMA4267 now
   - move to PartB and work first with random vectors, so covariance matrix and finally multivariate normal.
   - Student that have taken TMA4265 (Stocastic processes) might have covered parts of this before, and if they want can do as the students in a).
c) Students who are currently taking TMA4267 will this week/last week have covered Part A: random vectors and covariance matrix. They will in TMA4267 next cover Part A: multivariate normal distribution.
   - They can lood at Part A: random vectors and covariance matrix - as repetition,
   - and then look at Part B: multivariate normal as a warm up to TMA4267,
   - or may continue with the a) students.

Problem 1: Reflections and practicals

1. Describe a real-life application in which classification might be useful. Identify the response and the predictors. Is the goal inference or prediction?
2. Describe a real-life application in which regression might be useful. Identify the response and the predictors. Is the goal inference or prediction?
3. Take a look at Figure 2.9 in the book (p. 31).
   a. Will a flexible or rigid method typically have the highest test error?
   b. Does a small variance imply an overfit or rather an underfit to the data?
   c. Relate the problem of over-and underfitting to the bias-variance trade-off.
4. Exercise 7 from the book (p.53) slightly modified. The table below provides a training data set consisting of seven observations, two predictors and one qualitative response variable.

```
library(kableExtra)
knnframe = data.frame(x1 = c(3, 2, 1, 0, -1, 2, 1), x2 = c(3, 0, 1, 1,
    0, 1, 0), y = as.factor(c("A", "A", "A", "B", "B", "B", "B")))
print(knnframe)
```

```
##   x1 x2 y
## 1  3  3 A
## 2  2  0 A
## 3  1  1 A
## 4  0  1 B
## 5 -1  0 B
## 6  2  1 B
## 7  1  0 B
```

```
# kable(knnframe,format='html')
kable(knnframe)
```

x1	x2	y
3	3	A
2	0	A
1	1	A
0	1	B
-1	0	B
2	1	B
1	0	B

We wish to use this data set to make a prediction for Y when $X_1 = 1, X_2 = 2$ using the K-nearest neighbors classification method.

a. Compute the Euclidean distance between each observation and the test point, $X_1 = 1, X_2 = 2$.
b. What is our prediction with $K = 1$? Why?
c. What is our prediction with $K = 4$? Why?
d. If the Bayes decision boundary in this problem is highly non-linear, when would we expect the best value for K to be large or small? Why?
e. Install and load the `ggplot2` library:

```
install.packages(ggplot2)
library(ggplot2)
```

Plot the points in `R` using the functions `ggplot`, and `geom_points`.

f. Use the function `knn` from the `class` library to make a prediction for the test point using `k=1`. Do you obtain the same result as by hand?

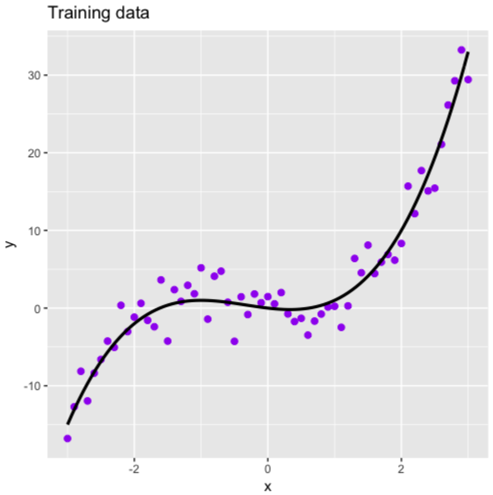

Figure 1: $f(x) = -x + x^2 + x^3$

g. Use the function `knn` to make a prediction for the test point using `k=4` and `k=7`.

Problem 2: Core concepts in statistical learning

(This was problem 1 from compulsory exercise 1 in 2018.)

We consider a regression problem, where the true underlying curve is $f(x) = -x + x^2 + x^3$ and we are considering $x \in [-3, 3]$ (see Figure 1).

This non-linear curve is only observed with added noise (either a random phenomenon, or unobservable variables influence the observations), that is, we observe $y = f(x) + \varepsilon$. In our example the error is sampled from $\varepsilon \sim N(0, 2^2)$.

In real life we are presented with a data set of pairs (x_i, y_i), $i = 1, \dots, n$, and asked to provide a prediction at a value x. We will use the method of K nearest neighbour regression to do this here.

We have a training set of $n = 61$ observations (x_i, y_i), $i = 1, \dots, n$. The KNN regression method provides a prediction at a value x by finding the closes K points and calculating the average of the observed y values at these points.

$$\hat{f}(x_0) = \frac{1}{K} \sum_{i \in \mathcal{N}_0} y_i$$

Given an integer K and a test observation x_0, the KNN regression first identifies the K points in the training data that are closest (Euclidean distance) to x_0, represented by $\mathcal{N}_0$. It then estimates the regression curve at x_0 as the average of the response values for the training observations in $\mathcal{N}_0$.

In addition we have a test set of $n = 61$ observations (at the same grid points as for the training set), but now with new observed values y.

We have considered $K = 1, \dots, 25$ in the KNN method. Our experiment has been repeated $M = 1000$ times (that is, M versions of training and test set).

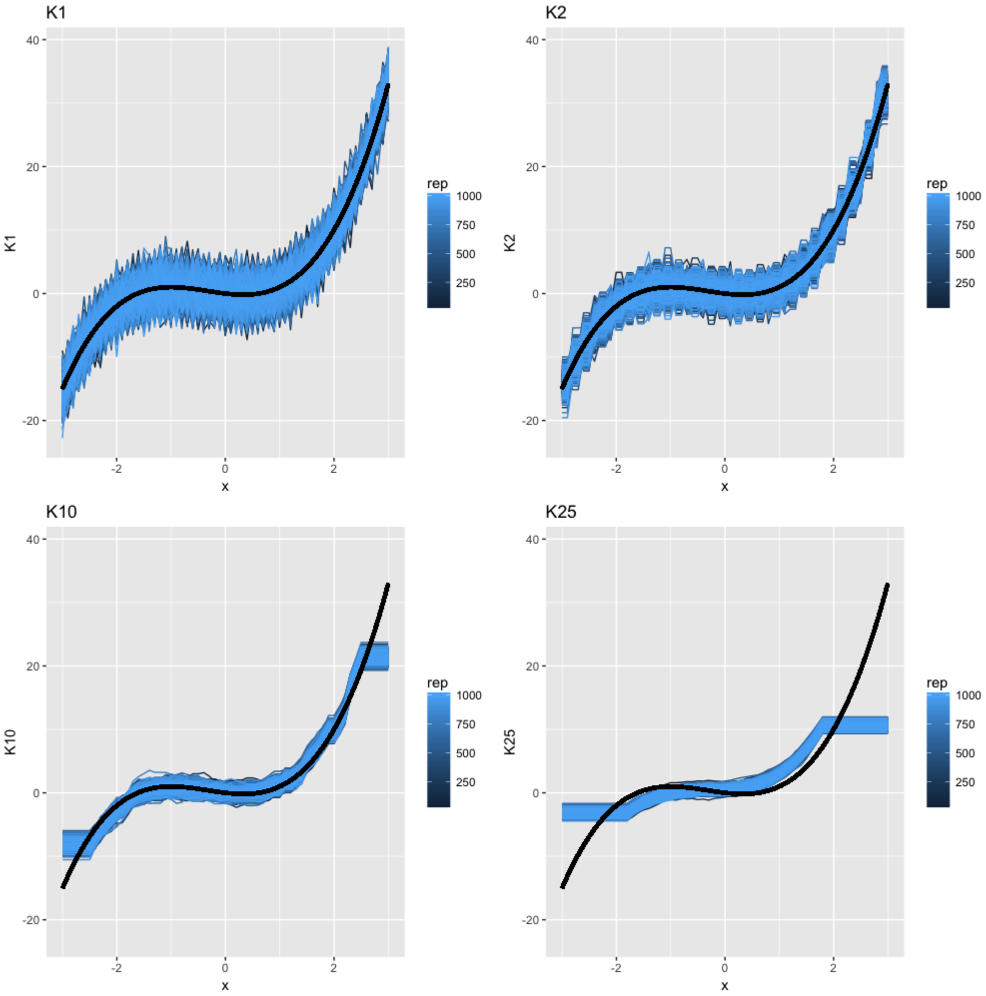

Figure 2: Figure 2

a) Training and test MSE

In the Figure 2 (above) you see the result of applying the KNN method with $K = 1, 2, 10, 25$ to our training data, repeated for M different training sets (blue lines). The black lines show the true underlying curve.

- Comment briefly on what you see.
- Does a high or low value of K give the most flexible fit?

In Figure 3 (below) you see mean-squared errors (mean of squared differences between observed and fitted values) for the training set and for the test set (right panel for one training and one test set, and left panel for M).

- Comment on what you see.
- What do you think is the "best" choice for K?

Remark: in real life we do not know the true curve, and need to use the test data to decide on model flexibility (choosing K).

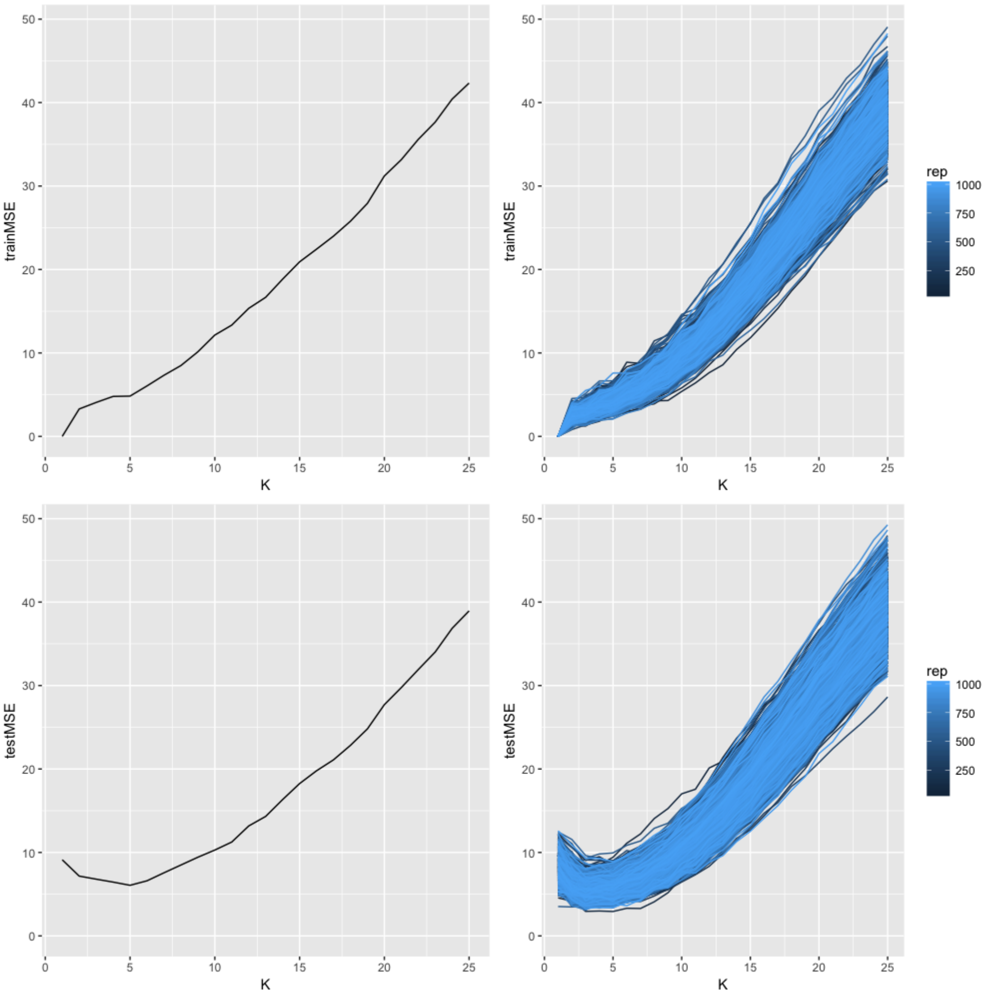

Figure 3: Mean squared errors for training and test sets

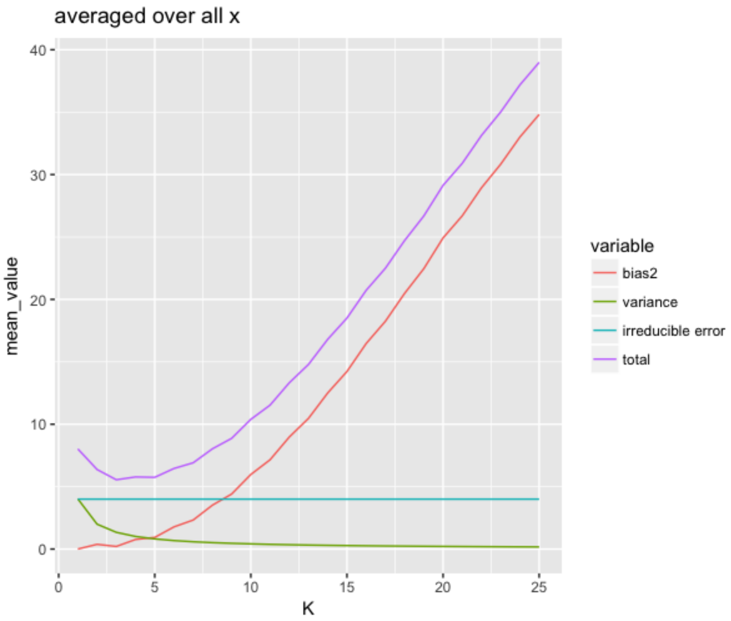

Figure 4: Figure 4

b) Bias-variance trade-off

Now we leave the real world situation, and assume we know the truth (this is to focus on bias-variance trade-off). You will not observe these curves in real life - but the understanding of the bias-variance trade-off is a core skill in this course!

In the Figure 4 (below) you see a plot of estimated squared bias, estimated variance, true irreducible error and the sum of these (labelled total) and averaged over all values of x

The the squared bias and the variance is calculated based on the predicted values and the "true" values (without the added noise) at each x.

- Explain how that is done. Hint: this is what the M repeated training data sets are used for.
- Focus on Figure 4. As the flexibility of the model increases (K decreases), what happens with
  - the squared bias,
  - the variance, and
  - the irreducible error?
- What would you recommend is the optimal value of K? Is this in agreement with what you found in a)?

Extra: We have chosen to also plot curves at four values of x - Figure 5 (below). Based on these four curves, that would you recommend is the optimal value of K? Is this in agreement with what you found previously (averaged over x)?

For completeness the R code used is given next (listed here with `M=100` but `M=1000` was used). You do not need to run the code, this is just if you have questions about how this was done.

```r
library(FNN)
library(ggplot2)
library(ggpubr)
library(reshape2)
library(dplyr)
```

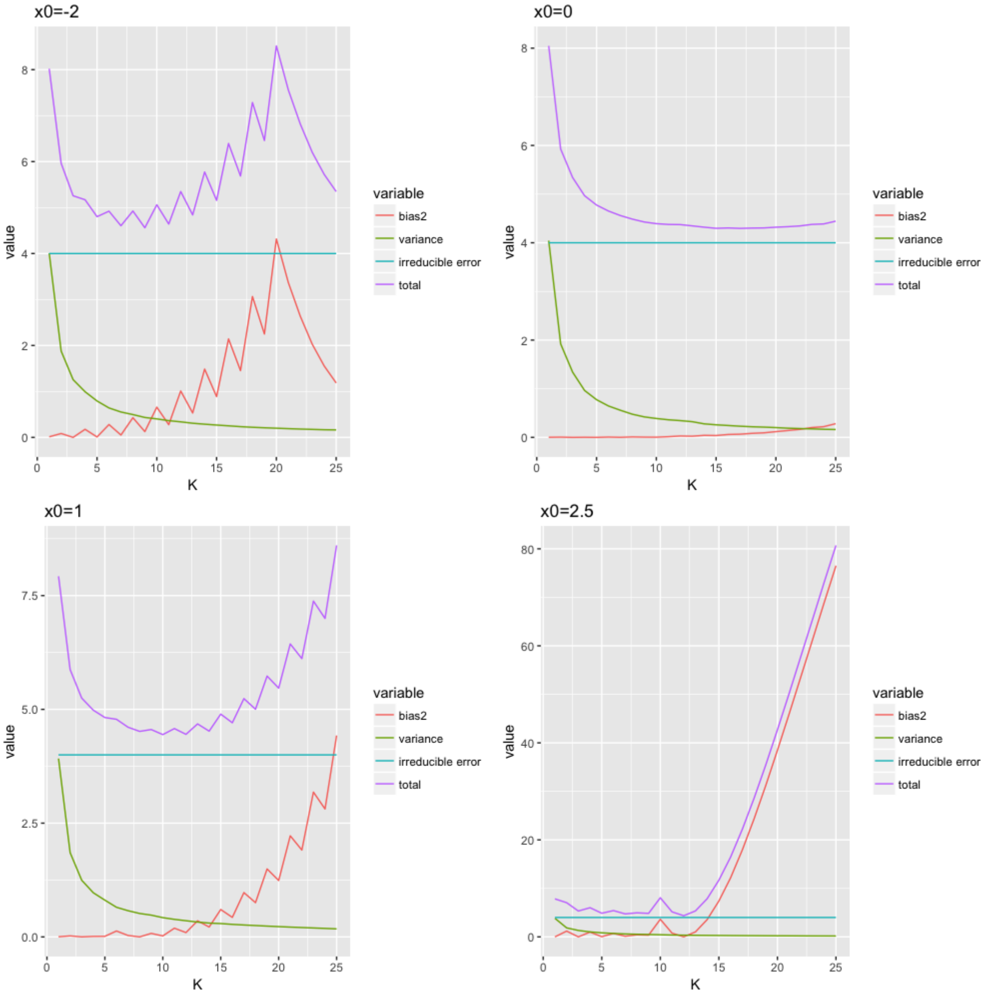

Figure 5: Figure 5

```r
maxK = 25
M = 1000  # repeated samplings, x fixed  - examples were run with M=1000
x = seq(-3, 3, 0.1)
dfx = data.frame(x = x)
truefunc = function(x) return(-x + x^2 + x^3)
true_y = truefunc(x)

set.seed(2)  # to reproduce
error = matrix(rnorm(length(x) * M, mean = 0, sd = 2), nrow = M, byrow = TRUE)
testerror = matrix(rnorm(length(x) * M, mean = 0, sd = 2), nrow = M,
    byrow = TRUE)
ymat = matrix(rep(true_y, M), byrow = T, nrow = M) + error
testymat = matrix(rep(true_y, M), byrow = T, nrow = M) + testerror

ggplot(data = data.frame(x = x, y = ymat[1, ]), aes(x, y)) + geom_point(col = "purple",
    size = 2) + stat_function(fun = truefunc, lwd = 1.1, colour = "black") +
    ggtitle("Training data")

predarray = array(NA, dim = c(M, length(x), maxK))
for (i in 1:M) {
    for (j in 1:maxK) {
        predarray[i, , j] = knn.reg(train = dfx, test = dfx, y = c(ymat[i,
            ]), k = j)$pred
    }
}
# first - just plot the fitted values - and add the true curve in
# black M curves and choose k=1,2,10,30 in KNN

# rearranging to get data frame that is useful
thislwd = 1.3
stackmat = NULL
for (i in 1:M) stackmat = rbind(stackmat, cbind(x, rep(i, length(x)),
    predarray[i, , ]))
colnames(stackmat) = c("x", "rep", paste("K", 1:maxK, sep = ""))
sdf = as.data.frame(stackmat)
yrange = range(apply(sdf, 2, range)[, 3:(maxK + 2)])
# making the four selected plots
p1 = ggplot(data = sdf, aes(x = x, y = K1, group = rep, colour = rep)) +
    scale_y_continuous(limits = yrange) + geom_line()
p1 = p1 + stat_function(fun = truefunc, lwd = thislwd, colour = "black") +
    ggtitle("K1")
p2 = ggplot(data = sdf, aes(x = x, y = K2, group = rep, colour = rep)) +
    scale_y_continuous(limits = yrange) + geom_line()
p2 = p2 + stat_function(fun = truefunc, lwd = thislwd, colour = "black") +
    ggtitle("K2")
p10 = ggplot(data = sdf, aes(x = x, y = K10, group = rep, colour = rep)) +
    scale_y_continuous(limits = yrange) + geom_line()
p10 = p10 + stat_function(fun = truefunc, lwd = thislwd, colour = "black") +
    ggtitle("K10")
p25 = ggplot(data = sdf, aes(x = x, y = K25, group = rep, colour = rep)) +
    scale_y_continuous(limits = yrange) + geom_line()
p25 = p25 + stat_function(fun = truefunc, lwd = thislwd, colour = "black") +
    ggtitle("K30")
ggarrange(p1, p2, p10, p25)

# calculating trainMSE and testMSE
trainMSE = matrix(ncol = maxK, nrow = M)
for (i in 1:M) trainMSE[i, ] = apply((predarray[i, , ] - ymat[i, ])^2,
    2, mean)
testMSE = matrix(ncol = maxK, nrow = M)
for (i in 1:M) testMSE[i, ] = apply((predarray[i, , ] - testymat[i, ])^2,
    2, mean)
# rearranging to get data frame that is useful
stackmat = NULL
for (i in 1:M) stackmat = rbind(stackmat, cbind(rep(i, maxK), 1:maxK,
    trainMSE[i, ], testMSE[i, ]))
colnames(stackmat) = c("rep", "K", "trainMSE", "testMSE")
```

```r
sdf = as.data.frame(stackmat)
yrange = range(sdf[, 3:4])
# plotting training and test MSE
p1 = ggplot(data = sdf[1:maxK, ], aes(x = K, y = trainMSE)) + scale_y_continuous(limits = yrange) +
    geom_line()
pall = ggplot(data = sdf, aes(x = K, group = rep, y = trainMSE, colour = rep)) +
    scale_y_continuous(limits = yrange) + geom_line()
testp1 = ggplot(data = sdf[1:maxK, ], aes(x = K, y = testMSE)) + scale_y_continuous(limits = yrange) +
    geom_line()
testpall = ggplot(data = sdf, aes(x = K, group = rep, y = testMSE, colour = rep)) +
    scale_y_continuous(limits = yrange) + geom_line()
ggarrange(p1, pall, testp1, testpall)

# calculating bias^2 and variance
meanmat = matrix(ncol = length(x), nrow = maxK)
varmat = matrix(ncol = length(x), nrow = maxK)
for (j in 1:maxK) {
    meanmat[j, ] = apply(predarray[, , j], 2, mean)  # we now take the mean over the M simulations - to mimic E and Var at each
    varmat[j, ] = apply(predarray[, , j], 2, var)
}
bias2mat = (meanmat - matrix(rep(true_y, maxK), byrow = TRUE, nrow = maxK))^2  #here the truth is finally used!

# preparing to plot
df = data.frame(rep(x, each = maxK), rep(1:maxK, length(x)), c(bias2mat),
    c(varmat), rep(4, prod(dim(varmat))))  #irr is just 4
colnames(df) = c("x", "K", "bias2", "variance", "irreducible error")  #suitable for plotting
df$total = df$bias2 + df$variance + df$`irreducible error`
hdf = melt(df, id = c("x", "K"))
# averaged over all x - to compare to train and test MSE
hdfmean = hdf %>% group_by(K, variable) %>% summarise(mean_value = mean(value))
ggplot(data = hdfmean[hdfmean[, 1] < 31, ], aes(x = K, y = mean_value,
    colour = variable)) + geom_line() + ggtitle("averaged over all x")

# extra: what about different values of x?
hdfatxa = hdf[hdf$x == -2, ]
hdfatxb = hdf[hdf$x == 0, ]
hdfatxc = hdf[hdf$x == 1, ]
hdfatxd = hdf[hdf$x == 2.5, ]
pa = ggplot(data = hdfatxa, aes(x = K, y = value, colour = variable)) +
    geom_line() + ggtitle("x0=-2")
pb = ggplot(data = hdfatxb, aes(x = K, y = value, colour = variable)) +
    geom_line() + ggtitle("x0=0")
pc = ggplot(data = hdfatxc, aes(x = K, y = value, colour = variable)) +
    geom_line() + ggtitle("x0=1")
pd = ggplot(data = hdfatxd, aes(x = K, y = value, colour = variable)) +
    geom_line() + ggtitle("x0=2.5")
ggarrange(pa, pb, pc, pd)
```

Problem 3: Theory and practice - MSEtrain, MSEtest, and bias-variance

We will now look closely into the simulations and calculations performed for the MSEtrain, MSEtest, and bias-variance trade-off in PartA.

- The simulations are based on $f(x) = x^2$ and normal noise with mean 0 and standard deviation 2 is added.
- x is on a 0.1 grid from -2 to 4 (61 values).
- Parametric models of different complexity are fitted - poly1-poly20.
- M=100 simulations are done.

The aim of this problem is to understand:

- trainMSE
- testMSE
- bias-variance trade-off

a) Problem set-up

- See the code below. Explain what is done. (You need not understand the code in detail.) Run the code.
- We will learn more about the `lm` function in M 3 - now just think of this as fitting a polynomial regression and predict gives the fitted curve in our grid points. `predarray` is just a way to save M simulations of 61 gridpoints in x and 20 polynomial models.

```r
library(ggplot2)
library(ggpubr)
set.seed(2)  # to reproduce

M = 100  # repeated samplings, x fixed
nord = 20  # order of polynoms


x = seq(-2, 4, 0.1)
truefunc = function(x) return(x^2)
true_y = truefunc(x)

error = matrix(rnorm(length(x) * M, mean = 0, sd = 2), nrow = M, byrow = TRUE)
ymat = matrix(rep(true_y, M), byrow = T, nrow = M) + error

predarray = array(NA, dim = c(M, length(x), nord))
for (i in 1:M) {
    for (j in 1:nord) {
        predarray[i, , j] = predict(lm(ymat[i, ] ~ poly(x, j, raw = TRUE)))
    }
}
# M matrices of size length(x) times nord first, only look at
# variablity in the M fits and plot M curves where we had 1

# for plotting need to stack the matrices underneath eachother and
# make new variable 'rep'
stackmat = NULL
for (i in 1:M) stackmat = rbind(stackmat, cbind(x, rep(i, length(x)),
    predarray[i, , ]))
# dim(stackmat)
colnames(stackmat) = c("x", "rep", paste("poly", 1:20, sep = ""))
sdf = as.data.frame(stackmat)  #NB have poly1-20 now - but first only use 1,2,20
# to add true curve using stat_function - easiest solution
true_x = x
yrange = range(apply(sdf, 2, range)[, 3:22])
p1 = ggplot(data = sdf, aes(x = x, y = poly1, group = rep, colour = rep)) +
    scale_y_continuous(limits = yrange) + geom_line()
p1 = p1 + stat_function(fun = truefunc, lwd = 1.3, colour = "black") +
    ggtitle("poly1")
p2 = ggplot(data = sdf, aes(x = x, y = poly2, group = rep, colour = rep)) +
    scale_y_continuous(limits = yrange) + geom_line()
p2 = p2 + stat_function(fun = truefunc, lwd = 1.3, colour = "black") +
    ggtitle("poly2")
p10 = ggplot(data = sdf, aes(x = x, y = poly10, group = rep, colour = rep)) +
    scale_y_continuous(limits = yrange) + geom_line()
```

```
p10 = p10 + stat_function(fun = truefunc, lwd = 1.3, colour = "black") +
    ggtitle("poly10")
p20 = ggplot(data = sdf, aes(x = x, y = poly20, group = rep, colour = rep)) +
    scale_y_continuous(limits = yrange) + geom_line()
p20 = p20 + stat_function(fun = truefunc, lwd = 1.3, colour = "black") +
    ggtitle("poly20")
ggarrange(p1, p2, p10, p20)
```

---

b) Train and test MSE

- First we produce predictions at each grid point based on our training data (`x` and `ymat`)
- but we also draw new observations to calculate testMSE - see `testymat`
- observe how trainMSE and testMSE is calculated
- run the code

```
set.seed(2)  # to reproduce

M = 100  # repeated samplings,x fixed but new errors
nord = 20
x = seq(-2, 4, 0.1)
truefunc = function(x) return(x^2)
true_y = truefunc(x)

error = matrix(rnorm(length(x) * M, mean = 0, sd = 2), nrow = M, byrow = TRUE)
testerror = matrix(rnorm(length(x) * M, mean = 0, sd = 2), nrow = M,
    byrow = TRUE)
ymat = matrix(rep(true_y, M), byrow = T, nrow = M) + error
testymat = matrix(rep(true_y, M), byrow = T, nrow = M) + testerror

predarray = array(NA, dim = c(M, length(x), nord))
for (i in 1:M) {
    for (j in 1:nord) {
        predarray[i, , j] = predict(lm(ymat[i, ] ~ poly(x, j, raw = TRUE)))
    }
}
trainMSE = matrix(ncol = nord, nrow = M)
for (i in 1:M) trainMSE[i, ] = apply((predarray[i, , ] - ymat[i, ])^2,
    2, mean)
testMSE = matrix(ncol = nord, nrow = M)
for (i in 1:M) testMSE[i, ] = apply((predarray[i, , ] - testymat[i, ])^2,
    2, mean)
```

- Then we plot train and testMSE - first for one train + test data set, then for 99 more.

```
library(ggplot2)
library(ggpubr)

# format suitable for plotting
stackmat = NULL
for (i in 1:M) stackmat = rbind(stackmat, cbind(rep(i, nord), 1:nord,
    trainMSE[i, ], testMSE[i, ]))
colnames(stackmat) = c("rep", "poly", "trainMSE", "testMSE")
```

```
sdf = as.data.frame(stackmat)
yrange = range(sdf[, 3:4])
p1 = ggplot(data = sdf[1:nord, ], aes(x = poly, y = trainMSE)) + scale_y_continuous(limits = yrange) +
    geom_line()
pall = ggplot(data = sdf, aes(x = poly, group = rep, y = trainMSE, colour = rep)) +
    scale_y_continuous(limits = yrange) + geom_line()
testp1 = ggplot(data = sdf[1:nord, ], aes(x = poly, y = testMSE)) + scale_y_continuous(limits = yrange)
    geom_line()
testpall = ggplot(data = sdf, aes(x = poly, group = rep, y = testMSE,
    colour = rep)) + scale_y_continuous(limits = yrange) + geom_line()
ggarrange(p1, pall, testp1, testpall)
```

- More plots: first boxplot and then mean for train and test MSE

```
library(reshape2)
df = melt(sdf, id = c("poly", "rep"))[, -2]
colnames(df)[2] = "MSEtype"
ggplot(data = df, aes(x = as.factor(poly), y = value)) + geom_boxplot(aes(fill = MSEtype))
```

```
trainMSEmean = apply(trainMSE, 2, mean)
testMSEmean = apply(testMSE, 2, mean)
meandf = melt(data.frame(cbind(poly = 1:nord, trainMSEmean, testMSEmean)),
    id = "poly")
ggplot(data = meandf, aes(x = poly, y = value, colour = variable)) +
    geom_line()
```

---

c) Bias and variance - we use the truth!

Finally, we want to see how the expected quadratic loss can be decomposed into

- irreducible error: $\text{Var}(\varepsilon) = 4$
- squared bias: difference between mean of estimated parametric model chosen and the true underlying curve (`truefunc`)
- variance: variance of the estimated parametric model

Notice that the test data is not used - only predicted values in each x grid point.

Study and run the code. Explain the plots produced.

```
meanmat = matrix(ncol = length(x), nrow = nord)
varmat = matrix(ncol = length(x), nrow = nord)
for (j in 1:nord) {
    meanmat[j, ] = apply(predarray[, , j], 2, mean)  # we now take the mean over the M simulations - to
    varmat[j, ] = apply(predarray[, , j], 2, var)
}
# nord times length(x)
bias2mat = (meanmat - matrix(rep(true_y, nord), byrow = TRUE, nrow = nord))^2  #here the truth is final
```

- Plotting the polys as a function of x

```
df = data.frame(rep(x, each = nord), rep(1:nord, length(x)), c(bias2mat),
    c(varmat), rep(4, prod(dim(varmat))))  #irr is just 1
colnames(df) = c("x", "poly", "bias2", "variance", "irreducible error")  #suitable for plotting
df$total = df$bias2 + df$variance + df$`irreducible error`
hdf = melt(df, id = c("x", "poly"))
```

```r
hdf1 = hdf[hdf$poly == 1, ]
hdf2 = hdf[hdf$poly == 2, ]
hdf10 = hdf[hdf$poly == 10, ]
hdf20 = hdf[hdf$poly == 20, ]

p1 = ggplot(data = hdf1, aes(x = x, y = value, colour = variable)) +
    geom_line() + ggtitle("poly1")
p2 = ggplot(data = hdf2, aes(x = x, y = value, colour = variable)) +
    geom_line() + ggtitle("poly2")
p10 = ggplot(data = hdf10, aes(x = x, y = value, colour = variable)) +
    geom_line() + ggtitle("poly10")
p20 = ggplot(data = hdf20, aes(x = x, y = value, colour = variable)) +
    geom_line() + ggtitle("poly20")
ggarrange(p1, p2, p10, p20)
```

- Now plotting effect of more complex model at 4 chosen values of x, compare to Figures in 2.12 on page 36 in ISL (our textbook).

```r
hdfatxa = hdf[hdf$x == -1, ]
hdfatxb = hdf[hdf$x == 0.5, ]
hdfatxc = hdf[hdf$x == 2, ]
hdfatxd = hdf[hdf$x == 3.5, ]
pa = ggplot(data = hdfatxa, aes(x = poly, y = value, colour = variable)) +
    geom_line() + ggtitle("x0=-1")
pb = ggplot(data = hdfatxb, aes(x = poly, y = value, colour = variable)) +
    geom_line() + ggtitle("x0=0.5")
pc = ggplot(data = hdfatxc, aes(x = poly, y = value, colour = variable)) +
    geom_line() + ggtitle("x0=2")
pd = ggplot(data = hdfatxd, aes(x = poly, y = value, colour = variable)) +
    geom_line() + ggtitle("x0=3.5")
ggarrange(pa, pb, pc, pd)
```

---

d) Repeat a-c

- Then try to change the true function `truefunc` to something else - mayby order 3? What does this do the the plots produced? Maybe you then also want to plot poly3?
- Also try to change the standard deviation of the noise added to the curve (now it is sd=2). What happens if you change this to sd=1 or sd=3?
- Or, change to the true function that is not a polynomial?

Problem 4: An exam problem from 2018

We have a univariate continuous random variable Y and a covariate x. Further, we have observed a training set of independent observation pairs $\{x_i, y_i\}$ for $i = 1, \ldots, n$.

Assume a regression model

$$Y_i = f(x_i) + \varepsilon_i$$

where f is the true regression function, and ε_i is an unobserved random variable with mean zero and constant variance σ^2 (not dependent on the covariate). Using the training set we can find an estimate of the regression function f, and we denote this by $\hat{f}$. We want to use $\hat{f}$ to make a prediction for a new observation (not dependent on the observations in the training set) at a covariate value x_0. The predicted response value is then $\hat{f}(x_0)$. We are interested in the error associated with this prediction.

Q5: Write down the definition of the expected test mean squared error (MSE) at x_0.

Q6: Derive the decomposition of the expected test MSE into three terms.

Q7: Explain with words how we can interpret the three terms.

Assume that we have a method to estimate the regression function, where this method has a tuning parameter that controls the complexity of the model and that a large value of the tuning parameter gives high model complexity.

Q8: Make a sketch of how the expected test MSE (at x_0) and the decomposition into three terms could look as a function of the tuning parameter.

This decomposition has played a central role in our course.

Q9: In your opinion, what is the most important implication of this decomposition? Answer with *only one* sentence.

Further reading/resources

- Videoes on YouTube by the authors of ISL, Chapter 2

R packages

If you want to look at the .Rmd file and **knit** it, you need to first install the following packages (only once).

```
install.packages("knitr")
install.packages("kableExtra")
install.packages("rmarkdown")
install.packages("devtools")
install.packages("ggplot2")
install.packages("ggpubr")
install.packages("dplyr")
install.packages("reshape2")
install.packages("ElemStatLearn")
install.packages("GGally")
install.packages("class")
install.packages("mvtnorm")
install.packages("MASS")
install.packages("car")
install.packages("faraway")
install.packages("reshape")
```

Acknowledgements

Thanks to Julia Debik for contributing to this module page.