

Module 2: Recommended Exercises

TMA4268 Statistical Learning V2020

Martina Hall, Michail Spitieris, Stefanie Muff, Department of Mathematical Sciences, NTNU
January 10, 2020

Theoretical exercises

Problem 1

Describe a real-life application in which classification might be useful. Identify the response and the predictors. Is the goal inference or prediction?

Problem 2

Describe a real-life application in which regression might be useful. Identify the response and the predictors. Is the goal inference or prediction?

Problem 3

Take a look at Figure 2.9 in the course book (p.31).

- Will a flexible or rigid method typically have the highest test error?
- Does a small variance imply an overfit or rather an underfit to the data?
- Relate the problem of over- and underfitting to the bias-variance trade-off.

Problem 4 – Exercise 2.4.9 from ISL textbook (modified)

This exercise involves the `Auto` dataset from the `ISLR` library. Load the data into your R session by running the following commands

```
library(ISLR)
data(Auto)
```

PS: if the `ISLR` package is not installed (library function gives error) you can install it by running `install.packages("ISLR")`.

- View the data. What are the dimensions of the data? Which predictors are quantitative and which are qualitative?
- What is the range (min, max) of each quantitative predictor? Hint: use the `range()` function. For more advanced users, check out `sapply()`.
- What is the mean and standard deviation of each quantitative predictor?
- Now, make a new dataset called `ReducedAuto` where you remove the 10th through 85th observations. What is the range, mean and standard deviation of the quantitative predictors in this reduced set?
- Using the full dataset, investigate the quantitative predictors graphically using a scatterplot. Do you see any strong relationships between the predictors? Hint: try out `ggpairs()`.

- f) Suppose we wish to predict gas milage (`mpg`) on the basis of the other variables (both quantitative and qualitative). Make some plots showing the relationships between `mpg` and the qualitative predictors (hint: `geom_boxplot()`). Which predictors would you consider helpful when predicting `mpg`?
- g) The correlation of two variables X and Y are defined as

$$\text{cor}(X, Y) = \frac{\text{cov}(X, Y)}{\sigma_X \sigma_Y}.$$

Both the correlation matrix and covariance matrix are easily assessed in R with the `cor()` and `cov()` functions. Use only the covariance matrix to find the correlation between `mpg` and `displacement`, `mpg` and `horsepower`, and `mpg` and `weight`. Do your results coincide with the correlation matrix you find using `cor(Auto[, quant])`?

```
quant = c(1, 3, 4, 5, 6, 7)
covMat = cov(Auto[, quant])
```

Problem 5 – Multivariate normal distribution

The pdf of a multivariate normal distribution is on the form

$$f(\mathbf{x}) = \frac{1}{(2\pi)^{p/2} |\Sigma|} \exp\left\{-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu})^T \Sigma^{-1}(\mathbf{x} - \boldsymbol{\mu})\right\},$$

where \mathbf{x} is a random vector of size $p \times 1$, $\boldsymbol{\mu}$ is the mean vector of size $p \times 1$ and Σ is the covariance matrix of size $p \times p$.

- a) Use the `mvnrm()` function from the `MASS` library to simulate 1000 values from multivariate normal distributions with
- b)

$$\boldsymbol{\mu} = \begin{pmatrix} 2 \\ 3 \end{pmatrix} \quad \text{and} \quad \Sigma = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix},$$

ii)

$$\boldsymbol{\mu} = \begin{pmatrix} 2 \\ 3 \end{pmatrix} \quad \text{and} \quad \Sigma = \begin{pmatrix} 1 & 0 \\ 0 & 5 \end{pmatrix},$$

iii)

$$\boldsymbol{\mu} = \begin{pmatrix} 2 \\ 3 \end{pmatrix} \quad \text{and} \quad \Sigma = \begin{pmatrix} 1 & 2 \\ 2 & 5 \end{pmatrix},$$

iv)

$$\boldsymbol{\mu} = \begin{pmatrix} 2 \\ 3 \end{pmatrix} \quad \text{and} \quad \Sigma = \begin{pmatrix} 1 & -2 \\ -2 & 1 \end{pmatrix}.$$

- b) Plot the four sets of simulated values. Can you see which plot belongs to which distribution?

Problem 6: Theory and practice - training and test MSE; bias-variance

We will now look closely into the simulations and calculations performed for the training error (MSE_{train}), test error (MSE_{test}), and the bias-variance trade-off in lecture 1 of module 2.

Simulation setup:

- True function $f(x) = x^2$ with normal noise $\varepsilon \sim N(0, 2^2)$.
- $x = -2.0, -1.9, \dots, 4.0$ (grid with 61 values).
- Parametric models are fitted (polynomials of degree 1 to degree 20).
- M=100 simulations.

a) Problem set-up

- Look at the code below and run it yourself. Explain what is done (you do not need not understand the code in detail).
- We will learn more about the `lm` function in M3 - 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.

```
library(ggplot2)
library(ggpubr)
set.seed(2) # to reproduce
M = 100 # repeated samplings, x fixed
nord = 20 # order of polynoms
x = seq(from = -2, to = 4, by = 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
# variability 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)

```

- What do you observe in this plot? Which polynomial fits the best to the true curve?

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(from = -2, to = 4, by = 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)
testMSE = matrix(ncol = nord, nrow = M)
for (i in 1:M) {
  trainMSE[i, ] = apply((predarray[i, , ] - ymat[i, ])^2, 2, mean)
  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()

```

- Which value of the polynomial gives the smallest mean testMSE?
- Which gives the smallest mean trainMSE?
- Which would you use to predict a new value of y ?

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"))
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).

```

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)

```

- Study the final plot you produced: when the flexibility increases (poly increase), what happens with
 - i) the squared bias,
 - ii) the variance,
 - iii) the irreducible error?

d) Repeat a-c

- Then try to change the true function `truefunc` to something else - maybe order 3? What does this do to 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 $\text{sd}=2$). What happens if you change this to $\text{sd}=1$ or $\text{sd}=3$?
 - Or, change to the true function that is not a polynomial?
-

Acknowledgements

We thank Mette Langaas and her PhD students (in particular Julia Debik) from 2018 and 2019 for building up the original version of this exercise sheet.