# Solution to Series 5

**1.** **a)** Invertibility and representation formula of the inverse follow from the following calculation:

$$\begin{aligned}
& (A - \mathbf{a}\mathbf{b}^{\mathbf{T}}) \cdot \left[ A^{-1} + \frac{1}{1 - \mathbf{b}^{\mathbf{T}} A^{-1}\mathbf{a}} \cdot A^{-1}\mathbf{a}\mathbf{b}^{\mathbf{T}} A^{-1} \right] \\
= \; & \mathrm{Id} - \mathbf{a}\mathbf{b}^{\mathbf{T}} A^{-1} + \frac{1}{1 - \mathbf{b}^{\mathbf{T}} A^{-1}\mathbf{a}} \left( \mathbf{a}\mathbf{b}^{\mathbf{T}} A^{-1} - \mathbf{a} \left( \mathbf{b}^{\mathbf{T}} A^{-1}\mathbf{a} \right) \mathbf{b}^{\mathbf{T}} A^{-1} \right) \\
= \; & \mathrm{Id} - \mathbf{a}\mathbf{b}^{\mathbf{T}} A^{-1} + \frac{1}{1 - \mathbf{b}^{\mathbf{T}} A^{-1}\mathbf{a}} (1 - \mathbf{b}^{\mathbf{T}} A^{-1}\mathbf{a}) \mathbf{a}\mathbf{b}^{\mathbf{T}} A^{-1} = \mathrm{Id}.
\end{aligned}$$

**b)** Starting as in the hints given, we proceed as follows:

$$\begin{aligned}
\hat{\beta}^{(-\mathbf{i})} &= (A - \mathbf{x_i}\mathbf{x_i^T})^{-1}(\mathbf{c} - y_i \mathbf{x_i}) \\
&= A^{-1}\mathbf{c} - y_i A^{-1}\mathbf{x_i} + \frac{1}{1 - \mathbf{x_i^T} A^{-1}\mathbf{x_i}} A^{-1}\mathbf{x_i}\mathbf{x_i^T} A^{-1}(\mathbf{c} - y_i \mathbf{x_i}) \\
&= A^{-1}\mathbf{c} - y_i A^{-1}\mathbf{x_i} + \frac{A^{-1}\mathbf{x_i}(\mathbf{x_i^T} A^{-1}\mathbf{c})}{1 - \mathbf{x_i^T} A^{-1}\mathbf{x_i}} - \frac{A^{-1}\mathbf{x_i}(\mathbf{x_i^T} A^{-1}\mathbf{x_i}) y_i}{1 - \mathbf{x_i^T} A^{-1}\mathbf{x_i}} \\
&= \hat{\beta} - y_i A^{-1}\mathbf{x_i} \left( 1 + \frac{\mathbf{x_i^T} A^{-1}\mathbf{x_i}}{1 - \mathbf{x_i^T} A^{-1}\mathbf{x_i}} \right) + \mathbf{x_i^T}\hat{\beta} A^{-1}\mathbf{x_i} \frac{1}{1 - \mathbf{x_i^T} A^{-1}\mathbf{x_i}} \; .
\end{aligned}$$

And therefore:

$$\begin{aligned}
\hat{\beta}^{(-\mathbf{i})} - \hat{\beta} &= -\frac{(X^T X)^{-1}\mathbf{x_i}}{1 - \mathbf{x_i^T} A^{-1}\mathbf{x_i}} (y_i - \mathbf{x_i^T}\hat{\beta}) \\
&= -\frac{(y_i - \mathbf{x_i^T}\hat{\beta})}{1 - S_{ii}} (X^T X)^{-1}\mathbf{x_i},
\end{aligned}$$

where $S_{ii} = \mathbf{x_i^T}(X^T X)^{-1}\mathbf{x_i}$ is the diagonal element of the hat-matrix $S$ in the multiple-linear-regression case.

**c)** Finally we conclude:

$$\begin{aligned}
y_i - \mathbf{x_i^T}\hat{\beta}^{(-\mathbf{i})} &= y_i - \mathbf{x_i^T} \left[ \hat{\beta} - \frac{(y_i - \mathbf{x_i^T}\hat{\beta})}{1 - S_{ii}} (X^T X)^{-1}\mathbf{x_i} \right] \\
&= y_i - \mathbf{x_i^T}\hat{\beta} + \frac{S_{ii}}{1 - S_{ii}} (y_i - \mathbf{x_i^T}\hat{\beta}) \\
&= \left( 1 + \frac{S_{ii}}{1 - S_{ii}} \right) (y_i - \mathbf{x_i^T}\hat{\beta}) \\
&= \frac{1}{1 - S_{ii}} (y_i - \mathbf{x_i^T}\hat{\beta}).
\end{aligned}$$

By applying the square to both sides of the equation and averaging over $i$, the desired representation formula for the CV score now follows.

**2.** **a)** Read in and sort the data set for further analysis:

```
> diabetes <-
    read.table("http://stat.ethz.ch/Teaching/Datasets/diabetes2.dat",
               header = TRUE)
> reg <- diabetes[, c("Age", "C.Peptide")]
> names(reg) <-     c("x",   "y")
> reg <- reg[sort.list(reg$x), ]
```

We use a utility function for leave-one-out (LOO) cross-validation:

```
> ##' Calculates the LOO CV score for given data and regression prediction function
> ##'
> ##' @param reg.data: regression data; data.frame with columns 'x', 'y'
> ##' @param reg.fcn:  regr.prediction function; arguments:
> ##'                    reg.x: regression x-values
> ##'                    reg.y: regression y-values
> ##'                    x:     x-value(s) of evaluation point(s)
> ##'                  value: prediction at point(s) x
> ##' @return LOOCV score
> loocv <- function(reg.data, reg.fcn)
 {
   ## Help function to calculate leave-one-out regression values
   loo.reg.value <- function(i, reg.data, reg.fcn)
     return(reg.fcn(reg.data$x[-i], reg.data$y[-i], reg.data$x[i]))

   ## Calculate LOO regression values using the help function above
   n <- nrow(reg.data)
   loo.values <- sapply(1:n, loo.reg.value, reg.data, reg.fcn)

   ## Calculate and return MSE
   mean((reg.data$y - loo.values)^2)
 }
```

We first plot the data to guess a good bandwidth ($h = 4$; plot not shown here, it is the same as in Figure 3.1 of the lecture notes), then define a regression function that can be used with `loocv` defined above.

```
> plot(reg$x, reg$y)
> h <- 4
> reg.fcn.nw <- function(reg.x, reg.y, x)
   ksmooth(reg.x, reg.y, x.point = x, kernel = "normal", bandwidth = h)$y
> (cv.nw <- loocv(reg, reg.fcn.nw))
[1] 0.3905108
```

We calculate the hat matrix "manually" in order to calculate the degrees of freedom; this is the smoothing parameter used for other regression estimators:

```
> n <- nrow(reg)
> Id <- diag(n)
> S.nw <- matrix(0, n, n)
> for (j in 1:n)
   S.nw[, j] <- reg.fcn.nw(reg$x, Id[, j], reg$x)
> (df.nw <- sum(diag(S.nw)))
[1] 4.45845
>
```

We also do the calculation of the CV value with the hat matrix:

```
> y.fit.nw <- reg.fcn.nw(reg$x, reg$y, reg$x)
> (cv.nw.hat <- mean(((reg$y - y.fit.nw)/(1 - diag(S.nw)))^2))
[1] 0.3905108
```

Moreover, we can also simply use `hatMat` from the package `sfsmisc`:

```
> library(sfsmisc)
> #degrees of freedom
> hatMat(reg$x,trace=TRUE,pred.sm=reg.fcn.nw,x=reg$x)
[1] 4.45845
> #CV value
> S.nw.hatMat <- hatMat(reg$x,trace=FALSE,pred.sm=reg.fcn.nw,x=reg$x)
> (cv.nw.hatMat <- mean(((reg$y - y.fit.nw)/(1 - diag(S.nw.hatMat)))^2))
[1] 0.3905108
```

**b)** Local polynomial ("lp") regression from `loess`:

```
> reg.fcn.lp <- function(reg.x, reg.y, x) {
   lp.reg <- loess(reg.y ~ reg.x, enp.target = df.nw, surface = "direct")
   predict(lp.reg, x)
 }
> (cv.lp <- loocv(reg, reg.fcn.lp))
[1] 0.3849359
```

Again, we also calculate the CV value with the hat matrix constructed "manually":

```
> n <- nrow(reg)
> Id <- diag(n)
> S.lp <- matrix(0, n, n)
> for (j in 1:n)
   S.lp[, j] <- reg.fcn.lp(reg$x, Id[, j], reg$x)
> y.fit.lp <- reg.fcn.lp(reg$x, reg$y, reg$x)
> (cv.lp.hat <- mean(((reg$y - y.fit.lp)/(1 - diag(S.lp)))^2))
[1] 0.3849359
```

And once more, we also compute the CV value using `hatMat`:

```
> S.lp.hatMat <- hatMat(reg$x,trace=FALSE,pred.sm=reg.fcn.lp,x=reg$x)
> (cv.lp.hatMat <- mean(((reg$y - y.fit.lp)/(1 - diag(S.lp.hatMat)))^2))
[1] 0.3849359
```

Note that for both the kernel and the local polynomial regression, the alternative calculation using the hat matrix also gives the right result here. However, it is not known whether this is always the case for kernel or local polynomial regression (see Exercise 1 of this series).

**c)** Smoothing spline ("ss") regression from `smooth.spline` with fixed degrees of freedom. We begin by looking at the internally calculated CV value:

```
> est.ss <- smooth.spline(reg$x, reg$y, cv = TRUE, df = df.nw)
> est.ss$cv.crit
[1] 0.3886042
```

We then use the same smoothing parameter `spar` for our own calculations of the CV value:

```
> reg.fcn.ss <- function(reg.x, reg.y, x)
   {
     ss.reg <- smooth.spline(reg.x, reg.y, spar = est.ss$spar)
     predict(ss.reg, x)$y
   }
> (cv.ss <- loocv(reg, reg.fcn.ss))
[1] 0.3880219
```

Alternative calculation using the hat-matrix computed "manually":

```
> n <- nrow(reg)
> Id <- diag(n)
> S.ss <- matrix(0, n, n)
> for (j in 1:n)
   S.ss[, j] <- reg.fcn.ss(reg$x, Id[, j], reg$x)
> y.fit.ss <- reg.fcn.ss(reg$x, reg$y, reg$x)
> (cv.ss.hat <- mean(((reg$y - y.fit.ss)/(1 - diag(S.ss)))^2))
[1] 0.3886042
```

And alternative calculation using `hatMat`:

```
> S.ss.hatMat <- hatMat(reg$x,trace=FALSE,pred.sm=reg.fcn.ss,x=reg$x)
> (cv.ss.hatMat <- mean(((reg$y - y.fit.ss)/(1 - diag(S.ss.hatMat)))^2))
[1] 0.3886042
```

Note that there is a slight discrepancy between the CV score computed using `loocv` and the rest of the CV scores. In theory, all these values should be identical. However, the difference is caused by the way in which the degrees of freedoms are specified in `smooth.spline`, i.e., the `spar` parameter is internally converted to the $\lambda$ value and this conversion depends on the data. In each run of the CV (in `loocv`), the training data is slightly different, which means that a different $\lambda$ is used each time.

**d)** Smoothing spline regression with optimized degrees of freedom:

```
> smooth.spline(reg$x, reg$y, cv = TRUE)$cv.crit
[1] 0.3828729
```

**e)** Constant fit ("CF"):

```
> reg.fcn.cf <- function(reg.x, reg.y, x) mean(reg.y)
> (cv.cf <- loocv(reg, reg.fcn.cf))
[1] 0.531576
```

**f)** If the quality of a method is judged by cross-validation, the cross-validation mimics the error which the method is expected to produce on new, independent data. We leave out a point and apply the method independent of this point to predict its response. This is no longer true, if the method includes an optimal choice of a parameter by optimization of the cross-validation score (as method no. 4 does), because then the outcome depends on *all* cross-validations, and therefore it is no longer independent on the point left out at the moment. Thus it can be expected, that the resulting cross-validation score is over-optimistic. So we would conclude that local polynomial regression is most adequate in this task although differences to smoothing splines and kernel regression are small. Our constant fit, however, performed worst.