Statistical computing MATH10093
Computer lab 6
Solutions

Finn Lindgren

6/3/2019

Summary

In this lab session you will develop code for cross validated model selection for basic climate estimation in a part of Scotland. You will not hand in anything, but you should keep your code script file for later use.

1. Complete lab 5 by reading the solutions!
2. Initialise lab 6: Open `RStudio` and either
   (a) Open the relevant existing project for the lab, if you have one, or
   (b) Create a new project for the folder where you want to store the lab files.
   (c) Copy the `TMINallobs.csv` and `TMINalltest.csv` into the project folder, or a subdirectory `data/` of that folder.
   (d) Download and save the `lab06code.R` file to the same folder, and read the data (leave out `data/` if you didn't use a subfolder for the data):

```
source("lab06code.R")
TMINallobs <- read.csv(file = "data/TMINallobs.csv",
                       header = TRUE,
                       stringsAsFactors = FALSE)
TMINalltest <- read.csv(file = "data/TMINalltest.csv",
                        header = TRUE,
                        stringsAsFactors = FALSE)
```

3. Write a function `cvk_define_splits(N, K)` that takes inputs
   `N`: The number of data points in the Training&Validation set
   `K`: The number of "splits" or "folds" for cross validation

The output should be a numeric vector of length N, containing indices $1, \ldots, K$ defining a random splitting into K subsets. The size of each subset, N_k, should be equal to N/K if N is divisible by K. If N is not divisible by K they should have nearly equal size, for example

$$N_k = \lceil kN/K \rceil - \lceil (k-1)N/K \rceil, \quad k = 1, \ldots, K.$$

The `floor()` and `ceiling()` functions implement $\lfloor \cdot \rfloor$ and $\lceil \cdot \rceil$. To ensure that all $N_k \geq 1$, we require $K \leq N$.

Theory task: Prove that this choice of $N_1, \ldots, N_K$ sums to N.

Note: this function should be called only once in any cross validation procedure, since different random numbers will be generated each time it is called! The random numbers can be reset using the `set.seed()` function, but since that is a deterministic operation, that function should normally also be called only once (it is useful to call it once at the beginning of an R script to make the results repeatable and debugging easier).

Optional: Make the code give an error and helpful error message if $K > N$ (with the `stop()` function).

```r
#Solution:
cvk_define_splits <- function(N, K) {
  if (K > N) {
    stop("K-fold cross validation cannot have K > N")
  }
  sub_size <- ceiling((1:K) * N / K) - ceiling((0:(K-1)) * N / K)
  sample(rep(1:K, times = sub_size), size = N, replace = FALSE)
}
```

The output should look something like this (with some random permutation)

```r
# Should have 3 ones, 2 twos, 3 threes, and 2 fours:
cvk_define_splits(10, 4)

##  [1] 3 4 3 4 1 1 1 2 3 2

# Error message on invalid inputs:
cvk_define_splits(N = 4, K = 10)

## Error in cvk_define_splits(N = 4, K = 10):  K-fold cross validation
cannot have K > N
```

4. Write a function `cvk_split(data, splits, k)` that takes inputs

`data`: A data.frame with one observation per row

`splits`: The index vector defining the K data subsets, as produced by `cvk_define_splits`

`k`: The index $k \in 1, 2, \ldots, K$ for the subset to be used as validation set

The output should be a `list` with two named elements:

`train`: A data frame with all rows of `data` in subsets $j \neq k$

`valid`: A data frame with all rows of `data` in the kth subset

```
# Solution:
cvk_split <- function(data, splits, k) {
  list(train = data[splits != k, , drop = FALSE],
       valid = data[splits == k, , drop = FALSE])
}
# drop=FALSE is a safety feature to make sure the result is still a
# data.frame object. It's not usually needed when doing row indexing
# and the input has multiple columns, but it makes the code more robust.
```

Test the function:

```
mysplits <- cvk_define_splits(nrow(TMINallobs), 10)
data1 <- cvk_split(TMINallobs, mysplits, 1)
```

Compare the full data with the subsample:

```
# In R, to view the first rows of the data frames:
head(TMINallobs)
head(data1$train)
```

```
# In Rmd or Rnw, with library(xtable) and code chunk option results="asis":
print(xtable(head(TMINallobs)), size = "\\scriptsize")
```

	ID	Year	Month	Element	Day	Value	DecYear	Latitude	Longitude	Elevation	Name
1	UKE00105875	1960	1	TMIN	2	-6.70	1960.00	57.04	-3.22	283	BALMORAL
2	UKE00105875	1960	1	TMIN	3	-3.90	1960.01	57.04	-3.22	283	BALMORAL
3	UKE00105875	1960	1	TMIN	7	-3.30	1960.02	57.04	-3.22	283	BALMORAL
4	UKE00105875	1960	1	TMIN	9	-1.10	1960.02	57.04	-3.22	283	BALMORAL
5	UKE00105875	1960	1	TMIN	10	-7.20	1960.02	57.04	-3.22	283	BALMORAL
6	UKE00105875	1960	1	TMIN	11	0.60	1960.03	57.04	-3.22	283	BALMORAL

```
print(xtable(head(data1$train)), size = "\\scriptsize")
```

	ID	Year	Month	Element	Day	Value	DecYear	Latitude	Longitude	Elevation	Name
2	UKE00105875	1960	1	TMIN	3	-3.90	1960.01	57.04	-3.22	283	BALMORAL
3	UKE00105875	1960	1	TMIN	7	-3.30	1960.02	57.04	-3.22	283	BALMORAL
5	UKE00105875	1960	1	TMIN	10	-7.20	1960.02	57.04	-3.22	283	BALMORAL
6	UKE00105875	1960	1	TMIN	11	0.60	1960.03	57.04	-3.22	283	BALMORAL
7	UKE00105875	1960	1	TMIN	13	0.00	1960.03	57.04	-3.22	283	BALMORAL
8	UKE00105875	1960	1	TMIN	15	-2.20	1960.04	57.04	-3.22	283	BALMORAL

5. Look at the file `lab06code.R` and familiarise yourself with the function `train_and_validate()` and the other functions, that are all related to Coursework A. Check that the following gives the expected output (if you recall the approximate scores from Q1 in Coursework A):

```
scores <-
  train_and_validate(TMINallobs,
                     TMINalltest,
                     Value ~ Longitude + Latitude +
                             Elevation + I(DecYear - Year))
mean_score(scores, by.ID = FALSE)

##         SE       DS     Brier
## 1 21.82028 4.084626 0.1450926
```

6. Write a function `cvk_do_all(data, splits, formula)` that calls `train_and_validate()` for each of the K splits and collects the results. The inputs should be:

`data`: The data frame to be split into Training and Validation subsets

`splits`: The splitting information, e.g. as produced by `cvk_define_splits`

`formula`: A model formula suitable for `lm()`

The output should be a `data.frame` with K cross validated scores in each column for columns named `SE`, `DS`, and `Brier`.
Hints:
To find K from the `splits` parameter, use `K <- max(splits)`
There are many ways to iterate over the k-values. One option is a `for`-loop:

```
results <- ... # preallocate storage for the results
for (k in 1:K) {
  # do something and store in results, appropriately indexed by k
}
# Post-process the results
```

```
# Solution:
cvk_do_all <- function(data, splits, formula) {
  K <- max(splits)
  S_hat <- data.frame(SE = numeric(K),
                      DS = numeric(K),
                      Brier = numeric(K))
  for (k in 1:K) {
    data_k <- cvk_split(data, splits, k)
    scores_k <- train_and_validate(train = data_k$train,
                                   valid = data_k$valid,
                                   formula = formula)
    S_hat[k,] <- mean_score(scores_k, by.ID = FALSE)[c("SE", "DS", "Brier")]
```

```
  }
  S_hat
}
```

Test your function with e.g. this:

```
colMeans(cvk_do_all(TMINallobs,
                    rep(1:2, nrow(TMINallobs)/2),
                    Value ~ Elevation))

##         SE         DS      Brier
## 24.5094222  4.1990668  0.1562746
```

7. The following function (copy the code from the end of `lab06code.R`) is meant to construct a formula with `max_order` harmonic functions. Unfortunately, it fails to handle the case `max_order = 0`, when no harmonic functions should be included. Modify the code to fix this problem.
Hint: One option is to use `if (...) {...} else {...}`.

```
# See the code for the function:
make_formula

## function (max_order)
## {
##     form <- paste0("Value ~ Longitude + Latitude", " + Elevation + I(DecYear-Year)",
##         paste0(" + I(cos(2*pi*DecYear*", seq_len(max_order),
##             "))", collapse = ""), paste0(" + I(sin(2*pi*DecYear*",
##             seq_len(max_order), "))", collapse = ""))
##     as.formula(form)
## }

# Try the function
make_formula(2)

## Value ~ Longitude + Latitude + Elevation + I(DecYear - Year) +
##     I(cos(2 * pi * DecYear * 1)) + I(cos(2 * pi * DecYear * 2)) +
##     I(sin(2 * pi * DecYear * 1)) + I(sin(2 * pi * DecYear * 2))
## <environment: 0x564d9c4b1a80>

make_formula(0)

## Error in parse(text = x, keep.source = FALSE): <text>:1:81:  unexpected
')'
## 1:  Value ~ Longitude + Latitude + Elevation + I(DecYear-Year) + I(cos(2*pi*DecYear*)
##                                                                                  ^
```

```r
# Solution:
make_formula <- function(max_order) {
  form <-
    paste0("Value ~ Longitude + Latitude",
           " + Elevation + I(DecYear-Year)")
  if (max_order > 0) {
    form <-
      paste0(form,
             paste0(" + I(cos(2*pi*DecYear*", seq_len(max_order), "))",
                    collapse = ""),
             paste0(" + I(sin(2*pi*DecYear*", seq_len(max_order), "))",
                    collapse = ""))
  }
  as.formula(form)
}
```

8. Use `cvk_do_all()` to obtain cross validated prediction scores for models of the type in Quesion 2 of Coursework A.
   Evaluate cross-validated scores for a range of models with `max_order`$\in 0, 1, 2, 3, 4, 5$. Compare the scores for successive models and attempt to detect when adding harmonics no longer has a clear effect. Do the three different types of scores agree? Also compare with the scores for your favourite model when estimating with all of `TMINallobs` and validating with `TMINalltest`.
   Note: If you calculate the basic variance estimate from Lecture 6 (with $\frac{1}{K(K-1)}$, corrected in the pdf after the lecture) it seems either the variance is underestimated, or there is bias in the score estimates; likely due to the dependence between the scores and between cross-validation folds, as dicsussed in the lecture. A more conservative estimate might instead be to use $\frac{1}{K-1}$, which could be motivated as the approximate upper bound if the cross-validated scores were perfectly correlated. However, even this might not reflect the true uncertainty.
   Hint: Part of your code might look something like this:

```r
S0 <- cvk_do_all(TMINallobs, the_splits, make_formula(0))
S1 <- cvk_do_all(TMINallobs, the_splits, make_formula(1))
S1 - S0
```

```r
# Solution:
K <- 10
the_splits <- cvk_define_splits(nrow(TMINallobs), K)

# Lazy solution. A list of data.frame:s would be nicer!
S0 <- cvk_do_all(TMINallobs, the_splits, make_formula(0))
S1 <- cvk_do_all(TMINallobs, the_splits, make_formula(1))
S2 <- cvk_do_all(TMINallobs, the_splits, make_formula(2))
S3 <- cvk_do_all(TMINallobs, the_splits, make_formula(3))
```

```
S4 <- cvk_do_all(TMINallobs, the_splits, make_formula(4))
S5 <- cvk_do_all(TMINallobs, the_splits, make_formula(5))
```

```
S1 - S0
S2 - S1
S3 - S2
S4 - S3
S5 - S4
```

```
rbind(
  colMeans(S0),
  colMeans(S1),
  colMeans(S2),
  colMeans(S3),
  colMeans(S4),
  colMeans(S5))
```

```
##            SE       DS     Brier
## [1,] 23.17880 4.143277 0.1506553
## [2,] 10.19810 3.322230 0.1185123
## [3,] 10.10094 3.312653 0.1173534
## [4,] 10.09081 3.311649 0.1175030
## [5,] 10.08201 3.310775 0.1172932
## [6,] 10.07806 3.310383 0.1172357
```

```
rbind(
  colMeans(S1 - S0),
  colMeans(S2 - S1),
  colMeans(S3 - S2),
  colMeans(S4 - S3),
  colMeans(S5 - S4))
```

```
##                SE           DS         Brier
## [1,] -12.980696173 -0.8210477807 -3.214305e-02
## [2,]  -0.097162257 -0.0095765194 -1.158904e-03
## [3,]  -0.010131582 -0.0010041612  1.496561e-04
## [4,]  -0.008795290 -0.0008735982 -2.097868e-04
## [5,]  -0.003948266 -0.0003928028 -5.754068e-05
```

```
# Large improvement for model order 1 over order 0,
# some improvement for model order 2 over order 1,
# but only small change for order 3 (and Brier increases slightly)

S_final <- mean_score(
  train_and_validate(TMINallobs,
                     TMINalltest,
```

```
                     make_formula(2)),
  by.ID = FALSE)
rbind(colMeans(S2),
      S_final)

##          SE       DS     Brier
## 1 10.100938 3.312653 0.1173534
## 2  9.623071 3.265311 0.1149905

colMeans(S2) - S_final

##          SE         DS       Brier
## 1 0.4778674 0.04734206 0.002362901

# In this case, the cross validated scores were larger than for the test data.
# This is not always the case!
```

9. Optional: Do another analysis of the same models, but with a leave-station-out cross validation approach. Write a `cvk_define_splits_lso(groups)` function to define the data splitting based on the values in `groups`, which should be `TMINallobs$ID` when you call the function.
Hints:
Consider a combination of `as.factor()` and `as.numeric()`.
By using this function, the rest of the code can remain intact!

```
# Solution:
cvk_define_splits_lso <- function(groups) {
  as.numeric(as.factor(groups))
}

the_splits <- cvk_define_splits_lso(TMINallobs$ID)

# Lazy solution. A list of data.frame:s would be nicer!
S0 <- cvk_do_all(TMINallobs, the_splits, make_formula(0))
S1 <- cvk_do_all(TMINallobs, the_splits, make_formula(1))
S2 <- cvk_do_all(TMINallobs, the_splits, make_formula(2))
S3 <- cvk_do_all(TMINallobs, the_splits, make_formula(3))
S4 <- cvk_do_all(TMINallobs, the_splits, make_formula(4))
S5 <- cvk_do_all(TMINallobs, the_splits, make_formula(5))
```

```
S1 - S0
S2 - S1
S3 - S2
S4 - S3
S5 - S4
```

```r
rbind(
  colMeans(S0),
  colMeans(S1),
  colMeans(S2),
  colMeans(S3),
  colMeans(S4),
  colMeans(S5))
```

```
##            SE       DS     Brier
## [1,] 24.20036 4.188915 0.1544530
## [2,] 11.22533 3.429981 0.1231935
## [3,] 11.12903 3.421593 0.1227020
## [4,] 11.11917 3.420700 0.1228258
## [5,] 11.11030 3.419902 0.1226213
## [6,] 11.10597 3.419517 0.1225497
```

```r
rbind(
  colMeans(S1 - S0),
  colMeans(S2 - S1),
  colMeans(S3 - S2),
  colMeans(S4 - S3),
  colMeans(S5 - S4))
```

```
##                 SE            DS         Brier
## [1,] -12.975031540 -0.7589343072 -3.125952e-02
## [2,]  -0.096306535 -0.0083879929 -4.914830e-04
## [3,]  -0.009858242 -0.0008927802  1.237406e-04
## [4,]  -0.008869335 -0.0007979267 -2.044261e-04
## [5,]  -0.004326213 -0.0003851330 -7.159481e-05
```

```r
# Very similar score differences to before, but the scores
# themselves are larger, which was to be expected; each station has
# its own characteristics that we don't capture in these models.

S_final <- mean_score(
  train_and_validate(TMINallobs,
                     TMINalltest,
                     make_formula(2)),
  by.ID = FALSE)
rbind(colMeans(S2),
      S_final)
```

```
##          SE       DS     Brier
## 1 11.129025 3.421593 0.1227020
## 2  9.623071 3.265311 0.1149905
```

```r
colMeans(S2) - S_final
```

```
##         SE        DS       Brier
## 1 1.505954 0.1562817 0.007711578

# Again, the cross validated scores were larger than for the test data.
# However, the appropriate comparison here would have been to have had
# a seventh, held-out test station!
```