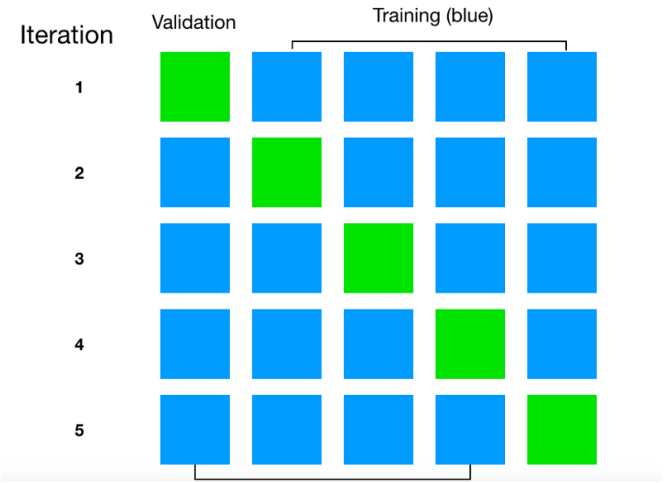
Swiss Paraplegic Research

Workshop:

Machine Learning and Prediction Modelling

Cross-Validation Function in R



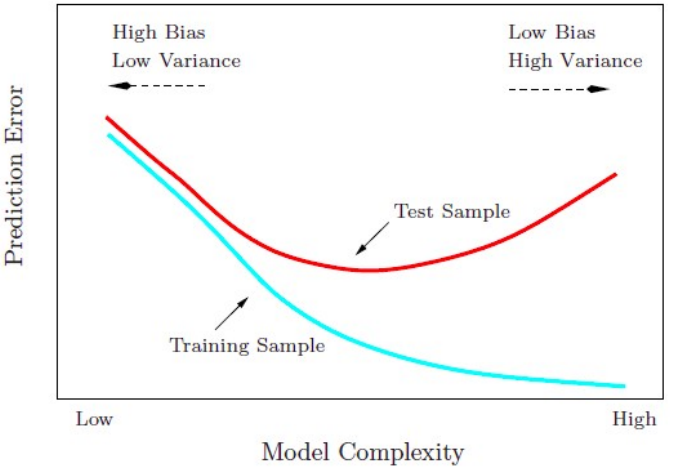
Yannick Rothacher

SPF, HS2025

Swiss Paraplegic Research

What is cross-validation?

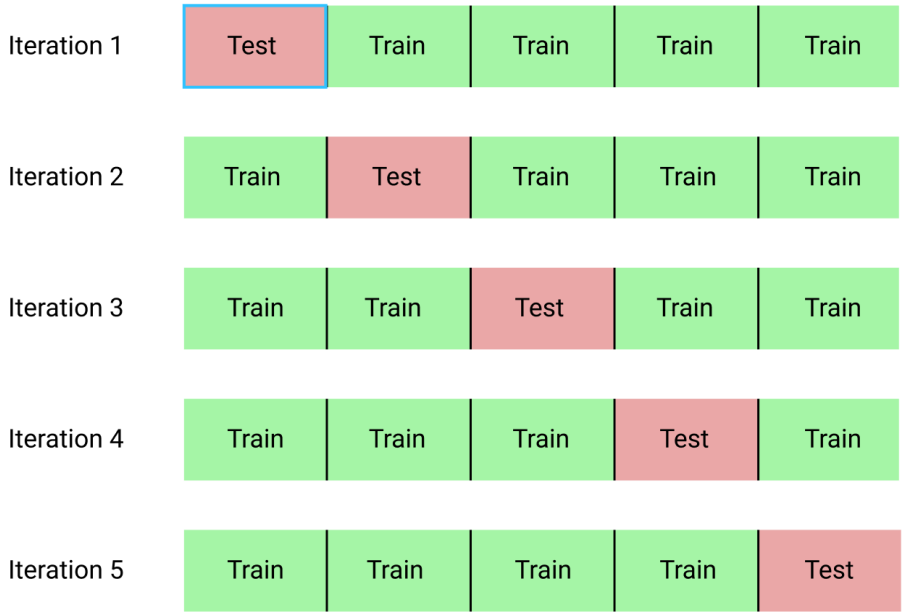
- Recap: To test how well a classifier is performing we wish to test its predictions on **new “test” data**
- We want a method to estimate the test-error as efficient as possible
  - Cross-validation is a popular method for estimating the test error
- Cross-validation works by **splitting** the data into multiple test- and training-sets and calculates the test-error for each test-set



Swiss Paraplegic Research

What is cross-validation?

- K-fold cross-validation (e.g. 5-fold below)



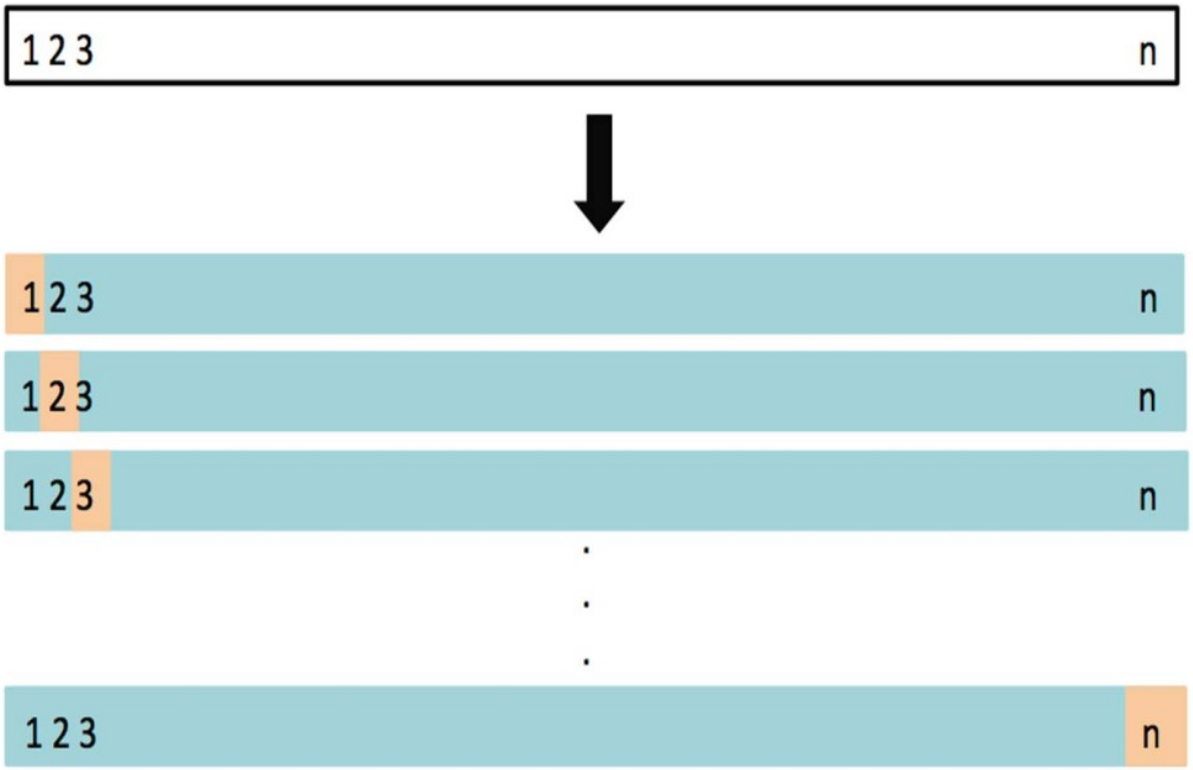
Also referred to as “validation sets”

- For each iteration use one k^{th} of the data as a test-set (fitting the model to the remaining data)
  - In the end **each observation** has been part of a test-set once
- Calculate the final test error based on all iterations

Swiss Paraplegic Research

Leave-One-Out Cross-validation

- Same as k-fold cross-validation but letting each observation be an independent test-set
- Each iteration: Use blue data as training-set to predict the orange observation.

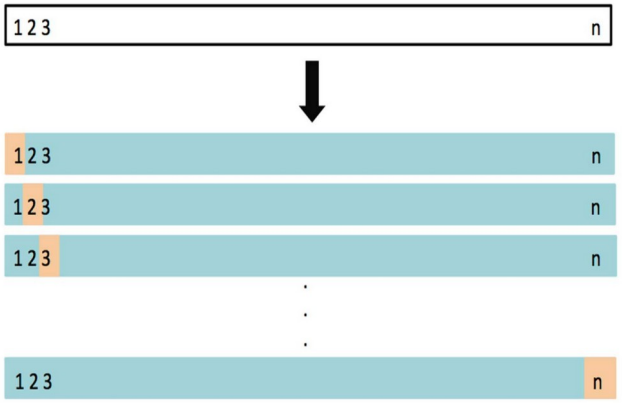


Swiss Paraplegic Research

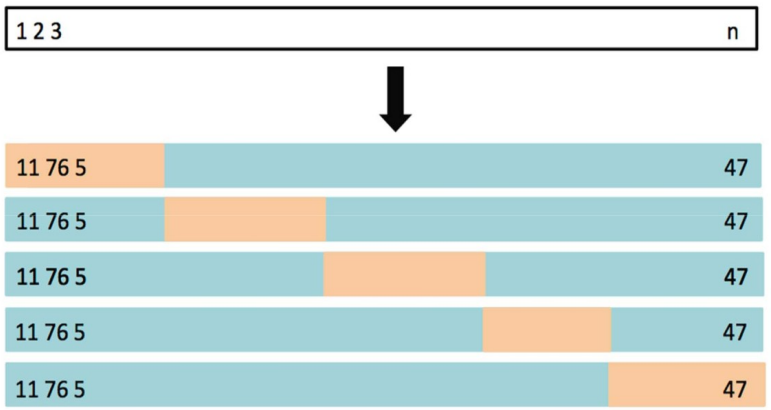
How to choose fold-size (k)?

- The fold size (k) can be chosen between **2 and n** (k=n results in leave-one-out CV)
- What value is best for k?
  - Leave-one-out CV has lowest possible bias but can have high variance. Can be slow.
  - K-fold CV is usually faster but is random.
- Standard is often to use **k=10 fold CV**

k=n



k=5



Swiss Paraplegic Research

Goal of this lecture

- Write your own **k-fold cross-validation** function in R for k-nearest neighbor classification

- … First we need to look at **functions** in R

Swiss Paraplegic Research

function() in R

- A function is a **set of commands** normally applied to some input and producing some output
- R has many built-in functions
  - e.g. mean(), sd(), max(), ...
- Creating a function in R follows the syntax:

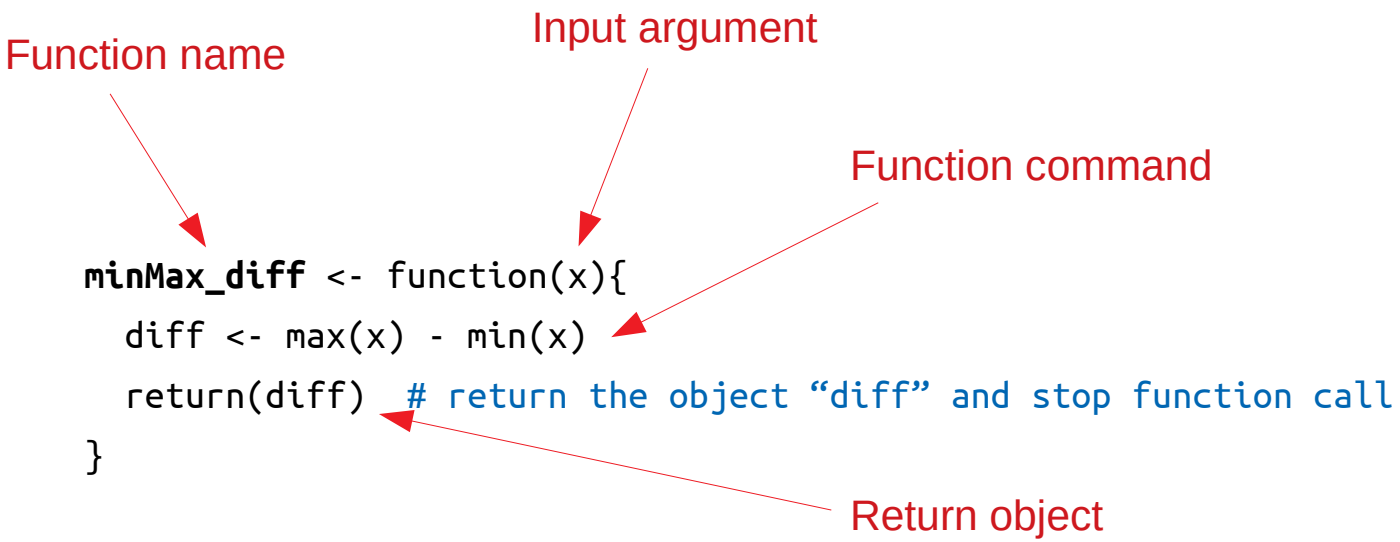
```
function_name <- function(arg_1, arg_2, ...) {
   Function body
   return(return_object)
}
```

  - The **function_name** is the name under which the function is stored
  - The arguments (**arg_1**, …) are input-values which are later used for the commands
  - The **Function body** are the commands that the function executes
  - Usually a function contains a **return object**, which is the output of the function

Swiss Paraplegic Research

function() in R

- Simple example of a function:
  - Calculate the **difference** between the **largest** and the **smallest** element of a numeric vector




```
minMax_diff <- function(x){
  diff <- max(x) - min(x)
  return(diff)  # return the object "diff" and stop function call
}

# Apply the function:
vec <- 1:20
minMax_diff(x = vec)
[1] 19
```

Swiss Paraplegic Research

function() in R

- A function can take **more than one** input argument
  - Same function calculating the difference between maximum element of **vector A** and minimum element of **vector B**:

```
minMax_diff <- function(X, Y){
  diff <- max(X) - min(Y)
  return(diff)  # return the object “diff” and stop function call
}

# Apply the function:
vecA <- 1:20
vecB <- c(3,22,60,20)
minMax_diff(X=vecA, Y=vecB)
[1] 17
```

- If we do not specify the input, R will return an error:

```
minMax_diff(X=vecA)  # input argument Y not specified
Error in minMax_diff(X = vecA) : argument "Y" is missing, with no default
```

Swiss Paraplegic Research

function() in R

- A function can take **more than one** input argument
  - Same function calculating the difference between maximum element of **vector A** and minimum element of **vector B**:

```
minMax_diff <- function(X, Y){
  diff <- max(X) - min(Y)
  return(diff)  # return the object "diff" and stop function call
}

# Apply the function:
vecA <- 1:20
vecB <- c(3,22,60,20)
minMax_diff(X=vecA, Y=vecB)
[1] 17
```

If I do not name the input arguments, the order of them will be used for assignment.

e.g. **`minMax_diff(vecB, vecA)`**

→ **`vecB`** will be **`X`**

- If we do not specify the input, R will return an error:

```
minMax_diff(X=vecA)  # input argument Y not specified
Error in minMax_diff(X = vecA) : argument "Y" is missing, with no default
```

Swiss Paraplegic Research

function() in R

- We can define default values for the input arguments (to use in case not specified)

```
minMax_diff <- function(X, Y=15:50){  # default value for Y defined
  diff <- max(X) - min(Y)
  return(diff)
}
# Apply the function:
vecA <- 1:20
minMax_diff(X=vecA)  # input argument Y not specified
[1] 5
```

- The return object can be **any R-object** (e.g. number like above, vector, character, data.frame, …)
- What if we want to produce **multiple objects** as output (can only have **one** return call in function)?
  - e.g. calculate the difference as above but also include the input-vectors in output
  - In this case we need to use a **list** as a return-object

Swiss Paraplegic Research

Lists in R

- A **list** is like a vector, but each element can be an independent **R-object**
  - For example a list can store a **numeric vector** as a first element, a **data frame** as a second element and a **string** as a third element:

```
# Create a list:
vec <- 1:5
dat <- iris[1:3,1:3]
string <- 'A little sentence'
L <- list(A=vec, B=dat, C=string)
L
$A
[1] 1 2 3 4 5

$B
  Sepal.Length Sepal.Width Petal.Length
1          5.1         3.5          1.4
2          4.9         3.0          1.4
3          4.7         3.2          1.3

$C
[1] "A little sentence"
```

```
# How to index in a list: use [[ ]]
L[[2]]  # access second element
  Sepal.Length Sepal.Width Petal.Length
1          5.1         3.5          1.4
2          4.9         3.0          1.4
3          4.7         3.2          1.3

L$C  # possible if the elements are named
[1] "A little sentence"

L[[2]][1,]    # access the first row
              # of second element
  Sepal.Length Sepal.Width Petal.Length
1          5.1         3.5          1.4
```

Swiss Paraplegic Research

function() in R

- By using a list as the return object of a function, we can include **multiple objects** in our output:

```
minMax_diff <- function(X, Y){
  diff <- max(X) - min(Y)
  L <- list(diff=diff, InputV1=X, InputV2=Y)
  return(L)  # return the list
}

Outp <- minMax_diff(X=1:20, Y=c(3,22,60,20))
Outp
$diff
[1] 17

$InputV1
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20

$InputV2
[1]  3 22 60 20
```

Swiss Paraplegic Research

function() in R

- Variables assigned inside a function are not stored in the global environment:

```
minMax_diff <- function(X, Y){
  diff <- max(X) - min(Y)
  L <- list(diff=diff, InputV1=X, InputV2=Y)
  return(L)  # return the list
}

Outp <- minMax_diff(X=1:20, Y=c(3,22,60,20))
L
Error: object 'L' not found
```

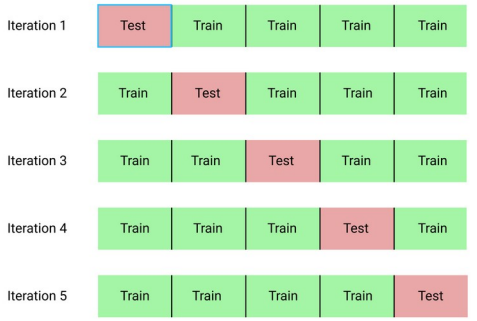
Swiss Paraplegic Research

Custom CV function

- To write our own **k-fold cross-validation function** (for KNN) we want to create something like this:

```
KNN_crossVal <- function(X, y, k_fold=10, KNN_k=1) {

   ...Perform cross-validation...

   return(List including error-rate and confusion-matrix)
}
```

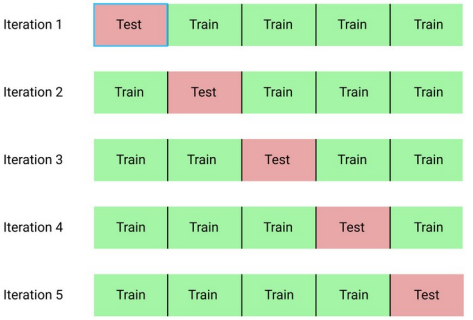
- We will split the cross-validation into two steps
  - 1) Divide a data frame into k test- and training-sets
  - 2) Fit the KNN classifier to the k training-sets and make predictions for k test sets



Swiss Paraplegic Research

Exercise 1

- **Step 1**: Write a function, which ...
  - takes a Vector and a k-parameter as input
  - Takes the indices of the vector, mixes them up and divides them evenly into k parts
  - saves these k parts for later use (stored in a list)



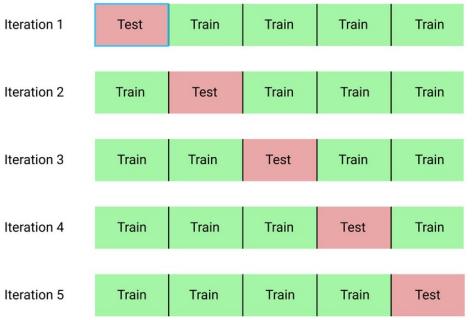

```
vec <- c(5, 21, 11, 4, 7, 8, 11, 2)
kDivide.Vec(vec, k = 3)
[[1]]
[1] 3 1 5

[[2]]
[1] 7 6 4

[[3]]
[1] 2 8
```

```
vec
[1] 5 21 11 4 7 8 11 2
1:length(vec) # indices
[1] 1 2 3 4 5 6 7 8
sample(1:length(vec)) # mixed up indices
[1] 3 1 5 7 6 4 2 8
```

Swiss Paraplegic Research

Exercise 1

- **Step 1**: Write a function, which ...
  - takes a Vector and a k-parameter as input
  - Takes the indices of the vector, mixes them up and divides them evenly into k parts
  - saves these k parts for later use (stored in a list)




```
vec <- c(5, 21, 11, 4, 7, 8, 11, 2)
kDivide.Vec(vec, k = 3)
[[1]]
[1] 3 1 5

[[2]]
[1] 7 6 4

[[3]]
[1] 2 8
```

```
vec
[1] 5 21 11 4 7 8 11 2
1:length(vec) # indices
[1] 1 2 3 4 5 6 7 8
sample(1:length(vec)) # mixed up indices
[1] 3 1 5 7 6 4 2 8
```

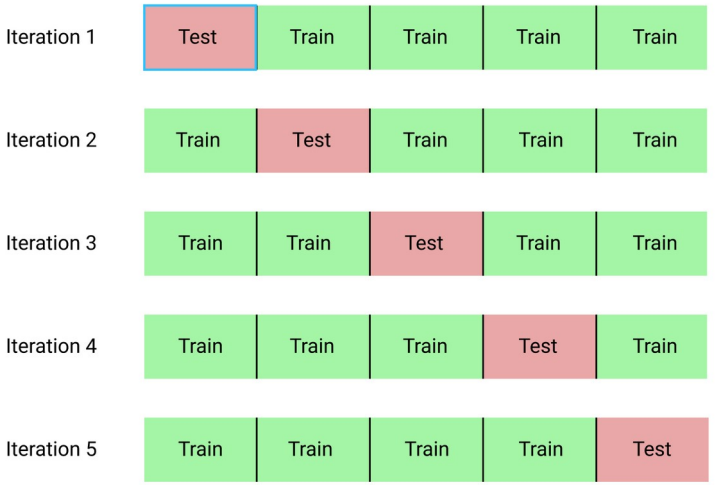
→ In above example the length of the vector is not divisible by k. To split a vector into k parts that are as equal in size as possible, you can use the following function:

```
split(vec, f=1:k)
```

Swiss Paraplegic Research

Solution 1

```
kDivide.Vec <- function(vec, k){
  n <- length(vec)
  ind_s <- sample(1:n)
  L <- split(ind_s, f=1:k)  # Can use suppressWarnings(split(ind_s, f = 1:k))
  return(L)
}

# Apply function:
set.seed(48484)
vec <- rnorm(n = 8)
kDivide.Vec(vec, k = 3)
$`1`
[1] 1 6 8

$`2`
[1] 3 2 4

$`3`
[1] 7 5
```

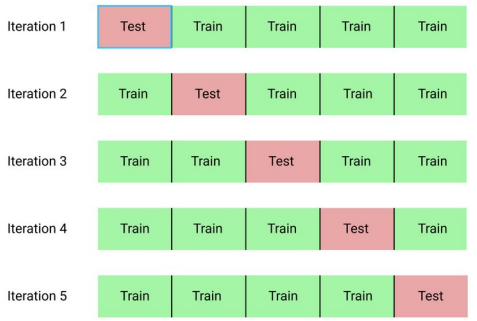


Swiss Paraplegic Research

```
KNN_crossVal <- function(X, y, k_fold=10, KNN_k=1) {

   ...Perform cross-validation...

   return(List including error-rate and confusion-matrix)
}
```

Exercise 2

- **Step 2**: Write a function, which …
  - takes a data frame containing the predictor variables (X), a vector containing the target variable categories of the observations (y), the "k-fold" parameter k and a "KNN" parameter k (how many neighbors are used) as input
  - Divides the row-indices of the data frame into k subsets
  - Applies the KNN classifier to each test- / train-set (and collect predictions to build confusion matrix)
  - Creates a list as output with the elements:
    1) The final confusion matrix
    2) The final test-error
- **Step 3**:
  - Apply the function to the iris data set to estimate the test-error of the KNN classifier with 10-fold cross validation and a KNN_k = 5



Swiss Paraplegic Research

Solution 2

- Check the **CVFunction** R script in Exercise folder
- Try to understand and recreate each step!