Accuracy Statistics in R

In this assignment we will focus on creating a confusion matrix in R. Additionally, we will perform a significance test, and calculate confidence intervals as well as the kappa coefficient.

Accuracy Matrix

A confusion matrix, also known as error or accuracy matrix, is a specific type of table showing the performance of an algorithm. The name *confusion matrix* comes from the fact that you can quickly see where the algorithm confuses two classes, which would indicate a misclassification.

Several metrics can be derived from the table:

- **User's accuracies**: Depict how often the class in the classification map will actually be present on the ground. The user's accuracy is complement of the commission error (user's accuracy = 100% – commission error);
- **Producer's accuracies**: Depict how often real features in the study area are correctly shown on the classification map. The producer's accuracy is complement of the o*mission error* (producer's accuracy = 100% – omission error);
- **Overall accuracy**: Characterize the proportion of all correctly classified validation points in percent.

Implementation in R

As usual, make sure to have the `raster` package available. First, we import our classification image "RF_classification.tif", the training shapefile "training_data.shp" and the validation shapefile "RF_validation.shp". In this example we use the output of the RF classification. However, the workflow is applicable to every classifier!

```
img.classified <- raster("RF_classification.tif")
shp.train <- shapefile("training_data.shp")
shp.valid <- shapefile("RF_validation.shp")
```

Our goal is first to generate two factor vectors, which we then compare in our confusion matrix:

1. **reference** : class labels you assigned manually in the previous section
2. **predicted** : class labels that resulted in the automatic RF (or SVM) classification

For the reference vector, we need to address the *validclass* column of our shapefile (we created this column in the previous assignment). We need to transform the vectors into factors using `as.factor()` (it becomes clear why in a second):

```
> reference <- as.factor(shp.valid$validclass)
> reference
  [1] 2    4    4    2    4    2    3    2    <NA> 4    2    <NA> 2    1    5    2    3    <NA> 4    3    3    3
 [23] 3    1    5    2    3    1    1    3    2    5    3    4    5    5    2    2    3    5    4    <NA> 2    3
 [45] <NA> <NA> <NA> <NA> 3    5    4    <NA> 3    1    2    5    5    3    2    3    3    2    2    2    5    2
 [67] 5    <NA> 1    4    1    2    3    3    3    <NA> 2    1    5    5    3    <NA> 3    3    3    2    3    <NA>
 [89] 1    <NA> 5    5    1    2    1    5    5    <NA> 4    3    2    1    3    3    1    3    3    3    2    2
[111] 2    5    3    <NA> 5    2    2    4    2    5    3    1    3    2    5    5    5    5    <NA> 5    2    3
[133] 1    3    1    3    2    1    3    2    4    2    2    3    1    2    2    2    1    1    <NA> 5    5    3
[155] 1    3    1    5    5    1    2    5    2    2    2    2    5    <NA> 2    1    2    5    1    1    3    2
[177] 3    3    2    4    1    3    3    1    5    3    5    3    3    2    5    5    1    4    5    2    2    3
[199] 3    1    5    2    <NA> 3    2    3    3    5    3    1    4    1    4    2    2    3    5    5    3    2
[221] 3    1    5    5    3    1    5    3    2    5    5    2    2    4    3    3    <NA> 4    2    3    4    5
[243] <NA> 2    3    2    4    3    1    3
Levels: 1 2 3 4 5
```

You may notice some NA entries in your reference values. These arise, for example, if you have skipped some validation points during the labeling in QGIS, because you could not make a clear classification based on the available image data. We simply ignore them later on in the statistics – no problem!

For the predicted vector, we need to extract the classification values at the validation coordinates. Here again, we want a factor object via `as.factor()`:

```
> predicted <- as.factor(extract(img.classified, shp.valid))
> predicted
  [1] 2 4 4 2 4 2 3 2 1 1 2 4 2 1 5 2 3 4 4 3 4 1 3 1 5 2 3 3 1 1 2 5 3 4 5 5 2 3 1 5 4 3 2 3 2 5 1 3 3 5 4 4 4 1 2 5
 [57] 5 4 3 4 1 4 2 2 5 2 5 4 1 4 1 2 1 1 4 4 2 3 5 5 4 2 3 4 3 3 4 4 3 5 5 5 1 3 1 5 5 4 4 3 2 4 1 1 4 3 4 3 4 2 2 5
[113] 1 3 5 4 2 4 3 5 2 3 1 2 5 5 5 5 4 5 2 1 3 1 4 3 2 1 1 4 4 2 2 3 1 2 2 2 1 3 5 5 5 3 1 4 1 5 5 3 1 5 2 2 2 2 5 1
[169] 3 3 2 5 3 1 3 2 2 1 4 4 3 1 3 1 5 3 5 4 3 2 5 5 3 4 5 2 2 1 1 3 5 3 1 1 2 1 3 5 3 1 4 1 4 3 2 1 5 5 1 2 4 1 5 5
[225] 4 1 5 3 3 5 5 4 2 4 3 3 1 4 2 3 4 5 2 2 1 2 4 1 1 4
Levels: 1 2 3 4 5
```

Once we prepared both factor vectors, we can utilize the table function in an elegant way to build a contingency table of the counts at each combination of factor levels. We can additionally name the vectors so that they are also displayed in the table accordingly:

```
accmat <- table("pred" = predicted, "ref" = reference)
accmat
    ref
pred  1  2  3  4  5
   1 20  1 23  1  0
   2  0 45  2  0  0
   3 12  9 26  0  0
   4  3  6 14 19  0
   5  0  0  0  0 47
```

The numbers thus reflect the number of validation pixels. All pixels that have a `NA` value in either reference or predicted were ignored here.
Excellent! This output already visualizes, if and where there are misclassifications in our map: all pixels located on the diagonal are correctly classified, all pixels off the diagonal are not.

We can now calculate the user's accuracies UA, producer's accuracies PA, and the overall accuracy OA:

```
> #Generate user's accuracy
> UA <- diag(accmat) / rowSums(accmat) * 100
> UA
        1         2         3         4         5
 44.44444  95.74468  55.31915  45.23810 100.00000
>
> #Generate producer's accuracy
> PA <- diag(accmat) / colSums(accmat) * 100
> PA
        1         2         3         4         5
 57.14286  73.77049  40.00000  95.00000 100.00000
>
> #Generate overall accuracy
> OA <- sum(diag(accmat)) / sum(accmat) * 100
> OA
[1] 68.85965
```

Actually we already extracted all information needed for a confusion matrix, so let us form a nicely formatted matrix in R:

```
> #Generate nicely looking matrix
> accmat.ext <- addmargins(accmat)
> accmat.ext <- rbind(accmat.ext, "Users" = c(PA, NA))
> accmat.ext <- cbind(accmat.ext, "Producers" = c(UA, NA, OA))
> colnames(accmat.ext) <- c(levels(as.factor(shp.train$class)), "Sum", "PA")
> rownames(accmat.ext) <- c(levels(as.factor(shp.train$class)), "Sum", "UA")
> accmat.ext <- round(accmat.ext, digits = 1)
> dimnames(accmat.ext) <- list("Prediction" = colnames(accmat.ext),
+                              "Reference" = rownames(accmat.ext))
> class(accmat.ext) <- "table"
> accmat.ext
           Reference
Prediction  field forest grassland urban water   Sum    UA
  field      20.0    1.0      23.0   1.0   0.0  45.0  44.4
  forest      0.0   45.0       2.0   0.0   0.0  47.0  95.7
  grassland  12.0    9.0      26.0   0.0   0.0  47.0  55.3
  urban       3.0    6.0      14.0  19.0   0.0  42.0  45.2
  water       0.0    0.0       0.0   0.0  47.0  47.0 100.0
  Sum        35.0   61.0      65.0  20.0  47.0 228.0
  PA         57.1   73.8      40.0  95.0 100.0        68.9
```

Significance Test

Furthermore, we can check if the result is purely coincidental, i.e., whether a random classification of the classes could have led to an identical result. We can use a binomial test for this. We only need two values for this test:
x = total number of correctly classified validation points, and
n = the total number of validation points in our confusion matrix:

```
sign <- binom.test(x = sum(diag(accmat)),
                   n = sum(accmat),
                   alternative = c("two.sided"),
                   conf.level = 0.95
)

pvalue <- sign$p.value
pvalue

1.227817e-08

CI95 <- sign$conf.int[1:2]
CI95

0.6241226 0.7480874
```

The p-value is much lower than 0.05, so the classification result is highly significant. If the classification were repeated under the same conditions, it can be assumed that the OA is 95% in the range of 62.2% to 74.8%.

Kappa-Coefficient

The Kappa Coefficient can be used to evaluate the accuracy of a classification. It evaluates how well the classification performs compared to map, in which all values are just randomly assigned.
The Kappa coefficient can range from -1 to 1.
A value of 0 indicates that the classification is as good as random values.

A value below 0 indicates the classification is significantly worse than random.
A value greater than 0 indicates that the classification is significantly better than random.

When you have the accuracy matrix as a table $m_{i,j}$ with c different classes, then Kappa is:

$$\kappa = \frac{N_o - N_e}{N - N_e}, \text{with}$$

$$N = \sum_{i,j=1}^{c} m_{i,j}$$

$$N_o = \sum_{i=1}^{c} m_{i,i}$$

$$N_e = \frac{1}{N} \cdot \sum_{l=1}^{c} \left(\sum_{j=1}^{c} m_{l,j} \cdot \sum_{i=1}^{c} m_{i,l} \right)$$

That looks pretty complicated. Using R, we can write our own function, which calculates κ for us! The calculation also looks much friendlier:

```
kappa <- function(m) {
  N <- sum(m)
  No <- sum(diag(m))
  Ne <- 1 / N * sum(colSums(m) * rowSums(m))
  return( (No - Ne) / (N - Ne) )
}
```

Now, we simply use `accmat` as an argument for our kappa-function:

```
kappa(accmat)
0.6093441
```

Thus, our classification example can be considered as significantly better than random. However, the usage of kappa has been questioned in many articles (e.g. PONTIUS and MILLONES 2011). It is advisable to read more about kappa, in order to correctly estimate its shortcomings. According to PONTIUS and MILLONES "Kappa indices are useless, misleading and/or flawed for the practical applications in remote sensing that we have seen".

Questions / prove your knowledge:

- What is the difference between user's and producer's accuracy?
- What is the difference between your reference and predicted vector?
- What does a significance test do?