Validation of soil maps

Bas Kempen

Version 2, 24 May 2018

This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License.

This tutorial shows how to assess the accuracy of a soil map through validation. It shows how to compute various accuracy measures and illustrates the functionality of the `caret` package for this purpose. It uses the results of a random forest model fitted soil organic carbon data from Macedonia. Though a random forest model is used here, maps generated with other statistical methods can be validated in a similar way. Note this tutorial only considers cross-validation and data-splitting methods for validation. Map validation using indepented probability sampling is not considered here. The tutorial draws on theory from Brus, Kempen, and Heuvelink (2011) and Chapter 7 of FAO (2018).

0 Setting up an R session

Before starting scripting, it is recommended to empty the computer memory, clean up the workspace and load the required libraries.

```
# empty memory and workspace
gc()
rm(list=ls())

# load libraries
require(randomForest)
require(caret)
```

Also, we need to set a random seed to make our results reproducible. Some of the validation approaches illustrated in this tutorial use random data splitting. This would mean that each time you run the code, the results will be (slightly) different. To make results reproducible we need to set a so-called random seed with the **set.seed()** function. The number used for this can be any number. In this way we ensure that we always get the same result when we run this code.

```r
# set random seed
set.seed(2011102)
```

1 Random forest out-of-bag accuracy assessment

1.1 Read the input data

This tutorial uses a soil organic carbon (SOC) dataset from Macedonia. SOC content was measured for more than 3,000 sampling sites across the country. The measurements were related to over 50 covariate layers, i.e. GIS raster layers of biophysical land surface propertyies, in a random forest model to derive predictive relationships.

We start by reading the input data: the **fitted random forest model** and the **regression matrix** that contains locations of the sampling sites and the SOC measurements with associated covariate values. Both are saved in an `R` output file that comes with this tutorial. We now read this file.

```r
# read the random forest model
load("randomForestModel.rda")
```

Before we continue, we briefly explore the input data with the **`str()`** function, that shows the data structure, and the **`summary()`** function, that gives a statistical summary of the data.

```r
# explore input data
summary(rm)

str(rf, max.level = 1)
```

1.2 Computing the out-of-bag prediction error

Map quality measures for quantitative soil maps of continuous soil properties (such as SOC content) are computed from the prediction error that is defined as the difference between the predicted value at a location and the observed or measured value at that location.

The prediction error for our random forest model can thus be computed from the predicted SOC content at the sampling sites and the measured SOC contents. Both properties are stored in the random forest model output object. The predicted values can be found in attribute `predicted` and the measured values in the attribute `y`.

Note that the predicted values are computed from the **out-of-bag** (OOB) samples. Each sample is OOB for approximately 37% of the trees fitted. Here we fitted a model with 500, so on average each sample is 184 times OOB. Each time a sample is OOB, the model is used make a prediction for that sample. This means that each sample receives on average 184 predictions. For each sample the overall OOB prediction is then computed by averaging the individual OOB predictions.

We proceed by computing the OOB prediction error.

```r
# compute prediction error
pe <- rf$predicted - rf$y

# statistical summary
summary(pe)
```

1.3 Computing map accuracy measures

There are many measures and statistics used to quantify the accuracy of a model and map. Here we consider three:

1. Mean Error
2. Mean Squared Error
3. Amount of Variance Explained

The **mean error** (ME) is a measure of bias which quantifies if there is a systematic over- or underestimation of the modelled variable. The **mean squared error** (MSE) is a measure of accuracy which quantifies the prediction error we make on average. From the MSE the **root mean squared error** (RMSE) is often computed by taking the square root. The RSME is on the same measurement scale as the target variable. The **amount of variance explained** (AVE) quantifies the fraction of variation (or variance) in the data that is explained by the model. It measures the improvement of the model prediction compared to using the mean of the data set as predictor (using the mean of the dataset as predictor is the most simple model one can use).

Note, if the squared error distribution is strongly skewed, for instance when several very large errors are present, then this skewness can severely inflate the (R)MSE. In such case, the (root) median squared error (RMedSE) is a more robust statistic for the 'average' error.

AVE versus R^2. Another accuracy measure that is often used for validation is the R^2, the coefficient of determination. The R^2 is only equal the AVE if the R^2 is computed as the fraction of explained variance, i.e. as 1 minus the ratio between the residual sum of squares and the total sum of squares. Sometimes, the R^2 is computed as the squared Pearon's correlation coeffcient between predicted and observed values. However, the R^2 computed this way is not directly a measure of how good the modelled values are, but rather a measure of how good a predictor of the target variable (here SOC) might be constructed from the modelled values. In reality the R^2 as squared correlation coefficient and R^2 as fraction of explained variation are often close. However, there can be cases when the difference is large, for instance when there is strong prediction bias. Thus, when an R^2 is reported, then it is important to explain how this value was computed (on basis of sum of squares or on basis of the correlation coefficient). To avoid any confusion, we prefer to use 'AVE' here which is based on sum of square statistics.

We will now calculate these accuracy measures 'manually'.

```
# ME
(me <- round(mean(pe),3))
```

```
## [1] 0.033
```

```
# MSE
(mse <- round(mean(pe**2),2))
```

```
## [1] 2.58
```

```
# RMSE
(rmse <- round(sqrt(mean(pe**2)),2))
```

```
## [1] 1.61
```

```
# RMedSE
(rmedse <- round(sqrt(median(pe**2)),2))
```

```
## [1] 0.44
```

```
# AVE
(ave <- round(1-(sum(pe**2)/(sum((rf$y-mean(rf$y))**2))),3))
```

```
## [1] 0.389
```

```
# rsq computed from correldation coefficient
(rsqCor <- round((cor(rf$predicted, rf$y)**2),3))
```

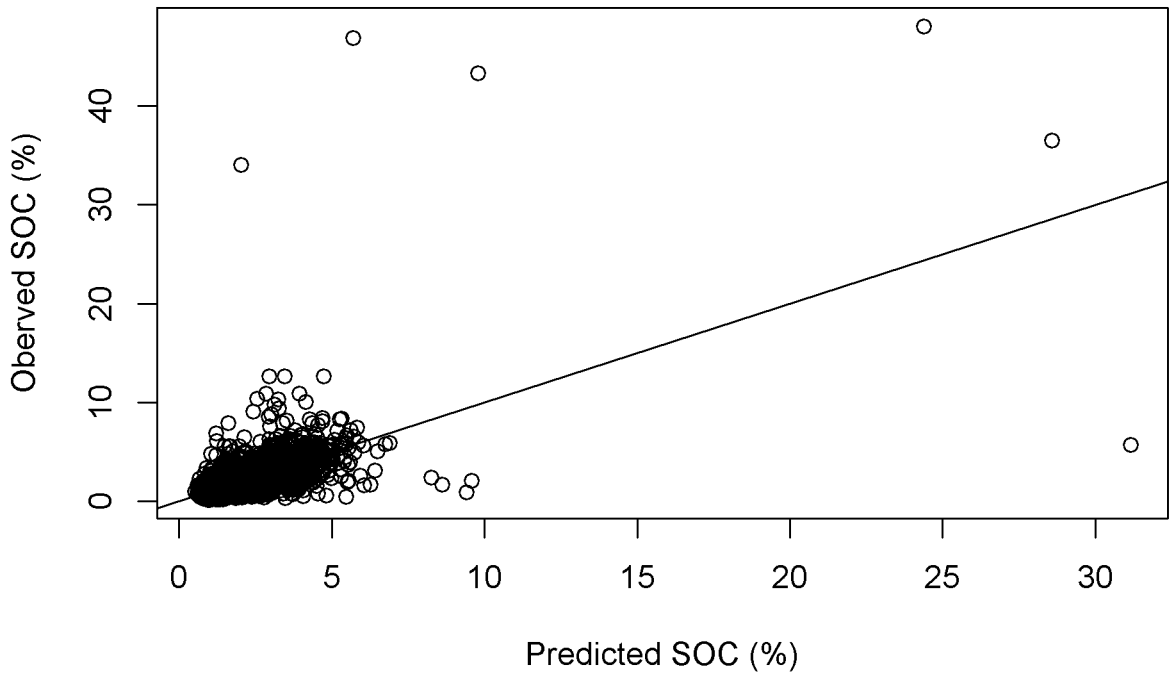
```
## [1] 0.39
```

The validation statistics can be easily compiled in a table and exported.

```
# create empty matrix
valstat <- matrix(nrow=6, ncol=1)

# add statistics
valstat <- rbind(me,mse,rmse,rmedse,ave,rsqCor)

# give row and column names
row.names(valstat) <- c('ME', 'MSE', 'RMSE', 'RMedSE', 'AVE', 'RSQ-COR')
colnames(valstat) <- c('value')

# export
write.csv(valstat, "rf_validation.csv")
```

It is good practice to make a scatterplot of the observed versus the predicted value and add the 1:1 line to this plot. Such plot can be informative.

```
# scatterplot
plot(rf$predicted, rf$y, xlab="Predicted SOC (%)", ylab="Oberved SOC (%)")
abline(0,1)
```



The plot shows that for some samples the difference between observed and predicted SOC content is very large, hence we have very large prediction errors (and thus a large difference between the RMSE and RMedSE). It is clear that the SOC content of sampling sites that have organic soils is not well modelled with our model.

It is worthwhile to point out that the random forest model output already contains the OOB MSE and the R^2 values in the `mse` and `rsq` attributes. These can be easily accessed.

```
# MSE
round(rf$mse[rf$ntree],2)
```

```
## [1] 2.58
```

```
# r-squared
round(rf$rsq[rf$ntree],3)
```

```
## [1] 0.389
```

Note that the model stores the MSE and R^2 cumulatively for each tree in the forest. Thus to obtain these statistics for the random forest model as a whole we must use the values calculated from all trees. This means here we have to access the 500^{th} value, hence the use of `rf$ntree` which returns the number of trees in the forest. Here 500.

Here we see that the R^2 value reported by the `randomForest` function is equal to the AVE value and is thus computed from the sum of squares, and not as the squared correlation coefficient.

We end this section with a small warning: always use the OOB prediction to validate. Do **not** use the fitted random forest model to predict at the sampling sites and then compare predictons with the observed values. This approach will grossly overestimate the accuracy measures since the calibrartion data will not be independent from the validation data. This is illustrated by the example below.

```
# predict at sampling sites
pred.ss <- predict(rf, newdata = rm)

# compute prediction error
pe2 <- pred.ss - rm$SOC

# compute RMSE
round(sqrt(mean(pe2**2)),2)
```

```
## [1] 0.79
```

```
# compute AVE
round(1-(sum(pe2**2)/(sum((rm$SOC-mean(rm$SOC))**2))),3)
```

```
## [1] 0.852
```

Compare these values with the OOB statistics. What do you observe?

1.4 Computing map accuracy measures using the caret package

The caret package is a versatile package for creating predictive models. It comes with two validation functions: `RMSE` and `R2`. We can apply these functions to the fitted random forest model.

```
# caret validation functions
round(RMSE(rf$predicted, rf$y),2)
```

```
## [1] 1.61
```

```
round(R2(rf$predicted, rf$y),3)
```

```
## [1] 0.39
```

Let's compare the R^2 value obtained with the `R2()` function with the squared correlation coefficient without rounding the values.

```
# compare to squared correlation coefficient
R2(rf$predicted, rf$y)
```

```
## [1] 0.3895854
```

```
cor(rf$predicted, rf$y)**2
```

```
## [1] 0.3895854
```

Here we see that the `R2()` function reports the squared correlation coefficient, which might not be equal to the fraction of the variance of the dataset that is explained by the model. This can be remedied by adding the argument `formula = "traditional"` to the function.

```
# caret R2
R2(rf$predicted, rf$y, formula = "traditional")
```

```
## [1] 0.3893114
```

```
# AVE
1-(sum(pe**2)/(sum((rf$y-mean(rf$y))**2)))
```

```
## [1] 0.3893114
```

Now the R^2 computed by `caret` is similar as the manually computed AVE.

2 Validation using data splitting

A frequently used approach for validation is based on **data splitting**. In data splitting, two subsets are created from the sampling dataset of which one is used to calibrate (or train) the model while the other is used for validation. Typically, a 70:30 split is applied in which 70% of the sampling sites is used for calibration and 30% for validation. Though we do not advocate the use of data splitting for validation we will show here how this can be done in `R` making use of the functionality of the `caret` package.

2.1 Data splitting

The `caret` package offers various methods for data splitting, (see its documentation). Here we will apply a simple one making use of the **`createDataPartition()`** function. The argument `p` is the percentage fo data that goes to the training set.

```
# partition the dataset
trainIndex <- createDataPartition(rm$SOC, p = 0.7, list = FALSE, times = 1)

# inspect
head(trainIndex)
```

```
##      Resample1
## [1,]         2
## [2,]         5
## [3,]         6
## [4,]         7
```

```
## [5,]          9
## [6,]         10
```

This function returns a vector with the row positions of the calibration dataset. We can now subset the dataset on basis of the data partitioning.

```
# subset the dataset
socTrain <- rm[ trainIndex,]
socTest  <- rm[-trainIndex,]
```

2.2 Fit random forest model

Now we fit a new random forest model using the training data set that contains 70% of the data. The **randomForest()** function of the **randomForest** package requires a minimum of two inputs: one `vector` containing the sampled values and one `data.frame` with the covariate data.

```
# check the column names and numbers in the training object (covariates are stored in columns 5 to 63)
names(socTrain)

# copy soil property values to a new vector
dTrain <- socTrain$SOC

# copy the covariates to a new data.frame
covarTrain <- socTrain[,5:63]

# fit the model (takes a little while to run!)
rf.ds <- randomForest(x = covarTrain, y = dTrain, do.trace = 25)
```

2.3 Predict and validate

We use the fitted model to predict for the test data...

```
# predict at sampling sites
pred.ds <- predict(rf.ds, newdata = socTest)

# compute prediction error
pe.ds <- pred.ds - socTest$SOC
```

.. and compute the validation statistics as we did previously.

```
# ME
(me.ds <- round(mean(pe.ds),3))
```

```
## [1] 0.045
```

```
# MSE
(mse.ds <- round(mean(pe.ds**2),2))
```

```
## [1] 1.05
```

```
# RMSE
(rmse.ds <- round(sqrt(mean(pe.ds**2)),2))
```

```
## [1] 1.02
```

```
# RMedSE
(rmedse.ds <- round(sqrt(median(pe.ds**2)),2))
```

```
## [1] 0.44
```

```
# AVE
(ave.ds <- round(1-(sum(pe.ds**2)/(sum((socTest$SOC-mean(socTest$SOC))**2))),3))
```

```
## [1] 0.413
```

```
# rsq computed from correldation coefficient
(rsqCor.ds <- round((cor(pred.ds, socTest$SOC)**2),3))
```

```
## [1] 0.416
```

We will also check the OOB validation statistics calculated from the training set.

```
# OOB MSE
round(rf.ds$mse[500],2)
```

```
## [1] 3.14
```

```
# OOB RMSE
round(sqrt(rf.ds$mse[500]),2)
```

```
## [1] 1.77
```

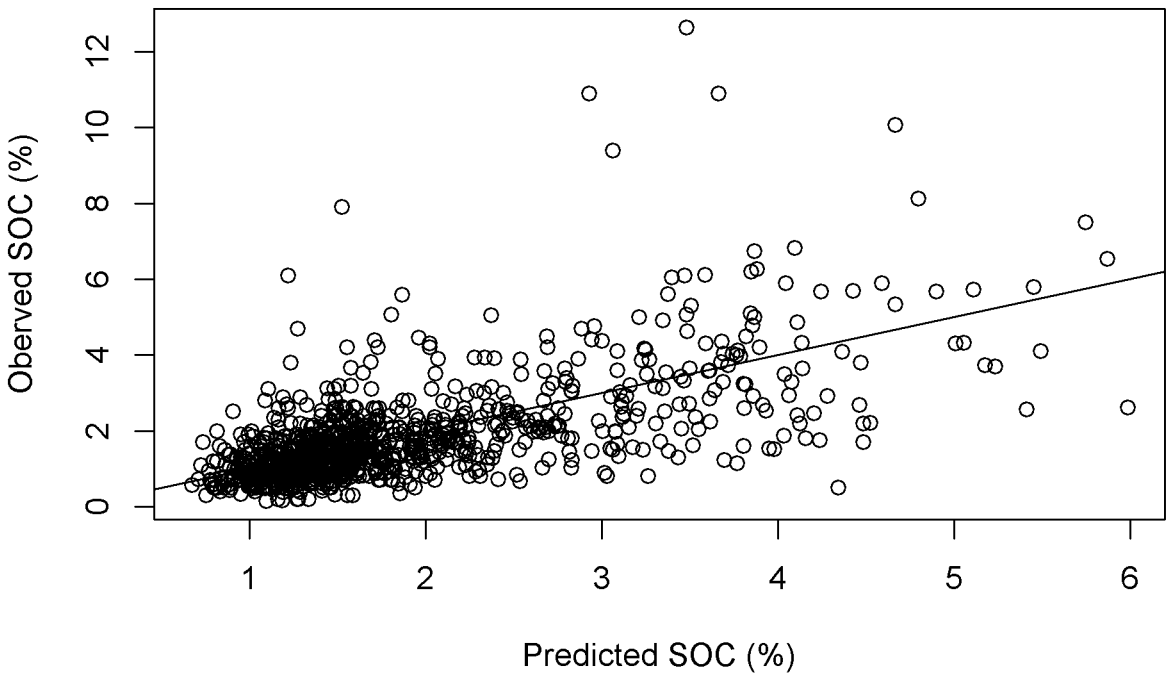
```
# OOB R2
round(rf.ds$rsq[500],3)
```

```
## [1] 0.404
```

First, we observe that the OOB validation statistics using the data splitting training set (70% of the dataset) are fairly similar to the OOB validation statistics obtained when using the full dataset for training the random forest model (see section 1.3). Apparently, leaving out 30% of the data calibration does not have a large effect on model accuracy *in this particular case*. **Note** however that these results can be very different if a different random seed number is used, i.e. the dataset will be split 70-30 in a different way leading to different validation results!

Second, we see that validation based on data splitting test set gives smaller errors and a slightly larger AVE compared to the OOB validation of the training data. Let's make a scatterplot of predicted versus observed SOC content on basis of the test dataset to explore the data splitting performance a bit further.

```
# scatterplot
plot(pred.ds, socTest$SOC, xlab="Predicted SOC (%)", ylab="Oberved SOC (%)")
abline(0,1)
```



If we compare this plot to the previous scatter plot in section 1.3 we see that the larger measured SOC values are not included in the test set. These values result in large prediction errors (as we saw in the scatterplot based on the full dataset), excluding these for validation will have a postive affect on the accuracy measures.

This illustrates that the outcome of validation based data splitting is very sensitive to the splitting of the dataset. Data splitting results are thus not very robust (stable). This is an undesirable property for validation. Preferably all data should be used for this. This can be achieved through cross-validation (see section 3 of this tutorial).

2.4 Including test data in the randomForest() function

The `randomForest()` function allows the inclusion of test data. The function will then automatically predict for the test data and compute MSE and R^2. These are stored under the `test` attribute in the model output

```
# copy soil property values to a new vectors
dTrain <- socTrain$SOC
dTest <- socTest$SOC

# copy the covariates to new data.frames
covarTrain <- socTrain[,5:63]
covarTest <- socTest[,5:63]

# fit the random forest model
rf.ds2 <- randomForest(x = covarTrain, y = dTrain, xtest = covarTest, ytest = dTest, do.trace = 25)

# inspect output
```

```
str(rf.ds2, max.level=2)
```

Access the accuracy measures of the training set (70% of the dataset).

```
# OOB MSE
round(rf.ds2$mse[500],2)
```

```
## [1] 3.17
```

```
# OOB RMSE
round(sqrt(rf.ds2$mse[500]),2)
```

```
## [1] 1.78
```

```
# OOB R2
round(rf.ds2$rsq[500],3)
```

```
## [1] 0.398
```

Access the accuracy measures of the test set (30% of the dataset).

```
# OOB MSE
round(rf.ds2$test$mse[500],2)
```

```
## [1] 1.05
```

```
# OOB RMSE
round(sqrt(rf.ds2$test$mse[500]),2)
```

```
## [1] 1.03
```

```
# OOB R2
round(rf.ds2$test$rsq[500],3)
```

```
## [1] 0.41
```

Note that the validation statistics for the test set reported here are slightly different from the validation test subset of previous subsection because a new random forest model was fitted here that will be slightly different from the one fitted before using the training subset. If the difference between the results would have been large, then model might not have been fully converged and a larger forest would then have to be grown (e.g. 1000 trees).

3 Cross-validation

Cross-validation has the advantage over data splitting that all are used for validation instead of a subset only. Again, the `caret` package offers extensive functionality for model cross-validation.

3.1 Setting up and running a cross-validation with caret

Cross-validation with the `caret` package is done with the versatile **`train`** function. One input to this function is an object that passes model training parameters to the `train` function. This object is created with **`trainControl`** function. (Check the help file for the `trainControl` function for more information: `?trainControl`.)

For cross-validation the most important arguments to this functio are the selection of the method (defined by the `method` argument; use `"cv"` for cross-validation) and the number of cross-validation folds (defined by the `number` argument). The choice for the number of folds is somewhat arbitrary. Here we use 5-fold cross-validation, which means that the dataset is splitted in 5 subsets of about equal size. Each subset is

left out once for model training. The model is then trained using the data from the other four subsets, and then used to predict at the sampling sites that were left out. This means that each sampling site gets an independent prediction, on contrary to data splitting.

```
# create an object with the training parameters
cvPar <- trainControl(
  method = "cv",
  number = 5,
  verboseIter = TRUE,
  savePredictions = TRUE
)

# inspect
str(cvPar)
```

We can now run the `train` function that does the cross-validation. As before, the sample data and covariate data have to be copied to separate objects that will be the inputs to the random forest model. Furthermore, we have to specify the `mtry` parameter that we would like to use. The value of this parameter is passed to the `tuneGrid` argument of the `train` function. The `tuneGrid` argument requires a `data.frame` so the the `mtry` value should be stored in a `data.frame`.

```
# copy soil property values to a new vectors
d <- rm$SOC

# copy the covariates to new data.frames
covar <- rm[,5:63]

# define the mtry parameter (1/3 of the number of covariates by default)
mtry <- data.frame(mtry = floor(ncol(covar)/3)) # or: rf$mtry

# cross-validation with caret package
rf.cv <- train(x = covar, y = d, method = "rf", trControl = cvPar, tuneGrid = mtry, do.trace=25)
```

3.2 Cross-validation output

Inspect the `train` output object and try to understand the output.

```
# inspect
str(rf.cv, max.level = 1)

# object class
class(rf.cv)
```

As you can see, the `train` object contains many outputs. The most relevant to us are:

- **`pred`**: a `data.frame` with the cross-validation prediction for each sampling site.
- **`resample`**: cross-validation results expressed in three accuarcy measures for each fo the five folds: the RMSE, the $R\hat{}2$ and the mean absolute error (MAE).
- **`results`**: the aggregated results for these statistics.
- **`finalModel`**: a statistical model (here a random forest model) trained on the full dataset using the optimal parameter setting. Note that the **`train`** function can also be used for parameter optimization, for instance for finding the `mtry` value that gives the most accurate predictions. In that case multiple `mtry` values will be evaluated the model that gives the most accurate result will then be returned by the `finalModel`.

Let's now take a closer look at these outputs. Start with the cross-validation predictions.

```
# cross validation predictions
str(rf.cv$pred)
```

```
## 'data.frame':    3324 obs. of  5 variables:
##  $ pred    : num  1.95 1.31 1.29 1.09 1.72 ...
##  $ obs     : num  1.32 2.38 2.05 1.3 1.43 1.47 1.62 1.1 2 43.3 ...
##  $ rowIndex: int  3 6 9 15 27 39 47 59 61 67 ...
##  $ mtry    : num  19 19 19 19 19 19 19 19 19 19 ...
##  $ Resample: chr  "Fold1" "Fold1" "Fold1" "Fold1" ...
```

```
head(rf.cv$pred)
```

```
##       pred  obs rowIndex mtry Resample
## 1 1.948924 1.32        3   19    Fold1
## 2 1.307175 2.38        6   19    Fold1
## 3 1.289620 2.05        9   19    Fold1
## 4 1.090996 1.30       15   19    Fold1
## 5 1.719426 1.43       27   19    Fold1
## 6 1.583670 1.47       39   19    Fold1
```

Attribute `pred` stores the cross-validation predictions, `obs` the observed values, `Resample` the cross-validation fold in which the sampling site was used as a test observation, `rowIndex`the corresponding row number in the original data table (here the regression matrix `rm`).

Continue with the `resample` output.

```
# results per fold
rf.cv$resample
```

```
##       RMSE  Rsquared       MAE Resample
## 1 2.284547 0.3170286 0.7681846    Fold1
## 2 1.567101 0.2281051 0.7100315    Fold2
## 3 1.109061 0.3374947 0.6457101    Fold3
## 4 1.391314 0.7111516 0.6942972    Fold4
## 5 1.042880 0.6913251 0.6588806    Fold5
```

The (large) variation in accuracy statistics between the five folds once more illustrates the instability of data splitting. Let's now look at the aggregated results.

The cross-validation statistics reported in the `resample` attribute are computed from the data in the `pred` attribute. Try if you can reproduce these from the data in the `pred` attribute (tip: select all data points for a specific fold and then compute the accuracy measures and repeat that for each fold; or use the more advanced `tapply()` function. For the latter conver the `Resample` variable to `factor`.)

Now take a look at the results aggregated over the folds.

```
# aggregated results
rf.cv$results
```

```
##   mtry     RMSE Rsquared       MAE    RMSESD RsquaredSD     MAESD
## 1   19 1.478981 0.457021 0.6954208 0.4978485  0.2268081 0.0482588
```

These results are obtained from the `resample` output by computing the average of the five folds and the the standard deviations for RMSE and *R^2*.

Finally, we take a look at the `final model`. For this model the out-of-bag statistics are reported. Note that these are computed using the full sample dataset as training dataset.

```
# MSE
round(rf.cv$finalModel$mse[500], digits = 3)
```

```
## [1] 2.57
```

```
# RMSE
round(sqrt(rf.cv$finalModel$mse[500]), digits = 3)
```

```
## [1] 1.603
```

```
# R2 (= AVE for OOB)
round(rf.cv$finalModel$rsq[500], digits = 3)
```

```
## [1] 0.392
```

We can compare these with the OOB statistics of the original random forest model.

```
# MSE
round(rf$mse[500], digits = 3)
```

```
## [1] 2.58
```

```
# RMSE
round(sqrt(rf$mse[500]), digits = 3)
```

```
## [1] 1.606
```

```
# R2 (= AVE for OOB)
round(rf$rsq[500], digits = 3)
```

```
## [1] 0.389
```

Not surprisingly these statistics are almost similar, indicating that the random forest model is stable. Furthermore, we observe that the cross-validation RMSE is smaller than the OOB RMSE and that the *R^2* is a bit larger.

3.3 Computing cross-validation statistics

Instead of using these aggregated statistics for reporting the cross-validation results, one could compute these from the cross-validation predictions which is preferable (instead of averaging RMSE values one should preferably average the fold-specific MSE values and than take the square root). This can be done as follows.

```
# MSE
round(mean((rf.cv$pred$pred-rf.cv$pred$obs)**2), digits = 3)
```

```
## [1] 2.386
```

```
# RMSE
round(sqrt(mean((rf.cv$pred$pred-rf.cv$pred$obs)**2)), digits = 3)
```

```
## [1] 1.545
```

```
# AVE
round(1-(sum((rf.cv$pred$pred-rf.cv$pred$obs)**2)/(sum((rf.cv$pred$obs-mean(rf.cv$pred$obs))**2))), digi
```

```
## [1] 0.435
```

Compare these with the OOB accuracy measures. What do you observe?

3.4 Appending predictions to the regression matrix

Alternatively, one could append the predicted values to the regression matrix and then compute the cross-validation errors and the accuracy measures.

The `rowIndex` column in the `pred` attribute refers to the row number of the sampling location in the orginial data set (here `rm`). To append the cross-validation predictions to the sample data set one must first sort the former so that the sample order is the same.

```
# copy cross-validation predictions to new data.frame
cv.pred <- rf.cv$pred

# order
cv.pred <- cv.pred[order(cv.pred$rowIndex, decreasing = FALSE),]

# append to dataset
rm$cv.pred <- cv.pred$pred

# calculate prediction error
rm$pe <- rm$cv.pred-rm$SOC
```

Try to compute the accuracy measures yourself.

Finally, the examples in this tutorial show that there are many different ways to validate that can give quite different results. None of these methods is wrong, only some are more preferable than others. It is therefore important that one always in a report or paper how the model was validated.

We have seen that the validation results are sensitive to the splitting of the data. To account for possible effects of data splitting on the validation results, one could consider repeating the cross-validation a number of times and then deriving the cross-validation accuracy statistics from the average of the repetitions. This can be done by passing the `repeats` argument to the 'trainControl()' function. Try it out!

That's it. Don't forget to save your outputs!

```
# save various outputs
save.image("validation.rda")
```

Acknowledgements

This tutorial was developed with support from the Cereal Systems Initiative for South Asia (CSISA), Bill and Melinda Gates Foundation and CIMMYT.

The soil data used in this tutorial is from the Macedonian Soil Information System (Vrscaj et al. 2017), and provided by the Secretariat of the Global Soil Partnership. The raw data was preprocessed into an `R data.frame` that is provided with this tutorial.

The covariate layers were all derived from freely available global biophysical data and have been made available by ISRIC for all territories in the world at 1km resolution from ftp://isric.org (user: gsp, password: gspisric).

References

Brus, D. J., B. Kempen, and G. B. M. Heuvelink. 2011. "Sampling for Validation of Digital Soil Maps." *European Journal of Soil Science* 62 (3): 394–407. doi:10.1111/j.1365-2389.2011.01364.x.

FAO. 2018. "Soil Organic Carbon Mapping Cookbook. Second Edition." Food and Agricultural Organization of the United Nations, Rome. http://www.fao.org/3/a-bs901e.pdf.

Vrscaj, B., L. Poggio, D. Muaketov, and R. Vargas. 2017. "Utilizing the Legacy Soil Data of Macedonia: The Creation of the Macedonian Soil Information System and its use for digital soil mapping and assessment

applications. Abstract Book of Pedometrics 2017, Wageningen, 26 June - 1 July 2017." http://www.pedometrics2017.org/s/Abstract-Book-Pedometrics-2017.pdf.