

Some methods for the quantification of prediction uncertainties for digital soil mapping

Soil Security Laboratory

2017

Soil scientists are quite aware of the current issues concerning the natural environment because our expertise is intimately aligned with their understanding and alleviation. We know that sustainable soil management alleviates soil degradation, improves soil quality and will ultimately ensure food security. Critical to better soil management is information detailing the soil resource, its processes and its variation across landscapes. Consequently, under the broad umbrella of “environmental monitoring”, there has been a growing need to acquire quantitative soil information (McBratney et al., 2003; Grimm and Behrens, 2010). The concerns of soil-related issues in reference to environmental management were raised by McBratney (1992) when stating that it is our duty as soil scientists, to ensure that the information we provide to the users of soil information is both accurate and precise, or at least of known accuracy and precision.

However, a difficulty we face is that soil can vary, seemingly erratically in the context of space and time (Webster, 2000). Thus the conundrum in model-based predictions of soil phenomena is that models are not “error free”. The unpredictability of soil variation combined with simplistic representations of complex soil processes inevitably leads to errors in model outputs.

We do not know the true character and processes of soils and our models are merely abstractions of these real processes. We know this; or in other words, in the absence of such confidence, we know we are uncertain about the true properties and processes that characterize soils (Brown and Heuvelink, 2005). The key is therefore to determine to what extent our uncertainties are propagated through a model of which effect the final predictions of a real-world process.

In modeling exercises, uncertainty of the model output is the summation of the three main sources generally described as: model structure uncertainty, model parameter uncertainty and model input uncertainty (Minasny and McBratney, 2002; Brown and Heuvelink, 2005). A detailed analysis of the contribution of each of the different sources of uncertainty is generally recommended. In this book chapter we will cover few approaches to estimate the uncertainty of model outputs. Essentially what this means is that given a defined level of confidence, model predictions from digital soil mapping will be co-associated with the requisite prediction interval or range. The approaches for quantifying the prediction uncertainties are:

-
- Universal kriging prediction variance.
 - bootstrapping
 - Empirical uncertainty quantification through data partitioning and cross validation.
 - Empirical uncertainty quantification through fuzzy clustering and cross validation

The data that will be used in this chapter is a small data set of subsoil pH that has been collected since 2001 to present from the Lower Hunter Valley in New South Wales, Australia. The soil data covers an area of approximately 220km². Validation of the quantification of uncertainty will be performed using a subset of these data. The mapping of the uncertainties will be conducted for a small region of the study area. The data for this section can be retrieved from the `ithir` package. The soil data is called `HV_subsoilpH` while the grids of environmental covariates is called `hunterCovariates_sub`.

1 Universal kriging prediction variance

In the chapter regarding digital soil mapping of continuous variables, universal kriging was explored. This model is ideal from the perspective that both the correlated variables and model residuals are handled simultaneously. This model also automatically generates prediction uncertainty via the kriging variance. It is with this variance estimate that we can define a prediction interval. For this example and the following, a 90% prediction interval will be defined for the mapping purposes. Although for validation, a number of levels of confidence will be defined and subsequently validated in order to assess the performance and sensitivity of the uncertainty estimates.

1.1 Defining the model parameters

First we need to load in all the libraries that are necessary for this section and load in the necessary data.

```
library(ithir)
library(sp)
library(rgdal)
library(raster)
library(gstat)

# Point data
data(HV_subsoilpH)
str(HV_subsoilpH)

## 'data.frame': 506 obs. of  14 variables:
##  $ X                : num  340386 340345 340559 340483 340734 ...
##  $ Y                : num  6368690 6368491 6369168 6368740 6368964 ...
##  $ pH60_100cm       : num  4.47 5.42 6.26 8.03 8.86 ...
```

```
## $ Terrain_Ruggedness_Index: num 1.34 1.42 1.64 1.04 1.27 ...
## $ AACN : num 1.619 0.281 2.301 1.74 3.114 ...
## $ Landsat_Band1 : int 57 47 59 52 62 53 47 52 53 63 ...
## $ Elevation : num 103.1 103.7 99.9 101.9 99.8 ...
## $ Hillshading : num 1.849 1.428 0.934 1.517 1.652 ...
## $ Light_insolation : num 1689 1701 1722 1688 1735 ...
## $ Mid_Slope_Positon : num 0.876 0.914 0.844 0.848 0.833 ...
## $ MRVBF : num 3.85 3.31 3.66 3.92 3.89 ...
## $ NDVI : num -0.143 -0.386 -0.197 -0.14 -0.15 ...
## $ TWI : num 17.5 18.2 18.8 18 17.8 ...
## $ Slope : num 1.79 1.42 1.01 1.49 1.83 ...

# Raster data
data(hunterCovariates_sub)
hunterCovariates_sub

## class : RasterStack
## dimensions : 249, 210, 52290, 11 (nrow, ncol, ncell, nlayers)
## resolution : 25, 25 (x, y)
## extent : 338422.3, 343672.3, 6364203, 6370428 (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=utm +zone=56 +south +ellps=WGS84 +datum=WGS84 +units=m +no_defs
## names : Terrain_Ruggedness_Index, AACN, Landsat_Band1, Elevation, Hillshading
## min values : 0.194371, 0.000000, 26.000000, 72.217499, 0.000000
## max values : 15.945321, 106.665482, 140.000000, 212.632507, 32.4
```

You will notice for `HV_subsoilpH` that these data have already been intersected with a number of covariates. The `hunterCovariates_sub` are a `rasterStack` of the same covariates (although the spatial extent is smaller).

Now to prepare the data for the universal kriging model.

```
# subset data for modeling
set.seed(123)
training <- sample(nrow(HV_subsoilpH), 0.7 * nrow(HV_subsoilpH))
cDat <- HV_subsoilpH[training, ]
vDat <- HV_subsoilpH[-training, ]
nrow(cDat)

## [1] 354

nrow(vDat)

## [1] 152
```

The `cDat` and `vDat` objects correspond to the model calibration and validation data sets respectively.

Now to prepare the data for the model

```
# coordinates
coordinates(cDat) <- ~X + Y

# remove CRS from grids
```

```
crs(hunterCovariates_sub) <- NULL
```

We will firstly use a step wise regression to determine a parsimonious model are the most important covariates.

```
# Full model
```

```
lm1 <- lm(pH60_100cm ~ Terrain_Ruggedness_Index + AACN + Landsat_Band1 + Elevation +  
Hillshading + Light_insolation + Mid_Slope_Positon + MRVBF + NDVI + TWI +  
Slope, data = cDat)
```

```
# Parsimous model
```

```
lm2 <- step(lm1, direction = "both", trace = 0)  
as.formula(lm2)
```

```
## pH60_100cm ~ AACN + Landsat_Band1 + Hillshading + Mid_Slope_Positon +  
## MRVBF + NDVI + TWI
```

```
summary(lm2)
```

```
##
```

```
## Call:
```

```
## lm(formula = pH60_100cm ~ AACN + Landsat_Band1 + Hillshading +  
## Mid_Slope_Positon + MRVBF + NDVI + TWI, data = cDat)
```

```
##
```

```
## Residuals:
```

```
##      Min       1Q   Median       3Q      Max  
## -2.9409 -0.8467 -0.1431  0.6870  3.2195
```

```
##
```

```
## Coefficients:
```

```
##              Estimate Std. Error t value Pr(>|t|)  
## (Intercept)    6.55363    1.00729   6.506 2.70e-10 ***  
## AACN           0.02652    0.00579   4.580 6.48e-06 ***  
## Landsat_Band1  -0.04391    0.01119  -3.925 0.000104 ***  
## Hillshading     0.07651    0.02139   3.576 0.000398 ***  
## Mid_Slope_Positon 0.88822    0.31849   2.789 0.005582 **  
## MRVBF           0.28889    0.09624   3.002 0.002878 **  
## NDVI            5.88079    1.06282   5.533 6.22e-08 ***  
## TWI             0.11132    0.05657   1.968 0.049889 *
```

```
## ---
```

```
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
##
```

```
## Residual standard error: 1.185 on 346 degrees of freedom
```

```
## Multiple R-squared:  0.2447, Adjusted R-squared:  0.2294
```

```
## F-statistic: 16.01 on 7 and 346 DF, p-value: < 2.2e-16
```

Now we can construct the universal kriging model using the step wise selected covariates.

```
vgm1 <- variogram(pH60_100cm ~ AACN + Landsat_Band1 + Hillshading + Mid_Slope_Positon +  
MRVBF + NDVI + TWI, cDat, width = 200)  
mod <- vgm(psill = var(cDat$pH60_100cm), "Sph", range = 10000, nugget = 0)  
model_1 <- fit.variogram(vgm1, mod)
```

```

gUK <- gstat(NULL, "hunterpH_UK", pH60_100cm ~ AACN + Landsat_Band1 + Hillshading +
  Mid_Slope_Positon + MRVBF + NDVI + TWI, cDat, model = model_1)
gUK

## data:
## hunterpH_UK : formula = pH60_100cm`~`AACN + Landsat_Band1 + Hillshading + Mid_Slope_Pos
## variograms:
##              model      psill      range
## hunterpH_UK[1]   Nug 0.9989982    0.000
## hunterpH_UK[2]   Sph 1.2847147 6959.286

```

1.2 Spatial mapping

Here we want to produce four maps that will correspond to:

1. The lower end of the 90% prediction interval or 5th percentile.
2. the universal kriging prediction.
3. The upper end of the 90% prediction interval or 95th percentile.
4. The prediction interval range.

For the prediction we use the raster `interpolate` function.

```

UK.P.map <- interpolate(hunterCovariates_sub, gUK, xyOnly = FALSE, index = 1,
  filename = "UK_predMap.tif", format = "GTiff", overwrite = T)

```

Setting the `index` value to 2 lets us map the kriging variance which is needed for the prediction interval. Taking the square root this estimates the standard deviation which we can then multiple for the z value that corresponds to a 90% probability which is 1.644854. We then both add and subtract that result from the universal kriging prediction to derive the 90% prediction limits.

```

# prediction variance
UK.var.map <- interpolate(hunterCovariates_sub, gUK, xyOnly = FALSE, index = 2,
  filename = "UK_predVarMap.tif", format = "GTiff", overwrite = T)

# standard deviation
f2 <- function(x) (sqrt(x))
UK.stdev.map <- calc(UK.var.map, fun = f2,
  filename = "UK_predSDMap.tif", format = "GTiff",
  progress = "text", overwrite = T)

# Z level
zlev <- qnorm(0.95)
f2 <- function(x) (x * zlev)
UK.mult.map <- calc(UK.stdev.map, fun = f2,
  filename = "UK_multMap.tif", format = "GTiff",
  progress = "text", overwrite = T)

```

```

# Add and subtract mult from prediction
m1 <- stack(UK.P.map, UK.mult.map)

# upper PL
f3 <- function(x) (x[1] + x[2])
UK.upper.map <- calc(m1, fun = f3, filename = "UK_upperMap.tif", format = "GTiff",
  progress = "text", overwrite = T)

# lower PL
f4 <- function(x) (x[1] - x[2])
UK.lower.map <- calc(m1, fun = f4, filename = "UK_lowerMap.tif", format = "GTiff",
  progress = "text", overwrite = T)

Finally to derive the 90% prediction limit range

# prediction range
m2 <- stack(UK.upper.map, UK.lower.map)

f5 <- function(x) (x[1] - x[2])
UK.piRange.map <- calc(m2, fun = f5, filename = "UK_piRangeMap.tif", format = "GTiff",
  progress = "text", overwrite = T)

```

So to plot them all together we use the following script. Here we explicitly create a color ramp that follows reasonably closely the pH color ramp. Then we scale each map to the common range for better comparison (Figure 1).

```

# color ramp
phCramp <- c("#d53e4f", "#f46d43", "#fdae61", "#fee08b", "#ffffbf", "#e6f598",
  "#abdda4", "#66c2a5", "#3288bd", "#5e4fa2", "#542788", "#2d004b")
brk <- c(2:14)
par(mfrow = c(2, 2))
plot(UK.lower.map, main = "90% Lower prediction limit", breaks = brk, col = phCramp)
plot(UK.P.map, main = "Prediction", breaks = brk, col = phCramp)
plot(UK.upper.map, main = "90% Upper prediction limit", breaks = brk, col = phCramp)
plot(UK.piRange.map, main = "Prediction limit range",
  col = terrain.colors(length(seq(0, 6.5, by = 1)) - 1), axes = FALSE,
  breaks = seq(0, 6.5, by = 1))

```

1.3 Validating the quantification of uncertainty

One of the ways to assess the performance of the uncertainty quantification is to evaluate the occurrence of times where an observed value is encapsulated by an associated prediction interval. Given a stated level of confidence, we should also expect to find the same percentage of observations encapsulated by its associated prediction interval. We define this percentage as the prediction interval coverage probability (PICP). The PICP was used in both Solomatine and Shrestha (2009) and Malone et al. (2011). To assess the sensitivity of the uncertainty quantification, we define prediction intervals at a number of levels of confidence and then assess the PICP. Ideally, a 1:1 relationship would

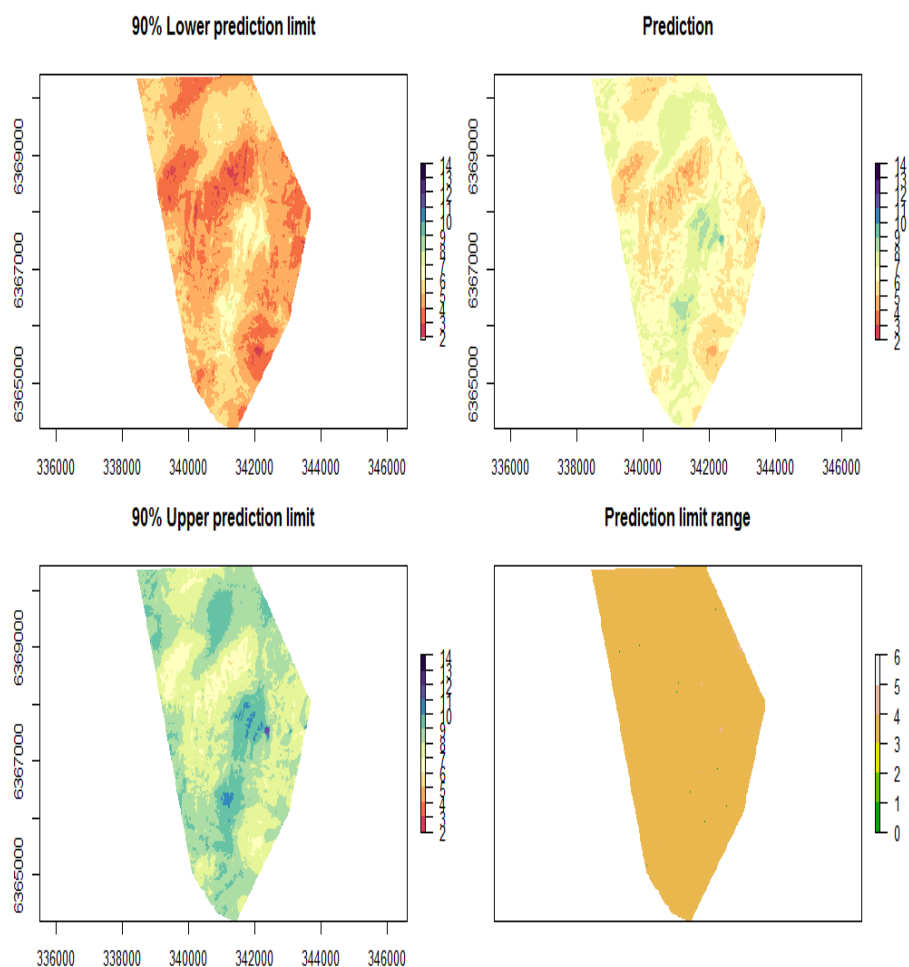


Figure 1: Soil pH predictions and prediction limits derived using a universal kriging model

ensue.

First we apply the universal kriging model `gUK` to the validation data in order to estimate pH and the prediction variance.

```
coordinates(vDat) <- ~X + Y

# Prediction
UK.preds.V <- as.data.frame(krige(pH60_100cm ~ AACN + Landsat_Band1 + Hillshading +
  Mid_Slope_Positon + MRVBF + NDVI + TWI, cDat, model = model_1, newdata = vDat))

## [using universal kriging]
UK.preds.V$stdev <- sqrt(UK.preds.V$var1.var)
str(UK.preds.V)
```

```
## 'data.frame': 152 obs. of 5 variables:
## $ X      : num  340559 340780 340861 340905 341131 ...
## $ Y      : num  6369168 6369166 6368874 6368790 6368945 ...
## $ var1.pred: num  6.65 7.16 6.4 5.77 5.72 ...
## $ var1.var : num  1.12 1.14 1.13 1.15 1.13 ...
## $ stdev    : num  1.06 1.07 1.06 1.07 1.06 ...
```

Then we define a vector of z values for a sequence of probabilities using the `qnorm` function.

```
qp <- qnorm(c(0.995, 0.9875, 0.975, 0.95, 0.9, 0.8, 0.7, 0.6, 0.55, 0.525))
```

Then we estimate the prediction limits for each confidence level.

```
# zfactor multiplication
vMat <- matrix(NA, nrow = nrow(UK.preds.V), ncol = length(qp))
for (i in 1:length(qp)) {
  vMat[, i] <- UK.preds.V$stdev * qp[i]
}

# upper
uMat <- matrix(NA, nrow = nrow(UK.preds.V), ncol = length(qp))
for (i in 1:length(qp)) {
  uMat[, i] <- UK.preds.V$var1.pred + vMat[, i]
}

# lower
lMat <- matrix(NA, nrow = nrow(UK.preds.V), ncol = length(qp))
for (i in 1:length(qp)) {
  lMat[, i] <- UK.preds.V$var1.pred - vMat[, i]
}
```

Then we want to evaluate the PICP for each confidence level.

```
bMat <- matrix(NA, nrow = nrow(UK.preds.V), ncol = length(qp))
for (i in 1:ncol(bMat)) {
  bMat[, i] <- as.numeric(vDat$pH60_100cm <= uMat[, i] & vDat$pH60_100cm >=
    lMat[, i])
}

colSums(bMat)/nrow(bMat)

## [1] 0.98684211 0.97368421 0.92105263 0.88157895 0.83552632 0.62500000
## [7] 0.46052632 0.25657895 0.08552632 0.03289474
```

Plotting the confidence level against the PICP provides a visual means to assess the fidelity about the 1:1 line. As can be seen on Figure 2, the PICP follows closely the 1:1 line.

```
# make plot
cs <- c(99, 97.5, 95, 90, 80, 60, 40, 20, 10, 5)
plot(cs, ((colSums(bMat)/nrow(bMat)) * 100))
```

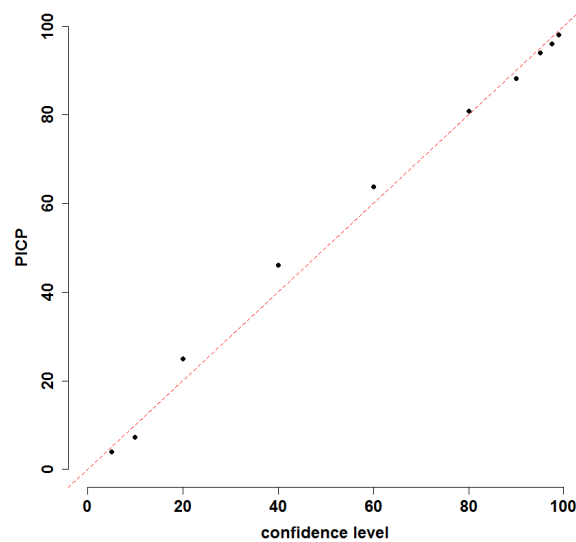


Figure 2: Plot of PICP and confidence level based on validation of universal kriging model.

So to summarize. We may evaluate the performance of the universal kriging model on the basis of the predictions. Using the validation data we would use the `goof` function for that purpose.

```
goof(observed = vDat$pH60_100cm, predicted = UK.preds.V$var1.pred)

##          R2 concordance      MSE      RMSE      bias
## 1 0.3585243 0.5037044 1.162424 1.078158 0.1464993
```

And then we may assess the uncertainties on the basis of the PICP like shown on Figure 2, together with assessing the quantiles of the distribution of the prediction limit range for a given prediction confidence level (here 90%).

```
cs <- c(99, 97.5, 95, 90, 80, 60, 40, 20, 10, 5) # confidence level
colnames(lMat) <- cs
colnames(uMat) <- cs
quantile(uMat[, "90"] - lMat[, "90"])

##          0%          25%          50%          75%          100%
## 3.403896 3.466760 3.490894 3.526162 3.700848
```

As can be noted above, the prediction interval range is relatively homogeneous and this is corroborated on the associated map in Figure 1.

2 Bootstrapping

Bootstrapping is a popular non-parametric approach for quantifying prediction uncertainties (Efron and Tibshirani, 1993). Bootstrapping involves repeated random sampling with replacement of the available data. With the bootstrap sample, a model is fitted, and can then be applied to generate a digital soil map. By repeating the process of random sampling and applying the model, we are able to generate probability distributions of the prediction realizations from each model at each pixel. A robust estimate may be determined by taking the average of all the simulated predictions at each pixel. By being able to obtain probability distributions of the outcomes, one is also able to quantify the uncertainty of the modeling by computing a prediction interval given a specified level of confidence. While the bootstrapping approach is relatively straightforward, there is a requirement to generate x number of maps, where x is the number of bootstrap samples. This obviously could be prohibitive from a computational and data storage point of view, but not altogether impossible (given parallel processing capabilities etc.) as was demonstrated by both Viscarra Rossel et al. (2015) and Liddicoat et al. (2015) whom both performed bootstrapping for quantification of uncertainties across very large mapping extents. In the case of Viscarra Rossel et al. (2015) this for the entire Australian continent at 100m resolution.

In the example below, the bootstrap method is demonstrated. We will be using Cubist modeling for the model structure and perform 50 bootstrap samples. We will use 70% of the available data to use for fitting models. The remaining 30% as has been done for all previous DSM approaches is for validation. Of the 70% of the available data for model fitting, 70% of these data are to be used for each bootstrap sample i.e. a random sample with replacement.

2.1 Defining the model parameters

For the first step, we do the random partitioning of the data into calibration and validation data sets. Again we are using the `HV_subsoilpH` data and the associated `hunterCovariates_sub` raster data stack.

```
# subset data for modeling
set.seed(667)
training <- sample(nrow(HV_subsoilpH), 0.7 * nrow(HV_subsoilpH))
cDat <- HV_subsoilpH[training, ]
vDat <- HV_subsoilpH[-training, ]
```

The `nbag` variable below holds the value for the number of bootstrap models we want to fit. Here it is 50. Essentially the bootstrap can be contained within a `for` loop, where upon each loop a sample of the available data is taken (here 70%) then a model is fitted. Note below the use of the `replace` parameter to indicate we want random sample with replacement. After a model is fitted, we save the model to file and will come back to it later. The `modelFile` variable shows the extensive use of the `paste` function in order to provide the pathway and file name for the model that we want to save on each

loop iteration. The `saveRDS` function allows us to save each of the model objects as rds files to the location specified. An alternative to save the models individually to file is to save them to elements within a `list`. When dealing with very large numbers of models and additionally are complex in terms of their parameterizations, the save to `list` elements alternative could run into computer memory limitation issues. The last section of the script below just demonstrates the use of the `list.files` functions to confirm that we have saved those models to file and they are ready to use.

```
# Number of bootstraps
nbag <- 50

# Fit cubist models for each bootstrap
library(Cubist)
for (i in 1:nbag) {
  trainingREP <- sample.int(nrow(cDat), 0.7 * nrow(cDat), replace = TRUE)

  fit_cubist <- cubist(x = cDat[trainingREP, c("Terrain_Ruggedness_Index",
    "AACN", "Landsat_Band1", "Elevation", "Hillshading", "Light_insolation",
    "Mid_Slope_Positon", "MRVBF", "NDVI", "TWI", "Slope")],
    y = cDat$pH60_100cm[trainingREP], cubistControl(rules = 5, extrapolation = 5),
    committees = 1)

  modelFile <- paste(paste(paste(paste(getwd(), "/folder_location/",
    sep = ""), "bootMod_", sep = ""), i, sep = ""), ".rds", sep = "")
  saveRDS(object = fit_cubist, file = modelFile)
}

# list all files in directory
c.models <- list.files(path = paste(getwd(), "/folder_location",
  sep = ""), pattern = "\\..rds$", full.names = TRUE)
head(c.models)

## [1] "C:/~/bootMod_1.rds"
## [2] "C:/~/bootMod_10.rds"
## [3] "C:/~/bootMod_11.rds"
## [4] "C:/~/bootMod_12.rds"
## [5] "C:/~/bootMod_13.rds"
## [6] "C:/~/bootMod_14.rds"
```

We can then assess the goodness of fit and validation statistics of the bootstrap models. This is done using the `goof` function as in previous examples. This time we incorporate that function within a `for` loop. For each loop, we read in the model via the `radRDS` function and then save the diagnostics to the `cubiMat` matrix object. After the iterations are completed, we use the `colMeans` function to calculate the means of the diagnostics over the 50 model iterations. You could also assess the variance of those means by a command such as `var(cubiDat[,1])`, which would return the variance of the R^2 values.

```

# calibration data
cubiMat <- matrix(NA, nrow = nbag, ncol = 5)
for (i in 1:nbag) {
  fit_cubist <- readRDS(c.models[i])
  cubiMat[i, ] <- as.matrix(goof(observed = cDat$pH60_100cm,
    predicted = predict(fit_cubist, newdata = cDat)))
}
cubiDat <- as.data.frame(cubiMat)
names(cubiDat) <- c("R2", "concordance", "MSE", "RMSE", "bias")
colMeans(cubiDat)

##           R2 concordance           MSE           RMSE           bias
## 0.25261147 0.45185565 1.46214146 1.20887737 -0.06697598

# Validation data
cubPred.V <- matrix(NA, ncol = nbag, nrow = nrow(vDat))
cubiMat <- matrix(NA, nrow = nbag, ncol = 5)
for (i in 1:nbag) {
  fit_cubist <- readRDS(c.models[i])
  cubPred.V[, i] <- predict(fit_cubist, newdata = vDat)
  cubiMat[i, ] <- as.matrix(goof(observed = vDat$pH60_100cm,
    predicted = predict(fit_cubist, newdata = vDat)))
}
cubPred.V_mean <- rowMeans(cubPred.V)

cubiDat <- as.data.frame(cubiMat)
names(cubiDat) <- c("R2", "concordance", "MSE", "RMSE", "bias")
colMeans(cubiDat)

##           R2 concordance           MSE           RMSE           bias
## 0.09010054 0.26690013 1.80777927 1.34013203 0.11262625

# Average validation MSE
avGMSE <- mean(cubiDat[, 3])

```

For the validation data, in addition to deriving the model diagnostic statistics, we are also saving the actual model predictions for these data for each iteration to the `cubPred.V` object. These will be used further on for validating the prediction uncertainties. The last line of the script above saves the mean of the mean square error (MSE) estimates from the validation data. The independent MSE estimator, accounts for both systematic and random errors in the modeling. This estimate of the MSE is needed for quantifying the uncertainties, as this error is in addition to that which are accounted for by the bootstrap, which are specifically those associated with the deterministic model component i.e. the model relationship between target variable and the covariates. Subsequently an overall prediction variance (at each point or pixel) will be the sum of the random error component (MSE) and the bootstrap prediction variance (as estimated from the mean of the realisations from the bootstrap modeling).

2.2 Spatial mapping

Our initial purpose here is to derive the mean and the variance of the predictions from each bootstrap sample. This requires loading in each bootstrap model, applying into the covariate data, then saving the predicted map to file or R memory. In the case below the predictions are saved to file. This is illustrated in the following script.

```
for (i in 1:nbag) {  
  fit_cubist <- readRDS(c.models[i])  
  mapFile <- paste(paste(paste(paste(getwd(), "/folder_location_of_rasters/",  
    sep = ""), "bootMap_", sep = ""), i, sep = ""), ".tif", sep = "")  
  predict(hunterCovariates_sub, fit_cubist, filename = mapFile,  
    format = "GTiff", overwrite = T)  
}
```

To evaluate the mean at each pixel from each of the created maps, the base function `mean` can be applied to a given stack of rasters. First we need to get the path location of the rasters. Notice from the `list.files` function and the `pattern` parameter, we are restricting the search of rasters that contain the string “bootMap”. Next we make a stack of those rasters, followed by the calculation of the mean, which is also written directly to file.

```
# Pathway to rasters  
files <- list.files(paste(getwd(), "/folder_location_of_rasters/",  
  sep = ""), pattern = "bootMap", full.names = TRUE)  
  
# Raster stack  
r1 <- raster(files[1])  
for (i in 2:length(files)) {  
  r1 <- stack(r1, files[i])  
}  
  
# Calculate mean  
meanFile <- paste(paste(paste(getwd(), "/folder_location_of_rasters/",  
  sep = ""), "meanPred_", sep = ""), ".tif", sep = "")  
bootMap.mean <- writeRaster(mean(r1), filename = meanFile,  
  format = "GTiff", overwrite = TRUE)
```

There is not a simple R function to use in order to estimate the variance at each pixel from the prediction maps. Therefore we resort to estimating it directly from the standard equation:

$$Var(X) = \frac{1}{1-n} \sum_{i=1}^n (x_i - \mu)^2 \quad (1)$$

The symbol μ in this case is the mean bootstrap prediction, and x_i is the i th bootstrap map. In the first step below, we estimate the square differences and save the maps to file. Then we calculate the sum of those squared differences, before deriving the variance prediction. The last step is to add the variance of the bootstrap predictions to the averaged MSE estimated from the validation

data.

```
# Square differences
for (i in 1:length(files)) {
  r1 <- raster(files[i])
  diffFile <- paste(paste(paste(paste(getwd(), "/folder_location_of_rasters/",
    sep = ""), "bootAbsDif_", sep = ""), i, sep = ""), ".tif", sep = "")
  jj <- (r1 - bootMap.mean)^2
  writeRaster(jj, filename = diffFile, format = "GTiff", overwrite = TRUE)
}

# calculate the sum of square differences Look for files with the bootAbsDif
# character string in file name
files2 <- list.files(paste(getwd(), "/folder_location_of_rasters/",
  sep = ""), pattern = "bootAbsDif", full.names = TRUE)

# stack
r2 <- raster(files2[1])
for (i in 2:length(files2)) {
  r2 <- stack(r1, files2[i])
}

sqDiffFile <- paste(paste(paste(getwd(), "/folder_location_of_rasters/",
  sep = ""), "sqDiffPred_", sep = ""), ".tif", sep = "")
bootMap.sqDiff <- writeRaster(sum(r2), filename = sqDiffFile, format = "GTiff",
  overwrite = TRUE)

# Variance
varFile <- paste(paste(paste(getwd(), "/folder_location_of_rasters/",
  sep = ""), "varPred_", sep = ""), ".tif", sep = "")
bootMap.var <- writeRaster(((1/(nbag - 1)) * bootMap.sqDiff), filename = varFile,
  format = "GTiff", overwrite = TRUE)

# Overall prediction variance
varFile2 <- paste(paste(paste(getwd(), "/folder_location_of_rasters/",
  sep = ""), "varPredF_", sep = ""), ".tif", sep = "")
bootMap.varF <- writeRaster((bootMap.var + avGMSE),
  filename = varFile, format = "GTiff",
  overwrite = TRUE)
```

To derive to 90% prediction interval we take the square root of the variance estimate and multiply that value by the z value that corresponds to a 90% probability. The z value is obtained using the `qnorm` function. The result is then either added or subtracted to the mean prediction in order to generate the upper and lower prediction limits respectively.

```
# Standard deviation
sdFile <- paste(paste(paste(getwd(), "/folder_location_of_rasters/",
  sep = ""), "sdPred_", sep = ""), ".tif", sep = "")
bootMap.sd <- writeRaster(sqrt(bootMap.varF), filename = sdFile,
```

```

format = "GTiff", overwrite = TRUE)

# standard error
seFile <- paste(paste(paste(getwd(), "/folder_location_of_rasters/",
sep = ""), "sePred_", sep = ""), ".tif", sep = "")
bootMap.se <- writeRaster((bootMap.sd * qnorm(0.95)), filename = seFile,
format = "GTiff", overwrite = TRUE)

# upper prediction limit
uplFile <- paste(paste(paste(getwd(), "/folder_location_of_rasters/",
sep = ""), "uplPred_", sep = ""), ".tif", sep = "")
bootMap.upl <- writeRaster((bootMap.mean + bootMap.se), filename = uplFile,
format = "GTiff", overwrite = TRUE)

# lower prediction limit
lplFile <- paste(paste(paste(getwd(), "/folder_location_of_rasters/",
sep = ""), "lplPred_", sep = ""), ".tif", sep = "")
bootMap.lpl <- writeRaster((bootMap.mean - bootMap.se), filename = lplFile,
format = "GTiff", overwrite = TRUE)

# prediction interval range
pirFile <- paste(paste(paste(getwd(), "/folder_location_of_rasters/",
sep = ""), "pirPred_", sep = ""), ".tif", sep = "")
bootMap.pir <- writeRaster((bootMap.upl - bootMap.lpl), filename = pirFile,
format = "GTiff", overwrite = TRUE)

```

As for the Universal kriging example, we can plot the associated maps of the predictions and quantified uncertainties (Figure 3).

```

phCramp <- c("#d53e4f", "#f46d43", "#fdae61", "#fee08b", "#ffffbf", "#e6f598",
"#abdda4", "#66c2a5", "#3288bd", "#5e4fa2", "#542788", "#2d004b")
brk <- c(2:14)
par(mfrow = c(2, 2))
plot(bootMap.lpl, main = "90% Lower prediction limit", breaks = brk, col = phCramp)
plot(bootMap.mean, main = "Prediction", breaks = brk, col = phCramp)
plot(bootMap.upl, main = "90% Upper prediction limit", breaks = brk, col = phCramp)
plot(bootMap.pir, main = "Prediction limit range",
col = terrain.colors(length(seq(0, 6.5, by = 1)) - 1), axes = FALSE,
breaks = seq(0, 6.5, by = 1))

```

2.3 Validating the quantification of uncertainty

You will recall the bootstrap model predictions on the validation data were saved to the `cubPred.V` object. We want estimate the standard deviation of those predictions for each point. Also recall that the prediction variance is the sum of the MSE and the bootstrap models prediction variance. Taking the square root of that summation results in standard deviation estimate.

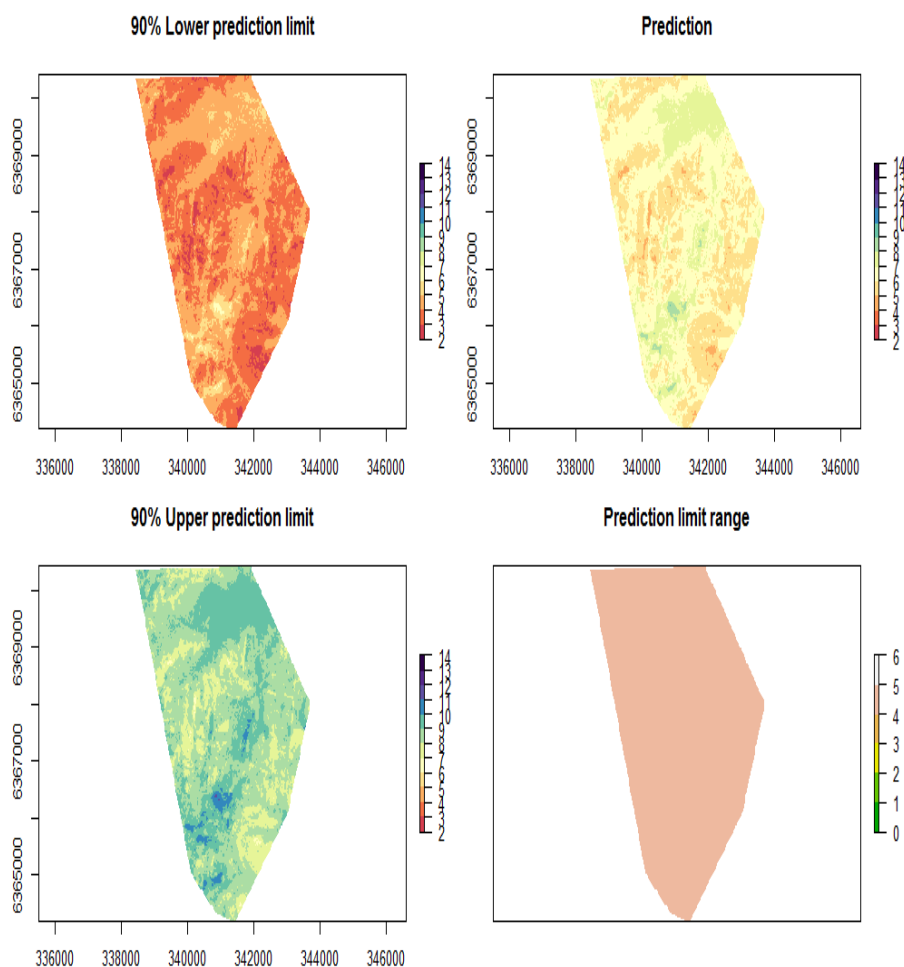


Figure 3: Soil pH predictions and prediction limits derived using bootstrapping.

```
val.sd <- matrix(NA, ncol = 1, nrow = nrow(cubPred.V))
for (i in 1:nrow(cubPred.V)) {
  val.sd[i, 1] <- sqrt(var(cubPred.V[i, ]) + avGMSE)
}
```

We then need to multiply the standard deviation by the corresponding percentile of the standard normal distribution in order to express the prediction limits at each level of confidence. Note the use of the `for` loop and the associated cycling through of the different percentile values.

```
# Percentiles of normal distribution
qp <- qnorm(c(0.995, 0.9875, 0.975, 0.95, 0.9, 0.8, 0.7, 0.6, 0.55, 0.525))

# zfactor multiplication
vMat <- matrix(NA, nrow = nrow(cubPred.V), ncol = length(qp))
```

```

for (i in 1:length(qp)) {
  vMat[, i] <- val.sd * qp[i]
}

```

Now we add or subtract the limits to/from the averaged model predictions to derive to prediction limits for each level of confidence.

```

# upper prediction limit
uMat <- matrix(NA, nrow = nrow(cubPred.V), ncol = length(qp))
for (i in 1:length(qp)) {
  uMat[, i] <- cubPred.V_mean + vMat[, i]
}

# lower prediction limit
lMat <- matrix(NA, nrow = nrow(cubPred.V), ncol = length(qp))
for (i in 1:length(qp)) {
  lMat[, i] <- cubPred.V_mean - vMat[, i]
}

```

Now we assess the PICP for each level confidence. Recalling that we are simply assessing whether the observed value is encapsulated by the corresponding prediction limits, then calculating the proportion of agreement to total number of observations.

```

bMat <- matrix(NA, nrow = nrow(cubPred.V), ncol = length(qp))
for (i in 1:ncol(bMat)) {
  bMat[, i] <- as.numeric(vDat$pH60_100cm <= uMat[, i] & vDat$pH60_100cm >=
    lMat[, i])
}

colSums(bMat)/nrow(bMat)

## [1] 1.00000000 1.00000000 0.99342105 0.96710526 0.90131579 0.69078947
## [7] 0.45394737 0.21052632 0.08552632 0.03289474

```

As can be seen on Figure 4, there is an indication that the prediction uncertainties could be a little too liberally defined, where particularly at the higher level of confidence the associated PICP is higher.

```

# make plot
cs <- c(99, 97.5, 95, 90, 80, 60, 40, 20, 10, 5) # confidence level
plot(cs, ((colSums(bMat)/nrow(bMat)) * 100))

```

Quantiles of the distribution of the prediction limit range are express below for the validation data (in terms of the 90% level of confidence). Compared to the universal kriging approach, the uncertainties quantified from the bootstrapping approach are higher in general.

```

cs <- c(99, 97.5, 95, 90, 80, 60, 40, 20, 10, 5) # confidence level
colnames(lMat) <- cs
colnames(uMat) <- cs
quantile(uMat[, "90"] - lMat[, "90"])

```

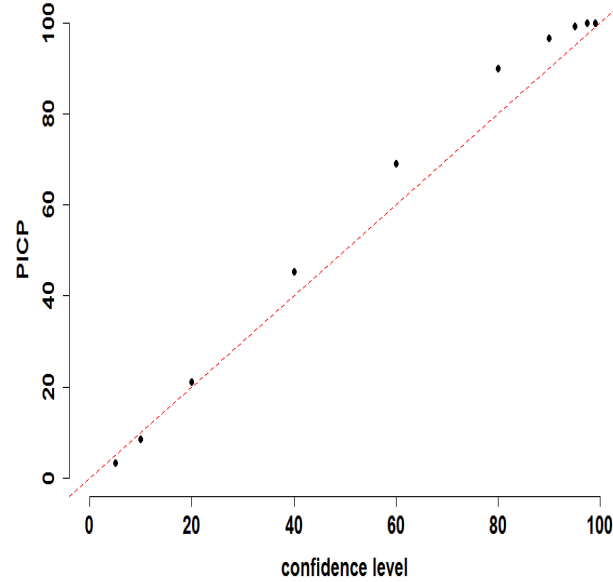


Figure 4: Plot of PICP and confidence level based on validation of bootstrapping model.

```
##          0%      25%      50%      75%      100%
## 4.551972 4.702860 4.795511 4.950161 6.610331
```

3 Empirical uncertainty quantification through data partitioning and cross validation.

The next two approaches for uncertainty quantification are empirical methods whereby the prediction intervals are defined from the distribution of model errors i.e. the deviation between observation and model predictions. In the examples to follow however, the prediction limits are not spatially uniform. Rather they are a function of the landscape. For example, in some areas of the landscape, or in some particular landscape situations, the prediction is going to be more accurate than in other situations. For the immediate approach, an explicit landscape partition is used to define likely areas where the quantification of uncertainty may differ. This approach uses the Cubist data mining algorithm for prediction and partitioning of the landscape. Within each partitioned area, we can then define the distribution of model errors. This approach for uncertainty quantification was used by Malone et al. (2014) where soil pH was predicted using a Cubist model and residual kriging spatial model.

This approach and the next are useful from the viewpoint that complex

models (in terms of parameterisation) are able to be used within a digital soil mapping framework where there is a necessity to define a stated level of prediction confidence. With universal kriging, this is the restriction of the linear model (which in many situations could be the best model). For empirical approaches to uncertainty quantification, this restriction does not exist. Further to this, the localized allocation of prediction limits i.e. in given landscape situations provides a means to identify areas where mapping is accurate and where it is not so much. Often this is a function of the sampling density. This explicit allocation of prediction intervals may assist with the future design of soil sampling programs.

In the approach to be described the partitioning is a “hard” partition, for the next approach described in the next section, it is a “soft” or “fuzzy” partition.

3.1 Defining the model parameters

We will use a Cubist model regression kriging approach as used previously, and then define the subsequent prediction intervals.

First we prepare the data.

```
# subset data for modeling
set.seed(667)
training <- sample(nrow(HV_subsoilpH), 0.7 * nrow(HV_subsoilpH))
cDat <- HV_subsoilpH[training, ]
vDat <- HV_subsoilpH[-training, ]
```

Now we fit the Cubist model using all available environmental covariates.

```
library(Cubist)

# Cubist model
hv.cub.Exp <- cubist(x = cDat[, c("Terrain_Ruggedness_Index", "AACN", "Landsat_Band1",
  "Elevation", "Hillshading", "Light_insolation", "Mid_Slope_Positon", "MRVBF",
  "NDVI", "TWI", "Slope")],
  y = cDat$pH60_100cm, cubistControl(unbiased = TRUE, rules = 100,
  extrapolation = 10, sample = 0, label = "outcome"), committees = 1)

summary(hv.cub.Exp)

##
## Call:
## cubist.default(x = cDat[, c("Terrain_Ruggedness_Index",
## = cubistControl(unbiased = TRUE, rules = 100, extrapolation = 10,
## sample = 0, label = "outcome"))
##
##
##
## Cubist [Release 2.07 GPL Edition] Tue Jan 10 12:59:14 2017
## -----
```

```

##
##      Target attribute `outcome'
##
## Read 354 cases (12 attributes) from undefined.data
##
## Model:
##
## Rule 1: [116 cases, mean 6.183380, range 3.956412 to 9.249626, est err 0.714710]
##
##      if
## NDVI <= -0.191489
## TWI <= 13.17387
##      then
## outcome = 9.610208 + 0.0548 AACN - 0.0335 Elevation + 0.131 Hillshading
##           + 3.5 NDVI + 0.076 Terrain_Ruggedness_Index - 0.055 Slope
##           - 0.023 Landsat_Band1 + 0.03 MRVBF + 0.07 Mid_Slope_Positon
##           + 0.005 TWI
##
## Rule 2: [164 cases, mean 6.533212, range 3.437355 to 9.741758, est err 0.986175]
##
##      if
## TWI > 13.17387
##      then
## outcome = 7.471082 + 0.0215 AACN + 0.108 Hillshading + 4.2 NDVI
##           + 0.24 MRVBF + 1.16 Mid_Slope_Positon - 0.0104 Elevation
##           - 0.069 Slope + 0.077 Terrain_Ruggedness_Index
##           - 0.028 Landsat_Band1 + 0.047 TWI
##
## Rule 3: [74 cases, mean 6.926269, range 2.997182 to 9.630296, est err 1.115631]
##
##      if
## NDVI > -0.191489
## TWI <= 13.17387
##      then
## outcome = 11.743466 + 0.0416 AACN - 0.091 Landsat_Band1
##           - 0.0117 Elevation + 3 NDVI + 0.048 Hillshading
##           + 0.065 Terrain_Ruggedness_Index - 0.046 Slope
##
##
## Evaluation on training data (354 cases):
##
##      Average |error|          1.085791
##      Relative |error|          0.95
##      Correlation coefficient    0.44
##
##
## Attribute usage:
##      Conds  Model
##

```

```
##      100%      79%      TWI
##      54%      100%      NDVI
##              100%      Terrain_Ruggedness_Index
##              100%      AACN
##              100%      Landsat_Band1
##              100%      Elevation
##              100%      Hillshading
##              100%      Slope
##              79%      Mid_Slope_Positon
##              79%      MRVBF
##
##
## Time: 0.0 secs
```

The `hv.cub.Exp` model indicates three subsets in the data were made. Assessing the goodness of fit of this model, we use the `goof` function. This model seems to perform OK.

```
goof(observed = cDat$pH60_100cm, predicted = predict(hv.cub.Exp, newdata = cDat))

##              R2 concordance      MSE      RMSE      bias
## 1 0.3247524 0.4779242 1.266316 1.125307 2.640776e-07
```

Now we want to assess the model residuals for spatial auto correlation. In this case we need to look at the variogram of the residuals.

```
cDat$residual <- cDat$pH60_100cm - predict(hv.cub.Exp, newdata = cDat)

# coordinates
coordinates(cDat) <- ~X + Y
# residual variogram model
vgm1 <- variogram(residual ~ 1, cDat, width = 200)
mod <- vgm(psill = var(cDat$residual), "Sph", range = 10000, nugget = 0)
model_1 <- fit.variogram(vgm1, mod)

gOK <- gstat(NULL, "hunterpH_cubistRES", residual ~ 1, cDat, model = model_1)
gOK

## data:
## hunterpH_cubistRES : formula = residual~~1 ; data dim = 354 x 13
## variograms:
##              model      psill      range
## hunterpH_cubistRES[1]  Nug 0.9986889 0.000
## hunterpH_cubistRES[2]  Sph 0.9891089 7693.044
```

This output indicates there is a reasonable variogram of the residuals with which may help improve the overall predictive model. We can determine whether there is any improvement later when we perform the validation.

With a model defined, it is now necessary to estimate the within partition prediction limits. This is done using a leave-one-out cross validation procedure using a Cubist model regression kriging model as we did for the spatial model. What we need to do first is to determine which observations in the data set

belong to which partition. Looking at the `summary` output above we can examine the threshold criteria that define the data partitions.

```
# Assign a rule to each observation if more than one observation
cDat1 <- as.data.frame(cDat)
cDat1$rule = 9999
# rule 1
cDat1$rule[which(cDat1$NDVI <= -0.191489 & cDat1$TWI <= 13.17387)] <- 1
# rule 2
cDat1$rule[which(cDat1$TWI > 13.17387)] <- 2
## rule 3
cDat1$rule[which(cDat1$NDVI > -0.191489 & cDat1$TWI <= 13.17387)] <- 3

cDat1$rule <- as.factor(cDat1$rule)
summary(cDat1$rule)

##      1      2      3
## 115 165   74
```

Now we can subset `cDat1` based on its defined rule or partition and then perform the LOCV. The script below shows the procedure for the process using the first rule.

```
# subset the data
cDat1.r1 <- cDat1[which(cDat1$rule == 1), ]
target.C.r1 <- cDat1.r1$pH60_100cm

##### Leave-one-out cross validation #####
looResiduals <- numeric(nrow(cDat1.r1))
for (i in 1:nrow(cDat1.r1)) {
  loocubistPred <- cubist(x = cDat1.r1[-i, 4:14], y = target.C.r1[-i],
    cubistControl(unbiased = F, rules = 1, extrapolation = 5,
    sample = 0, seed = sample.int(4096, size = 1) -
    1L, label = "outcome"),
    committees = 1) # fit cubist model

  # fit cubist model
  cDat11.r1.sub <- cDat1.r1[-i, ]
  cDat11.r1.sub$pred <- predict(loocubistPred, newdata = cDat11.r1.sub,
    neighbors = 0)
  cDat11.r1.sub$resids <- target.C.r1[-i] - cDat11.r1.sub$pred

  # residual variogram
  vgm.r1 <- variogram(resids ~ 1, ~X + Y, cDat11.r1.sub, width = 100)
  mod.r1 <- vgm(psill = var(cDat11.r1.sub$resids), "Sph", range = 10000, nugget = 0)
  model_1.r1 <- fit.variogram(vgm.r1, mod.r1)

  # interpolate residual on withheld point
  int.resids1.r1 <- krige(cDat11.r1.sub$resids ~ 1, locations = ~X + Y,
    data = cDat11.r1.sub, newdata = cDat1.r1[i, c("X", "Y")],
    model = model_1.r1)[, 3]
```

```

# Cubist model predict on withheld point
looPred <- predict(loocubistPred, newdata = cDat1.r1[i, ], neighbors = 0)

# Add
looResiduals[i] <- target.C.r1[i] - (looPred + int.resids1.r1)
}

```

We are interested in evaluating the 90% prediction interval. We do this for each data partition by taking the lower 5th and upper 95th quantiles of the residual distribution, and ultimately add these to the model predictions. For the validation, we also have to evaluate the quantiles for a range of confidence levels (in this case sequentially from 5% to 99%) for calculation of the PICP. This is done using the `quantile` function.

```

# Rule 1 90% confidence
r1.ulPI <- quantile(looResiduals1, probs = c(0.05, 0.95), na.rm = FALSE,
names = F, type = 7)
# rule lower and upper PI
r1.ulPI

## [1] -1.210957  1.763378

# Confidence interval range
r1.q <- quantile(looResiduals1, probs = c(0.005, 0.995, 0.0125, 0.9875, 0.025,
0.975, 0.05, 0.95, 0.1, 0.9, 0.2, 0.8, 0.3, 0.7, 0.4, 0.6, 0.45, 0.55, 0.475,
0.525), na.rm = FALSE, names = F, type = 7)

# Rule 2 90% confidence
r2.ulPI <- quantile(looResiduals2, probs = c(0.05, 0.95), na.rm = FALSE,
names = F, type = 7)
# rule lower and upper PI
r2.ulPI

## [1] -2.077091  2.314473

# Confidence interval range
r2.q <- quantile(looResiduals2, probs = c(0.005, 0.995, 0.0125, 0.9875, 0.025,
0.975, 0.05, 0.95, 0.1, 0.9, 0.2, 0.8, 0.3, 0.7, 0.4, 0.6, 0.45, 0.55, 0.475,
0.525), na.rm = FALSE, names = F, type = 7)

# Rule 3 90% confidence
r3.ulPI <- quantile(looResiduals3, probs = c(0.05, 0.95), na.rm = FALSE,
names = F, type = 7)
# rule lower and upper PI
r3.ulPI

## [1] -1.847980  1.677535

# Confidence interval range
r3.q <- quantile(looResiduals3, probs = c(0.005, 0.995, 0.0125, 0.9875, 0.025,
0.975, 0.05, 0.95, 0.1, 0.9, 0.2, 0.8, 0.3, 0.7, 0.4, 0.6, 0.45, 0.55, 0.475,
0.525), na.rm = FALSE, names = F, type = 7)

```

As can be seen, the prediction limits for each partition of the data are quite different from each other.

3.2 Spatial mapping

To create the maps in the same fashion as was done for bootstrapping and universal kriging, we do more-or-less as the same for the Cubist regression kriging. First create the regression kriging map.

```
# map
map.cubist <- predict(hunterCovariates_sub, hv.cub.Exp, args = list(neighbors = 0),
  filename = "rk_cubist.tif", format = "GTiff", overwrite = T)

# kriged residuals
map.cubist.res <- interpolate(hunterCovariates_sub, gOK, xyOnly = TRUE, index = 1,
  filename = "rk_residuals.tif", format = "GTiff",
  datatype = "FLT4S", overwrite = TRUE)

# raster stack of predictions and residuals
r2 <- stack(map.cubist, map.cubist.res)
f1 <- function(x) calc(x, sum)
map.cubist.final <- calc(r2, fun = sum, filename = "cubistRK.tif",
  format = "GTiff", overwrite = T)
```

To derive the upper and lower prediction limits we can apply the raster calculations in a very manual way, such that each line of the `rasterStack` is read in and then evaluated to determine which rule each entry on the line belongs to. Then given the rule, the corresponding upper and lower limits are appended to a new raster together with a raster which indicates which rule was applied where. For small raster data sets where the mapping extent is small, this approach works fine. It can also be applied for very large rasters, but can take some time as the reading of the raster occurs line by line. There are however, raster options to process them this way in chunk form.

```
# Create new raster datasets
upper1 <- raster(hunterCovariates_sub[[1]])
lower1 <- raster(hunterCovariates_sub[[1]])
rule1 <- raster(hunterCovariates_sub[[1]])
upper1 <- writeStart(upper1, filename = "cubRK_upper1.tif", format = "GTiff",
  overwrite = TRUE)
lower1 <- writeStart(lower1, filename = "cubRK_lower1.tif", format = "GTiff",
  overwrite = TRUE)
rule1 <- writeStart(rule1, filename = "cubRK_rule1.tif", format = "GTiff",
  datatype = "INT2S", overwrite = TRUE)

for (i in 1:dim(upper1)[1]) {
  # extract raster information line by line
  cov.Frame <- as.data.frame(getValues(hunterCovariates_sub, i))
  ulr.Frame <- matrix(NA, ncol = 3, nrow = dim(upper1)[2])
  # append in partition information
```

```

# rule 1
ulr.Frame[which(cov.Frame$NDVI <= -0.191489 & cov.Frame$TWI <= 13.17387),
  1] <- r1.ulPI[2]
ulr.Frame[which(cov.Frame$NDVI <= -0.191489 & cov.Frame$TWI <= 13.17387),
  2] <- r1.ulPI[1]
ulr.Frame[which(cov.Frame$NDVI <= -0.191489 & cov.Frame$TWI <= 13.17387),
  3] <- 1
# rule 2
ulr.Frame[which(cov.Frame$TWI > 13.17387), 1] <- r2.ulPI[2]
ulr.Frame[which(cov.Frame$TWI > 13.17387), 2] <- r2.ulPI[1]
ulr.Frame[which(cov.Frame$TWI > 13.17387), 3] <- 2
# rule 3
ulr.Frame[which(cov.Frame$NDVI > -0.191489 & cov.Frame$TWI <= 13.17387),
  1] <- r3.ulPI[2]
ulr.Frame[which(cov.Frame$NDVI > -0.191489 & cov.Frame$TWI <= 13.17387),
  2] <- r3.ulPI[1]
ulr.Frame[which(cov.Frame$NDVI > -0.191489 & cov.Frame$TWI <= 13.17387),
  3] <- 3

ulr.Frame <- as.data.frame(ulr.Frame)
names(ulr.Frame) <- c("upper", "lower", "rule")
# write to raster then close
pred_upper <- ulr.Frame$upper
pred_lower <- ulr.Frame$lower
pred_rule <- ulr.Frame$rule
upper1 <- writeValues(upper1, pred_upper, i)
lower1 <- writeValues(lower1, pred_lower, i)
rule1 <- writeValues(rule1, pred_rule, i)
print(i)
}
upper1 <- writeStop(upper1)
lower1 <- writeStop(lower1)
rule1 <- writeStop(rule1)

```

Now we can derive the prediction interval by adding the upper and lower limits to the regression kriging prediction that was made earlier. Then we can estimate the prediction interval range.

```

# raster stack of predictions and prediction limits
r2 <- stack(map.cubist.final, lower1) #lower
mapRK.lower <- calc(r2, fun = sum, filename = "cubistRK_lowerPL.tif",
format = "GTiff", overwrite = T)

# raster stack of predictions and prediction limits
r2 <- stack(map.cubist.final, upper1) #upper
mapRK.upper <- calc(r2, fun = sum, filename = "cubistRK_upperPI.tif",
format = "GTiff", overwrite = T)

```

```

# Prediction interval range
r2 <- stack(mapRK.lower, mapRK.upper) #diff
mapRK.PIrange <- calc(r2, fun = diff, filename = "cubistRK_PIrange.tif",
format = "GTiff", overwrite = T)

```

Now we can plot the maps as before. We can note this time that the prediction interval range is smaller for the cubist regression kriging than it is for the universal kriging approach (Figure 5).

```

# color ramp
phCramp <- c("#d53e4f", "#f46d43", "#fdae61", "#fee08b", "#ffffbf", "#e6f598",
"#abdda4", "#66c2a5", "#3288bd", "#5e4fa2", "#542788", "#2d004b")
brk <- c(2:14)
par(mfrow = c(2, 2))
plot(mapRK.lower, main = "90% Lower prediction limit", breaks = brk, col = phCramp)
plot(mapRK.cubist.final, main = "Prediction", breaks = brk, col = phCramp)
plot(mapRK.upper, main = "90% Upper prediction limit", breaks = brk, col = phCramp)
plot(mapRK.PIrange, main = "Prediction limit range", col = terrain.colors(length(seq(0,
6.5, by = 1)) - 1), axes = FALSE, breaks = seq(0, 6.5, by = 1))

```

3.3 Validating the quantification of uncertainty

For validation we assess both the quality of the predictions and the quantification of uncertainty as was done earlier for the universal kriging. Below we assess the validation of the cubist model alone and the cubist regression kriging model.

```

# Cubist model prediction
vPreds <- predict(hv.cub.Exp, newdata = vDat)
# Residual prediction
coordinates(vDat) <- ~X + Y
OK.preds.V <- as.data.frame(krige(residual ~ 1, cDat, model = model_1, newdata = vDat))
## [using ordinary kriging]

# Regression kriging predictions
OK.preds.V$cubist <- vPreds
OK.preds.V$finalP <- OK.preds.V$cubist + OK.preds.V$var1.pred

# Validation regression
goof(observed = vDat$pH60_100cm, predicted = OK.preds.V$cubist)

##           R2 concordance      MSE      RMSE      bias
## 1 0.2218322    0.422689 1.352345 1.162903 0.2017115

# regression kriging
goof(observed = vDat$pH60_100cm, predicted = OK.preds.V$finalP)

##           R2 concordance      MSE      RMSE      bias
## 1 0.2860766    0.5116982 1.257862 1.121544 0.1360403

```

The regression kriging model is a little more accurate than the Cubist model

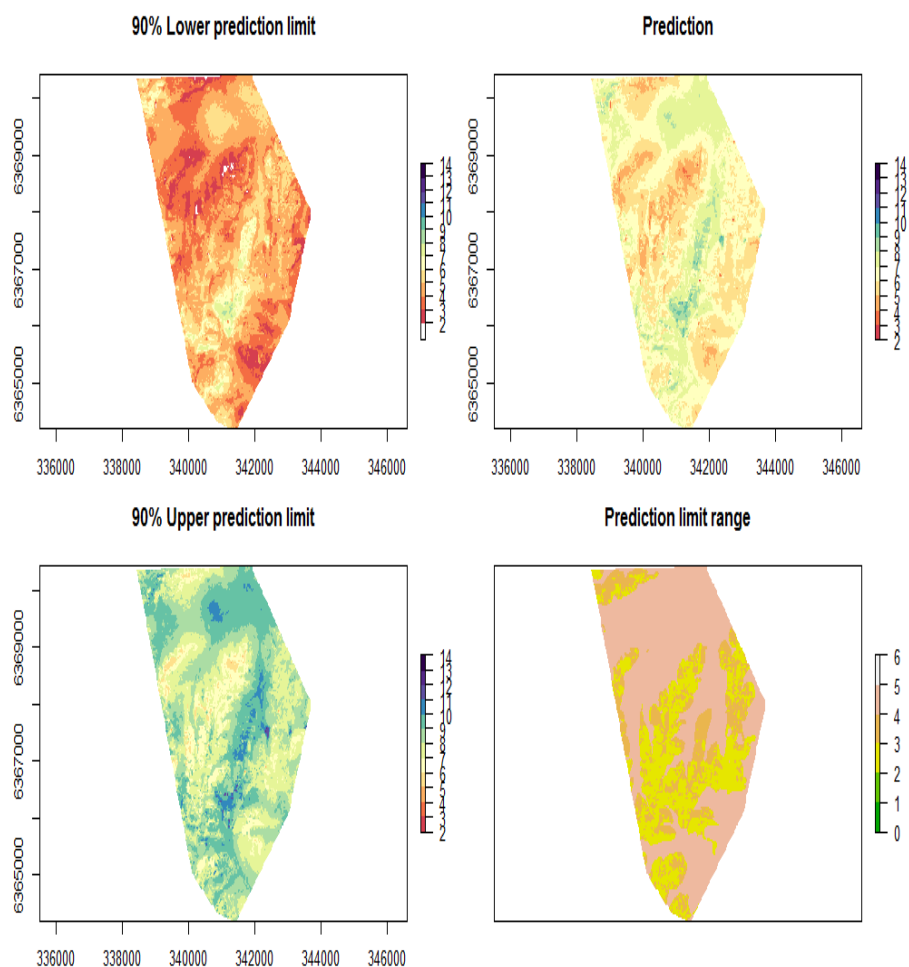


Figure 5: Soil pH predictions and prediction limits derived using a Cubist regression kriging model

alone.

The first step in estimating the uncertainty about the validation points is to evaluate with rule set or partition it belongs to.

```
# Assign a rule to each observation if more than one observation
vDat1 <- as.data.frame(vDat)
# Insert for rules bit
vDat1$rule = 9999
# rule 1
vDat1$rule[which(vDat1$NDVI <= -0.191489 & vDat1$TWI <= 13.17387)] <- 1
# rule 2
vDat1$rule[which(vDat1$TWI > 13.17387)] <- 2
## rule 3
vDat1$rule[which(vDat1$NDVI > -0.191489 & vDat1$TWI <= 13.17387)] <- 3
```

```
vDat1$rule <- as.factor(vDat1$rule)
summary(vDat1$rule)
```

```
## 1 2 3
## 43 65 44
```

```
# append regression kriging predictions
vDat1 <- cbind(vDat1, OK.preds.V[, "finalP"])
names(vDat1)[ncol(vDat1)] <- "RKpred"
```

Then we can define the prediction interval that corresponds to each observation for each level of confidence.

```
# Upper PL
ulMat <- matrix(NA, nrow = nrow(vDat1), ncol = length(r1.q))
for (i in seq(2, 20, 2)) {
  ulMat[which(vDat1$rule == 1), i] <- r1.q[i]
  ulMat[which(vDat1$rule == 2), i] <- r2.q[i]
  ulMat[which(vDat1$rule == 3), i] <- r3.q[i]
}

# Lower PL
for (i in seq(1, 20, 2)) {
  ulMat[which(vDat1$rule == 1), i] <- r1.q[i]
  ulMat[which(vDat1$rule == 2), i] <- r2.q[i]
  ulMat[which(vDat1$rule == 3), i] <- r3.q[i]
}

# upper and lower prediction limits
ULpreds <- ulMat + vDat1$RKpred

# binary
bMat <- matrix(NA, nrow = nrow(ULpreds), ncol = (ncol(ULpreds)/2))
cnt <- 1
for (i in seq(1, 20, 2)) {
  bMat[, cnt] <- as.numeric(vDat1$pH60_100cm <= ULpreds[, i + 1] & vDat1$pH60_100cm
    >= ULpreds[, i])
  cnt <- cnt + 1
}

colSums(bMat)/nrow(bMat)

## [1] 0.98684211 0.98684211 0.96710526 0.90789474 0.82236842 0.61842105
## [7] 0.34868421 0.23684211 0.08552632 0.03289474
```

The PICP estimates appear to correspond quite well with the respective confidence levels. This can be observed from the plot on Figure 6 too.

```
# make plot
cs <- c(99, 97.5, 95, 90, 80, 60, 40, 20, 10, 5) # confidence level
plot(cs, ((colSums(bMat)/nrow(bMat)) * 100))
abline(a = 0, b = 1, lty = 2, col = "red")
```

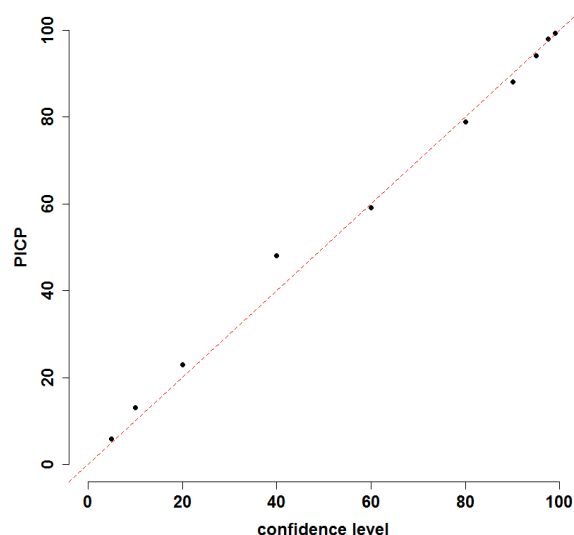


Figure 6: Plot of PICP and confidence level based on validation of Cubist regression kriging model.

From the validation observations the prediction intervals range between 2.45 and 4.6 with a median of about 3.36 pH units when using the Cubist regression kriging model.

```
quantile(ULpreds[, 8] - ULpreds[, 7])
##          0%          25%          50%          75%         100%
## 2.974336 2.974336 3.525515 4.391563 4.391563
```

4 Empirical uncertainty quantification through fuzzy clustering and cross validation

Like the previous uncertainty approach, this approach is similar in that uncertainty is expressed in the form of quantiles of the underlying distribution of model error (residuals). It contrasts however in terms of how the environmental data is partitioned. For the previous approach, partitioning was based upon the hard classification defined by a fitted Cubist model. This approach uses a fuzzy clustering method of partition.

Essentially the approach is based on previous research by Shrestha and Solomatine (2006) where the idea is to partition a feature space into clusters (with a fuzzy k-means routine) which share similar model errors. A prediction interval (PI) is constructed for each cluster on the basis of the empirical

distribution of residual observations that belong to each cluster. A PI is then formulated for each observation in the feature space according to the grade of their memberships to each cluster. They applied this methodology to artificial and real hydrological data sets and it was found to be superior to other methods which estimate a PI. The Shrestha and Solomatine (2006) approach computes the PI independently and while free of the prediction model structure, it requires only the model or prediction outputs. Tranter et al. (2010) extended this approach to deal with observations that are outside of the training domain.

The method presented in this exercise was introduced by Malone et al. (2011) which modifies slightly the Shrestha and Solomatine (2006) and Tranter et al. (2010) approaches to enable it for a DSM framework. The approach is summarized by the flow diagram on Figure 7.

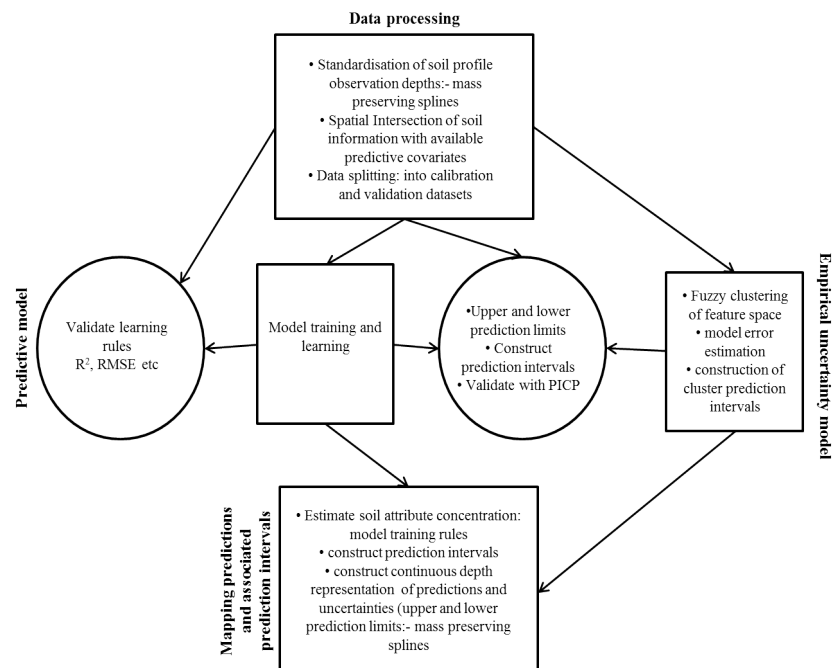


Figure 7: Flow diagram of the general procedure for achieving the outcome of mapping predictions and their uncertainties (upper and lower prediction limits) within a digital soil mapping framework. The 3 components for achieving this outcome are the prediction model, the empirical uncertainty model and the mapping component. Sourced from Malone et al. (2011).

The process for deriving the uncertainties is much the same as for the previous approach using the Cubist regression kriging approach. One benefit of using a fuzzy kmeans approach is that the spatial distribution of uncertainty is represented as a continuous variable. Further, the incorporation of extragrades in the fuzzy kmeans classifying provides an explicit means to identify and highlight areas of the greatest uncertainty and possibly where new sampling efforts should be prioritized. As shown on 7 the approach entails 3

main processes:

- Calibrating the spatial model
- deriving the uncertainty model which includes both estimations of model errors and fuzzy kmeans with extragrades classification
- Creation of maps of both spatial soil predictions and uncertainties.

Naturally, this framework is validated using a withheld or better still independent data set.

4.1 Defining the model parameters

Here we will use a random forest regression kriging model for the prediction of soil pH across the study area. This model will also be incorporated into the uncertainty model via leave-one-out cross validation in order to derive the model errors. As before, we begin by preparing the data.

```
# Point data subset data for modeling
set.seed(667)
training <- sample(nrow(HV_subsoilpH), 0.7 * nrow(HV_subsoilpH))
cDat <- HV_subsoilpH[training, ]
vDat <- HV_subsoilpH[-training, ]
```

Now we fit a random forest model using all possible covariates.

```
# fit the model
library(randomForest)

hv.RF.Exp <- randomForest(pH60_100cm ~ Terrain_Ruggedness_Index + AACN + Landsat_Band1 +
  Elevation + Hillshading + Light_insolation + Mid_Slope_Positon + MRVBF +
  NDVI + TWI + Slope, data = cDat, importance = TRUE, ntree = 1000)

goof(observed = cDat$pH60_100cm, predicted = predict(hv.RF.Exp, newdata = cDat),
  plot.it = FALSE)

##           R2 concordance           MSE           RMSE           bias
## 1 0.9311825 0.8993517 0.2718095 0.5213536 0.002527227
```

The `hv.RF.Exp` model appears to perform quite well when we examine the goodness of fit statistics.

Now we can examine the model residuals for any presence of spatial structure with variogram modeling. For the output below it does seem that there is some useful correlation structure in the residuals that will likely help to improve upon the performance of the `hv.RF.Exp` model.

```
# Estimate the residual
cDat$residual <- cDat$pH60_100cm - predict(hv.RF.Exp, newdata = cDat)
# residual variogram model
coordinates(cDat) <- ~X + Y
vgm1 <- variogram(residual ~ 1, data = cDat, width = 200)
mod <- vgm(psill = var(cDat$residual), "Sph", range = 10000, nugget = 0)
```

```

model_1 <- fit.variogram(vgm1, mod)
model_1

##      model      psill      range
## 1   Nug 0.9834345    0.000
## 2   Sph 0.1790487 6408.035

gOK <- gstat(NULL, "hunterpH_residual_RF", residual ~ 1, cDat, model = model_1)

```

Like before, we need to estimate the model errors and a good way to do this is via a LOCV approach. The script below is more-or-less a repeat from earlier with the Cubist regression kriging modeling except now we are using the random forest modeling.

```

# Uncertainty analysis
cDat1 <- as.data.frame(cDat)
names(cDat1)
cDat1.r1 <- cDat1
target.C.r1 <- cDat1.r1$pH60_100cm

looResiduals <- numeric(nrow(cDat1.r1))
for (i in 1:nrow(cDat1.r1)) {
  looRFPred <- randomForest(pH60_100cm ~ Terrain_Ruggedness_Index + AACN +
    Landsat_Band1 + Elevation + Hillshading + Light_insolation + Mid_Slope_Positon +
    MRVBF + NDVI + TWI + Slope, data = cDat1.r1[-i, ], importance = TRUE,
    ntree = 1000)

  cDat11.r1.sub <- cDat1.r1[-i, ]
  cDat11.r1.sub$pred <- predict(looRFPred, newdata = cDat11.r1.sub)
  cDat11.r1.sub$resids <- target.C.r1[-i] - cDat11.r1.sub$pred

  # residual variogram
  vgm.r1 <- variogram(resids ~ 1, ~X + Y, cDat11.r1.sub, width = 200)
  mod.r1 <- vgm(psill = var(cDat11.r1.sub$resids), "Sph", range = 10000, nugget = 0)
  model_1.r1 <- fit.variogram(vgm.r1, mod.r1)
  model_1.r1

  # interpolate residual
  int.resids1.r1 <- krige(cDat11.r1.sub$resids ~ 1, locations = ~X + Y,
    data = cDat11.r1.sub, newdata = cDat1.r1[i, c("X", "Y")],
    model = model_1.r1, debug.level = 0)[, 3]
  looPred <- predict(looRFPred, newdata = cDat1.r1[i, ])
  looResiduals[i] <- target.C.r1[i] - (looPred + int.resids1.r1)
}

# Combine residual to main data frame
cDat1 <- cbind(cDat1, looResiduals)

##      [1] "X"                                "Y"
##      [3] "pH60_100cm"                        "Terrain_Ruggedness_Index"
##      [5] "AACN"                              "Landsat_Band1"

```

## [7]	"Elevation"	"Hillshading"
## [9]	"Light_insolation"	"Mid_Slope_Positon"
## [11]	"MRVBF"	"NDVI"
## [13]	"TWI"	"Slope"
## [15]	"residual"	"looResiduals"

The defining aspect of this uncertainty approach is fuzzy clustering with extragrades. McBratney and de Gruijter (1992) recognized a limitation of the normal fuzzy clustering algorithm in that it had an inability to distinguish between observations very far from the cluster centroids and those at the centre of the centroid configuration. These observations were termed extragrades as opposed to intragrades, which are the observations that lie between the main clusters. The extragrades are considered the outliers of the data set and have a distorting influence on the configuration of the main clusters (Lagacherie et al., 1997). McBratney and de Gruijter (1992) developed an adaptation to the FKM algorithm which distinguishes observations that should belong to an extragrade cluster. The FKM with extragrades algorithm minimizes the objective function:

$$J_e(\mathbf{C}, \mathbf{M}) = \alpha \sum_{i=1}^n \sum_{j=1}^c m_{ij}^{\psi} d_{ij}^2 + (1 - \alpha) \sum_{i=1}^n m_{i*}^{\psi} \sum_{i=1}^n d_{ij}^{-2} \quad (2)$$

where $\mathbf{C}$ is the $c \times p$ matrix of cluster centers where c is the cluster and p is the number of variables. $\mathbf{M}$ is the $n \times c$ matrix of partial memberships, where n is the number of observations; $m_{ij} \in [0, 1]$ is the partial membership of the i th observation to the j th cluster. $\psi \geq 1$ is the fuzziness exponent. The square distance between the i th observation to the j th cluster is d_{ij}^2 . m_{i*}^{ψ} denotes the membership to the extragrade cluster. This function also requires the parameter alpha (α) to be defined, which is used to evaluate membership to the extragrade cluster.

A very good stand-alone software developed by Minasny (2002) called 'Fuzme' contains the FKM with extragrades method, plus other clustering methods. The software may be downloaded for free from <http://sydney.edu.au/agriculture/pal/software/fuzme.shtml>. The source script to this software has also been written to an R package of the same name. Normally, the stand-alone software would be used because it is computationally much faster. However, using the fuzme R package allows one to easily integrate the clustering procedures into a standard R workflow. For example, one of the issues of clustering procedures is the uncertainty regarding the selection of an optimal cluster number for a given data set. There are a number of ways to determine an optimal solution. Some popular approaches include to determining the cluster combination which minimizes the Fuzzy Performance Index (FPI) or Modified partition Entropy (MPE). The MPE establishes the degree of fuzziness created by a specified number of clusters for a defined exponent value. The notion is that the smaller the MPE, the more suitable is the corresponding number of clusters at the given exponent value. A more sophisticated analysis is to look at the derivative of $J_e(\mathbf{C}, \mathbf{M})$ with respect to ψ and is used to simultaneously establish the optimal ψ and cluster

size. More is discussed about each of these indices by Odeh et al. (1992) and Bragato (2004).

The key generally is to define clusters which can be easily distinguished compared to a collection of clusters that are all quite similar. Such diagnostic criteria are useful; however it would be more beneficial to determine an optimal cluster configuration based on criteria that are meaningful to the current situation of deriving prediction uncertainties. In this case we might want to evaluate the optimal cluster number *and* fuzzy exponent that maximizes the fidelity of PICP to confidence interval, and minimizes the prediction interval range. Hence in this section we will integrate the R `fuzme` with the DSM uncertainties workflow to achieve those ends.

The idea here is to determine an optimal cluster number and fuzzy exponent based on criteria related to prediction interval width and PICP, together with some of the other fuzzy clustering criteria. We essentially want to perform fuzzy clustering with extragrades upon the environmental covariate of the data points in the `cDat1` object. Fuzzy clustering with extragrades is implemented via a two-step procedure using the `fobjk` and `fkme` functions from the `fuzme` package. The `fobjk` function allows one to find an optimal solution for alpha (α) which is the parameter that allows one to control the membership of data to the extragrade cluster. For example, if we wanted to stipulate that we want the average extragrade membership of a data set to be 10%, then we need to find an optimal solution for alpha to achieve that outcome. This can be controlled through the `Ureq` parameter of the `fobjk` function.

The first step is to load the `fuzme` library and prepare the input data.

```
library(fuzme)
```

Now we need to prepare the data for clustering, and parameterize the other inputs of the function. The other inputs are: `nclass`, which is the number of clusters you want to define. Note that an extra cluster will be defined as this will be the extragrade cluster and the associated memberships. `data` is the data needed for clustering, `U` is an initial membership matrix in order to get the fuzzy clustering algorithm operable. `phi` is the fuzzy exponent, while `distype` refers to the distance metric to be used for clustering. There are 3 possible distance metrics available. These are: Euclidean (1), Diagonal (2), and Mahalanobis (3). As an example of using the `fobjk` lets define 4 clusters with a fuzzy exponent of 1.2, and with the average extragrade membership of 10%. Currently this function is pretty slow to compute the optimal alpha, so be prepared to wait a while.

```
# Parameterize fuzzy objective function
data.t <- cDat1[, 4:14] # data to be clustered
nclass.t <- 4 # number of clusters
phi.t <- 1.2
distype.t <- 3 #3 = Mahalanobis distance
Ureq.t <- 0.1 #average extragrade membership

# initial membership matrix
scatter.t <- 0.2 # scatter around initial membership
```

```

ndata.t <- dim(data.t)[1]
U.t <- initmember(scatter = scatter.t, nclass = nclass.t, ndata = ndata.t)

# run fuzzy objective function
alfa.t <- fobjk(Ureq = Ureq.t, nclass = nclass.t, data = data.t, U = U.t,
               phi = phi.t, distype = distype.t)

```

Remember the `fobjk` function will only return the optimal `alfa` value. This value then gets inserted into the associated `fkme` function in order to estimate the memberships of the data to each cluster and the extragrade cluster. The `fkme` function also returns the cluster centroids too.

```

alfa.t

##           1
## 0.01136809

```

The `fkme` function is parameterized similarly to `fobjk`, yet with some additional inputs related to the number of allowable iterations for convergence (`maxiter`), the convergence criterion value (`toldif`), and whether or not to display behind-the-scenes processing.

```

tester <- fkme(nclass = nclass.t, data = data.t, U = U.t, phi = phi.t,
              alfa = alfa.t, maxiter = 500, distype = distype.t, toldif = 0.01,
              verbose = 1)

```

The `fkme` function returns a list with a number of elements. At this stage we are primarily interested in the elements `membership` and `centroid` which we will use a little later on.

As described earlier, there are a number of criteria to assess the validity of a particular clustering configuration. We can evaluate these by using the `fvalidity` function. It essentially takes in a few of the outputs from the `fkme` function.

```

fvalidity(U = tester$membership[, 1:4], dist = tester$distance,
          centroid = tester$centroid, nclass = 4, phi = 1.2,
          W = tester$distNormMat)

##           fpi           mpe           S   dJ/dphi
## 1 0.5149733 0.4026764 0.8377058 -1710.19

```

Another useful metric is the confusion index (after Burrough et al. (1997)) which in our case looks at the similarity between the highest and second highest cluster memberships. The confusion index is estimated for each data point. Taking the average over the data set provides some sense of whether cluster can be distinguished from each other.

```

mean(confusion(nclass = 5, U = tester$membership))

## [1] 0.3356157

# Note number of cluster is set to 5 to account for the Additional
# extragrade cluster.

```

To assess the clustering performance using the criteria of the PICP and prediction interval range, we need to first assign each data point a one of the clusters we have derived. The assignment is based on the cluster which has the highest membership grade. The script below provides a method for evaluating which data point belongs to which cluster

```
membs <- as.data.frame(tester$membership)
membs$class <- 99999
for (i in 1:nrow(membs)) {
  mbs2 <- membs[i, 1:ncol(tester$membership)]
  # which is the maximum probability on this row
  membs$class[i] <- which(mbs2 == max(mbs2))[1]
}
membs$class <- as.factor(membs$class)
summary(membs$class)

## 1 2 3 4 5
## 58 84 78 90 44
```

Then we combine it to the `cDat1` object which contains the information regarding the model errors (specifically in the `looResiduals` column).

```
# combine
cDat1 <- cbind(cDat1, membs$class)
names(cDat1)[ncol(cDat1)] <- "class"
levels(cDat1$class)

## [1] "1" "2" "3" "4" "5"
```

Then we derive the cluster model error. This entails splitting the `cDat1` object up on the basis of the cluster with the highest membership i.e. `cDat1$class`.

```
cDat2 <- split(cDat1, cDat1$class)

# cluster lower prediction limits
quanMat1 <- matrix(NA, ncol = 10, nrow = length(cDat2))
for (i in 1:length(cDat2)) {
  quanMat1[i, ] <- quantile(cDat2[[i]][, "looResiduals"], probs = c(0.005,
    0.0125, 0.025, 0.05, 0.1, 0.2, 0.3, 0.4, 0.45, 0.475), na.rm = FALSE,
    names = F, type = 7)
}
row.names(quanMat1) <- levels(cDat1$class)
quanMat1[nrow(quanMat1), ] <- quanMat1[nrow(quanMat1), ] * 2
quanMat1 <- t(quanMat1)
row.names(quanMat1) <- c(99, 97.5, 95, 90, 80, 60, 40, 20, 10, 5) #

# cluster upper prediction limits
quanMat2 <- matrix(NA, ncol = 10, nrow = length(cDat2))
for (i in 1:length(cDat2)) {
  quanMat2[i, ] <- quantile(cDat2[[i]][, "looResiduals"], probs = c(0.995,
    0.9875, 0.975, 0.95, 0.9, 0.8, 0.7, 0.6, 0.55, 0.525), na.rm = FALSE,
    names = F, type = 7)
```

```

}
quanMat2[quanMat2 < 0] <- 0
row.names(quanMat2) <- levels(cDat1$class)
quanMat2[nrow(quanMat2), ] <- quanMat2[nrow(quanMat2), ] * 2
quanMat2 <- t(quanMat2)
row.names(quanMat2) <- c(99, 97.5, 95, 90, 80, 60, 40, 20, 10, 5) #

```

The objects `quanMat1` and `quanMat2` represent the lower and upper model errors for each cluster for each quantile respectively. For the extragrade cluster, we multiple the error by a constant, here 2, in order to explicitly indicate that the extragrade cluster (being outliers of the data) have a higher uncertainty.

Using the validation or independent data that has been withheld, we evaluate the PICP and prediction interval width. This requires first allocating cluster memberships to the points on the basis of outputs from using the `fkme` function, then using these together with the cluster prediction limits to evaluate weighted averages of the prediction limits for each point. With that done we can then derive the unique upper and lower prediction interval limits for each point at each confidence level. First, for the membership allocation, we use the `fuzExall` function. Essentially this function takes in outputs from the `fkme` function and in our case, specifically that concerning the `tester` object. Recall that the validation data is saved to the `vDat` object.

```

vDat1 <- as.data.frame(vDat)
names(vDat1)

## [1] "X" "Y"
## [3] "pH60_100cm" "Terrain_Ruggedness_Index"
## [5] "AACN" "Landsat_Band1"
## [7] "Elevation" "Hillshading"
## [9] "Light_insolation" "Mid_Slope_Positon"
## [11] "MRVBF" "NDVI"
## [13] "TWI" "Slope"

# covariates of the validation data
vCovs <- vDat1[, c("Terrain_Ruggedness_Index", "AACN", "Landsat_Band1", "Elevation",
  "Hillshading", "Light_insolation", "Mid_Slope_Positon", "MRVBF", "NDVI",
  "TWI", "Slope")]

# run fkme allocation function
fuzAll <- fuzExall(data = vCovs, phi = 1.2, centroid = tester$centroid, distype = 3,
  W = tester$distNormMat, alfa = tester$alfa)

# Get the memberships
fuz.me <- fuzAll$membership

```

A "fuzzy committee" approach is used to estimate the underlying residual at each point. In this case the upper and lower limits are derived by weighted mean of the lower and upper model errors of each cluster, where the weights are the cluster memberships. This can be defined mathematically as:

$$PI_i^L = \sum_{j=1}^c m_{ij} PIC_j^L \text{ and } PI_i^U = \sum_{j=1}^c m_{ij} PIC_j^U \quad (3)$$

where PI_i^L and PI_i^U correspond to the weighted lower and upper limits for the i th observation. PIC_j^L and PIC_j^U are the lower and upper limits for each cluster j , and m_{ij} is the membership grade of the i th observation to cluster j (which were derived in the previous step). In R, this can be interpreted as:

```
# lower prediction limit
lPI.mat <- matrix(NA, nrow = nrow(fuz.me), ncol = 10)
for (i in 1:nrow(lPI.mat)) {
  for (j in 1:nrow(quanMat1)) {
    lPI.mat[i, j] <- sum(fuz.me[i, 1:ncol(fuz.me)] * quanMat1[j, ])
  }
}

# upper prediction limit
uPI.mat <- matrix(NA, nrow = nrow(fuz.me), ncol = 10)
for (i in 1:nrow(uPI.mat)) {
  for (j in 1:nrow(quanMat2)) {
    uPI.mat[i, j] <- sum(fuz.me[i, 1:ncol(fuz.me)] * quanMat2[j, ])
  }
}
```

Then we want to add these values to the actual regression kriging predictions.

```
# Regression kriging predictions
vPreds <- predict(hv.RF.Exp, newdata = vDat)
coordinates(vDat) <- ~X + Y
OK.preds.V <- as.data.frame(krige(residual ~ 1, cDat, model = model_1, newdata = vDat))

## [using ordinary kriging]
OK.preds.V$randomForest <- vPreds
OK.preds.V$finalP <- OK.preds.V$randomForest + OK.preds.V$var1.pred

# Add prediction limits to regression kriging predictions
vDat1 <- cbind(vDat1, OK.preds.V$finalP)
names(vDat1)[ncol(vDat1)] <- "RF_rkFin"

# Derive validation lower prediction limits
lPL.mat <- matrix(NA, nrow = nrow(fuz.me), ncol = 10)
for (i in 1:ncol(lPL.mat)) {
  lPL.mat[, i] <- vDat1$RF_rkFin + lPI.mat[, i]
}

# Derive validation upper prediction limits
uPL.mat <- matrix(NA, nrow = nrow(fuz.me), ncol = 10)
for (i in 1:ncol(uPL.mat)) {
```

```

    uPL.mat[, i] <- vDat1$RF_rkFin + uPI.mat[, i]
  }

```

Now as in the previous uncertainty approaches we estimate the PICP for each level of confidence. We can also estimate the average prediction interval length too. We will do this for the 90% confidence level.

```

bMat <- matrix(NA, nrow = nrow(fuz.me), ncol = 10)
for (i in 1:10) {
  bMat[, i] <- as.numeric(vDat1$pH60_100cm <= uPL.mat[, i] & vDat1$pH60_100cm >=
    lPL.mat[, i])
}

# PICP
colSums(bMat)/nrow(bMat)

## [1] 0.9605263 0.9407895 0.9276316 0.8618421 0.7763158 0.5723684 0.3552632
## [8] 0.2302632 0.1447368 0.1315789

# prediction interval width (averaged)
as.matrix(mean(uPL.mat[, 4] - lPL.mat[, 4]))

## [1]
## [1,] 3.913579

```

Recall that our motivation at the moment to derive an optimal cluster number and fuzzy exponent based on criteria of the PICP and prediction interval width. Above were the steps for evaluating those values for one clustering parameter configuration i.e. 4 clusters (plus and extragrade) with a fuzzy exponent of 1.2. Essentially we need to run sequentially, different combinations of cluster number and fuzzy exponent value and then assess to criteria resulting from each of the different configurations in order to find the optimum. For example we might initiate the process by specifying:

```

nclass.t <- seq(2, 6, 1)
nclass.t

## [1] 2 3 4 5 6

phi.t <- seq(1.2, 1.6, 0.1)
phi.t

## [1] 1.2 1.3 1.4 1.5 1.6

```

Then it is a matter of using all possible combinations of these values in the `fobjk` and `fkme` functions that are needed ultimately to estimate the PICP and prediction interval width as was just done above for a single combination. Computationally this can be pretty complex to organize. However, for brevity, below are the clustering outputs from implementing the described procedure for finding the optimal clustering parameter configuration.

```

fkme.outs

##   class expon      alfa confusion      fpi      mpe      S
## 1      2    1.2 0.005124426 0.3826885 0.5664630 0.6686026 2.434422e+00

```

```

## 2      2      1.3 0.005379610 0.5546698 0.7839436 0.8995941 3.054369e+00
## 3      2      1.4 0.005247453 0.6992748 0.9354939 1.0541716 4.641781e+00
## 4      2      1.5 0.004997139 0.8418018 1.0441222 1.1612614 1.396096e+01
## 5      2      1.6 0.004700295 0.9707954 1.0915822 1.2156737 1.466621e+06
## 6      3      1.2 0.008058919 0.3724314 0.4757238 0.5303243 1.981344e+00
## 7      3      1.3 0.008700468 0.5496682 0.6911314 0.7563128 2.668302e+00
## 8      3      1.4 0.009085534 0.7501051 0.8629492 0.9272163 1.732949e+01
## 9      3      1.5 0.009135423 0.8499948 0.9400907 1.0114920 3.759586e+09
## 10     3      1.6 0.008965663 0.9622403 1.0269282 1.0983380 1.931253e+06
## 11     4      1.2 0.011368089 0.3356154 0.4098895 0.4404190 1.466233e+00
## 12     4      1.3 0.012443913 0.5194940 0.6338683 0.6721924 1.612367e+00
## 13     4      1.4 0.013312052 0.7052652 0.8180113 0.8575417 3.673102e+00
## 14     4      1.5 0.013979655 0.8456542 0.9112886 0.9580570 3.265964e+08
## 15     4      1.6 0.014225644 0.9552375 1.0052042 1.0566683 1.238797e+07
## 16     5      1.2 0.014470766 0.3284219 0.3786159 0.3889102 1.346230e+00
## 17     5      1.3 0.016338497 0.5085432 0.6032573 0.6227569 1.345716e+00
## 18     5      1.4 0.018119534 0.7390841 0.8163015 0.8378098 3.718401e+04
## 19     5      1.5 0.019640328 0.8469274 0.9033973 0.9361176 3.123211e+07
## 20     5      1.6 0.020083903 0.9506230 0.9945531 1.0345826 1.146549e+07
## 21     6      1.2 0.018138555 0.3353890 0.3670481 0.3605114 1.231055e+00
## 22     6      1.3 0.020091380 0.4990245 0.5730767 0.5777931 1.222209e+00
## 23     6      1.4 0.023620259 0.7438648 0.8203979 0.8315384 2.894973e+05
## 24     6      1.5 0.025368293 0.8495561 0.9027086 0.9267227 1.174473e+11
## 25     6      1.6 0.026754212 0.9462337 0.9880089 1.0205813 1.641539e+09
##      dJdphi      PICIP      PIw
## 1 -1156.906 0.12973684 3.975894
## 2 -1434.856 0.15605263 3.992315
## 3 -1558.146 0.14421053 3.885237
## 4 -1594.820 0.17052632 3.883247
## 5 -1528.474 0.15736842 3.857267
## 6 -1361.192 0.11105263 3.861995
## 7 -1734.959 0.09789474 3.968063
## 8 -1922.302 0.14289474 3.976594
## 9 -1857.203 0.13368421 3.895518
## 10 -1803.114 0.33631579 4.483795
## 11 -1356.265 0.15184211 3.913579
## 12 -1810.981 0.17026316 3.995139
## 13 -2039.601 0.12947368 4.019338
## 14 -1973.390 0.14684211 4.197816
## 15 -1882.285 0.20473684 4.257101
## 16 -1351.156 0.21631579 3.903318
## 17 -1849.211 0.20052632 3.850294
## 18 -2161.102 0.15315789 3.833518
## 19 -2038.082 0.11000000 4.185526
## 20 -1897.424 0.29026316 4.499928
## 21 -1374.898 0.24263158 3.818262
## 22 -1833.395 0.20315789 3.848154
## 23 -2250.853 0.20052632 3.653220
## 24 -2075.234 0.13631579 4.155074

```

```
## 25 -1886.003 0.34947368 4.619364
```

The data frame above lists the optimal α , clustering validity diagnostics, PICP and prediction interval width diagnostics for each combination. The PICP column is actually an absolute distance of the PICP at each level of prescribed confidence. Subsequently we should look for a minimum in that regard. Overall the best combination considering PICP and PIw together is 3 clusters with a fuzzy exponent of either 1.2 or 1.3. Based on the other fuzzy validity criteria a fuzzy exponent of 1.2 is optimal. Now we just re-run the function with these optimal values in order to derive the cluster centroids which are need in order to create maps of the prediction interval and range.

```
U.t <- initmember(scatter = 0.2, nclass = 3, ndata = ndata.t)
fkme.fin <- fkme(nclass = 3, data = data.t, U = U.t, phi = 1.2, alfa = 0.008058919,
  maxiter = 500, distype = 3, toldif = 0.01, verbose = 1)
```

Now we have to calculate the cluster model error limits. This is achieved by evaluating which cluster each data point in `cDat1` belongs to based on the maximum membership. Then we derive the quantiles of the model errors for each cluster.

```
# Assign cluster to respective data point
membs <- as.data.frame(fkme.fin$membership)
membs$class <- 99999
for (i in 1:nrow(membs)) {
  mbs2 <- membs[i, 1:ncol(fkme.fin$membership)]
  # which is the maximum probability on this row
  membs$class[i] <- which(mbs2 == max(mbs2))[1]
}
membs$class <- as.factor(membs$class)
summary(membs$class)

##      1      2      3      4
## 119 104   91   40

# combine to main data frame
cDat1 <- cbind(cDat1, membs$class)
names(cDat1)[ncol(cDat1)] <- "class"
levels(cDat1$class)

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

# split data frame based on class
cDat2 <- split(cDat1, cDat1$class)

# cluster lower prediction limits
quanMat1 <- matrix(NA, ncol = 10, nrow = length(cDat2))
for (i in 1:length(cDat2)) {
  quanMat1[i, ] <- quantile(cDat2[[i]][, "looResiduals"], probs = c(0.005,
    0.0125, 0.025, 0.05, 0.1, 0.2, 0.3, 0.4, 0.45, 0.475), na.rm = FALSE,
    names = F, type = 7)
}
```

```

row.names(quanMat1) <- levels(cDat1$class)
quanMat1[nrow(quanMat1), ] <- quanMat1[nrow(quanMat1), ] * 2
quanMat1 <- t(quanMat1)
row.names(quanMat1) <- c(99, 97.5, 95, 90, 80, 60, 40, 20, 10, 5) #

# cluster upper prediction limits
quanMat2 <- matrix(NA, ncol = 10, nrow = length(cDat2))
for (i in 1:length(cDat2)) {
  quanMat2[i, ] <- quantile(cDat2[[i]][, "looResiduals"], probs = c(0.995,
    0.9875, 0.975, 0.95, 0.9, 0.8, 0.7, 0.6, 0.55, 0.525), na.rm = FALSE,
    names = F, type = 7)
}
quanMat2[quanMat2 < 0] <- 0
row.names(quanMat2) <- levels(cDat1$class)
quanMat2[nrow(quanMat2), ] <- quanMat2[nrow(quanMat2), ] * 2
quanMat2 <- t(quanMat2)
row.names(quanMat2) <- c(99, 97.5, 95, 90, 80, 60, 40, 20, 10, 5)

```

4.2 Spatial mapping

With the spatial model defined together with the fuzzy clustering with the associated parameters and model errors, we can create the associated maps. First, the random forest regression kriging map. The map is shown on Figure 9.

```

map.Rf <- predict(hunterCovariates_sub, hv.RF.Exp, filename = "RF_HV.tif",
  format = "GTiff", overwrite = T)

# kriged residuals
crs(hunterCovariates_sub) <- NULL
map.KR <- interpolate(hunterCovariates_sub, gOK, xyOnly = TRUE, index = 1,
  filename = "krigedResid_RF.tif", format = "GTiff",
  datatype = "FLT4S", overwrite = TRUE)

# raster stack of predictions and residuals
r2 <- stack(map.Rf, map.KR)
f1 <- function(x) calc(x, sum)

# add both maps
mapRF.fin <- calc(r2, fun = sum, filename = "RF_RF.tif", format = "GTiff",
  overwrite = T)

```

Now we need to map the prediction intervals. Essentially for every pixel on the map we first need to estimate the membership value to each cluster. This membership is based on a distance of the covariate space and the centroids of each cluster. To do this we use the fuzzy allocation function (`fuzExall`) that was used earlier. This time we use the fuzzy parameters from the `fkme_final` object. We need to firstly create a `dataframe` from the `rasterStack` of covariates.

```

# Prediction Intervals
hunterCovs.df <- data.frame(cellNos = seq(1:ncell(hunterCovariates_sub)))
vals <- as.data.frame(getValues(hunterCovariates_sub))
hunterCovs.df <- cbind(hunterCovs.df, vals)
hunterCovs.df <- hunterCovs.df[complete.cases(hunterCovs.df), ]
cellNos <- c(hunterCovs.df$cellNos)
gXY <- data.frame(xyFromCell(hunterCovariates_sub, cellNos, spatial = FALSE))
hunterCovs.df <- cbind(gXY, hunterCovs.df)
str(hunterCovs.df)

## 'data.frame': 33252 obs. of 14 variables:
## $ x : num 340935 340960 340985 341010 341035 ...
## $ y : num 6370416 6370416 6370416 6370416 6370416 ...
## $ cellNos : int 101 102 103 104 105 106 107 108 109 110 ...
## $ Terrain_Ruggedness_Index: num 0.745 0.632 0.535 0.472 0.486 ...
## $ AACN : num 9.78 9.86 10.04 10.27 10.53 ...
## $ Landsat_Band1 : num 68 63 59 62 56 54 59 62 54 56 ...
## $ Elevation : num 103 103 102 102 102 ...
## $ Hillshading : num 0.94 0.572 0.491 0.515 0.568 ...
## $ Light_insolation : num 1712 1706 1701 1699 1697 ...
## $ Mid_Slope_Positon : num 0.389 0.387 0.386 0.386 0.386 ...
## $ MRVBF : num 0.376 0.765 1.092 1.54 1.625 ...
## $ NDVI : num -0.178 -0.18 -0.164 -0.169 -0.172 ...
## $ TWI : num 16.9 17.2 17.2 17.2 17.2 ...
## $ Slope : num 0.968 0.588 0.503 0.527 0.581 ...

```

Now we prepare all the other inputs for the `fuzExall` function, and then run it. This may take a little time.

```

# run fuzme allocation function
fuz.me_ALL <- fuzExall(data = hunterCovs.df[, 4:ncol(hunterCovs.df)],
centroid = fkme.fin$centroid, phi = 1.2, distype = 3, W = fkme.fin$distNormMat,
alfa = fkme.fin$alfa)
head(fuz.me_ALL$membership)

##           [,1]      [,2]      [,3]      [,4]
## [1,] 0.4198477 0.03991937 0.04619495 0.4940379882
## [2,] 0.8235166 0.04977852 0.07423964 0.0524652482
## [3,] 0.8701595 0.04153527 0.08148581 0.0068194501
## [4,] 0.8533413 0.03511594 0.10930323 0.0022395558
## [5,] 0.8360081 0.03625381 0.12709744 0.0006406516
## [6,] 0.8362873 0.03617278 0.12718138 0.0003585568

```

With the memberships estimated, lets visualize them by creating the associated membership maps (Figure 8).

```

# combine
hvCovs <- cbind(hunterCovs.df[, 1:2], fuz.me_ALL)
# Create raster
map.class1mem <- rasterFromXYZ(hvCovs[, c(1, 2, 3)])
names(map.class1mem) <- "class_1"

```

```

map.class2mem <- rasterFromXYZ(hvCovs[, c(1, 2, 4)])
names(map.class2mem) <- "class_2"
map.class3mem <- rasterFromXYZ(hvCovs[, c(1, 2, 5)])
names(map.class3mem) <- "class_3"
map.classExtramem <- rasterFromXYZ(hvCovs[, c(1, 2, 6)])
names(map.classExtramem) <- "class_ext"

par(mfrow = c(2, 2))
plot(map.class1mem, main = "cluster 1", col = terrain.colors(length(seq(0, 1,
  by = 0.2)) - 1), axes = FALSE, breaks = seq(0, 1, by = 0.2))
plot(map.class2mem, main = "cluster 2", col = terrain.colors(length(seq(0, 1,
  by = 0.2)) - 1), axes = FALSE, breaks = seq(0, 1, by = 0.2))
plot(map.class3mem, main = "cluster 3", col = terrain.colors(length(seq(0, 1,
  by = 0.2)) - 1), axes = FALSE, breaks = seq(0, 1, by = 0.2))
plot(map.classExtramem, main = "Extragrade", col = terrain.colors(length(seq(0,
  1, by = 0.2)) - 1), axes = FALSE, breaks = seq(0, 1, by = 0.2))

```

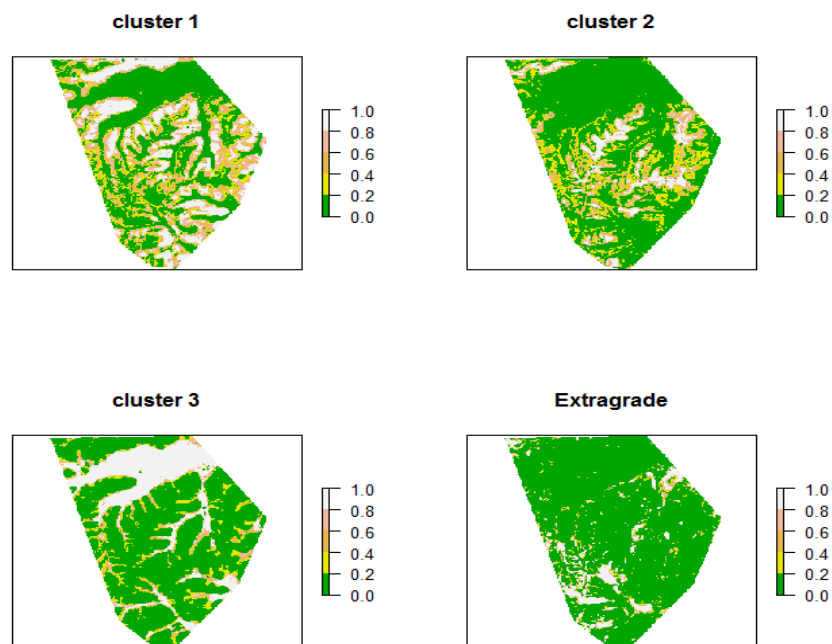


Figure 8: Fuzzy cluster memberships.

The last spatial mapping task is to evaluate the 90% prediction intervals. Again we use the fuzzy committee approach given the cluster memberships and the cluster model error limits.

```

# Lower limits
quanMat1["90", ]

```

```
##          1          2          3          4
## -1.532867 -1.451595 -1.792331 -3.140459
```

```
# upper limits
quanMat2["90", ]
```

```
##          1          2          3          4
## 2.023297 1.734574 2.182650 3.275685
```

Now we perform the weighted averaging prediction.

```
# Raster stack
s2 <- stack(map.class1mem, map.class2mem, map.class3mem, map.classExtramem)

# lower limit
f1 <- function(x) ((x[1] * quanMat1["90", 1]) + (x[2] * quanMat1["90", 2]) +
  (x[3] * quanMat1["90", 3]) + (x[4] * quanMat1["90", 4]))
mapRK.lower <- calc(s2, fun = f1, filename = "RF_lowerLimit.tif",
  format = "GTiff", overwrite = T)

# upper limit
f1 <- function(x) ((x[1] * quanMat2["90", 1]) + (x[2] * quanMat2["90", 2]) +
  (x[3] * quanMat2["90", 3]) + (x[4] * quanMat2["90", 4]))
mapRK.upper <- calc(s2, fun = f1, filename = "RF_upperLimit.tif",
  format = "GTiff", overwrite = T)
```

And finally we can derive the upper and lower prediction limits.

```
# raster stack
s3 <- stack(mapRF.fin, mapRK.lower, mapRK.upper)

# Lower prediction limit
f1 <- function(x) (x[1] + x[2])
mapRF.lowerPI <- calc(s3, fun = f1, filename = "RF_lowerPL.tif", format = "GTiff",
  overwrite = T)

# Upper prediction limit
f1 <- function(x) (x[1] + x[3])
mapRF.upperPI <- calc(s3, fun = f1, filename = "RF_upperPL.tif", format = "GTiff",
  overwrite = T)

# Prediction interval range
r2 <- stack(mapRF.lowerPI, mapRF.upperPI)
mapRF.PIrange <- calc(r2, fun = diff, filename = "cubistRK_PIrange.tif",
  format = "GTiff", overwrite = T)
```

And now we can display the necessary maps (Figure 9).

```
# color ramp
phCramp <- c("#d53e4f", "#f46d43", "#fdae61", "#fee08b", "#ffffbf", "#e6f598",
  "#abdda4", "#66c2a5", "#3288bd", "#5e4fa2", "#542788", "#2d004b")
brk <- c(2:14)
par(mfrow = c(2, 2))
```

```

plot(mapRF.lowerPI, main = "90% Lower prediction limit", breaks = brk, col = phCramp)
plot(mapRF.fin, main = "Prediction", breaks = brk, col = phCramp)
plot(mapRF.upperPI, main = "90% Upper prediction limit", breaks = brk, col = phCramp)
plot(mapRF.Pirange, main = "Prediction limit range", col = terrain.colors(length(seq(0,
6.5, by = 1)) - 1), axes = FALSE, breaks = seq(0, 6.5, by = 1))

```

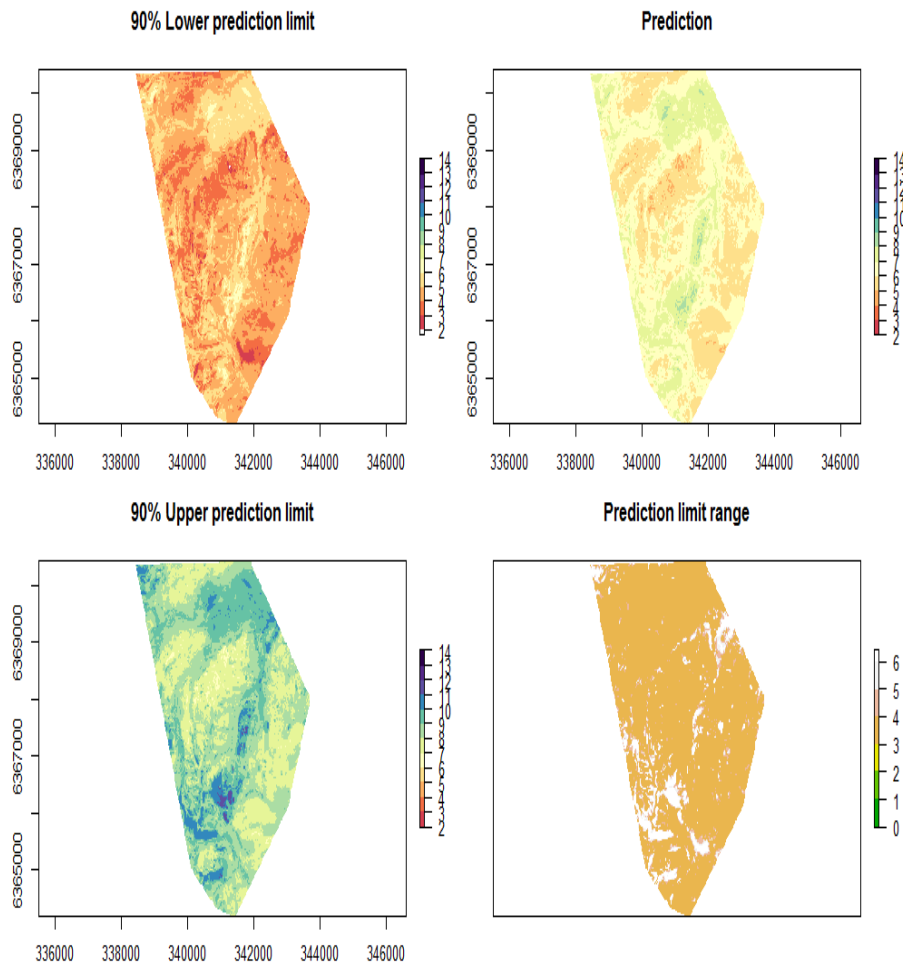


Figure 9: Soil pH predictions and prediction limits derived using a Random Forest regression kriging prediction model together with LOCV and fuzzy classification.

4.3 Validating the quantification of uncertainty

For the first step we can validate the random forest model alone and with the auto-correlated errors. As we had already applied the model earlier, it is just a matter of using the `goof` function to return the validation diagnostics.

```

# regression kriging
goof(observed = vDat$pH60_100cm, predicted = OK.preds.V$finalP, plot.it = FALSE)

##           R2 concordance      MSE      RMSE      bias
## 1 0.1454277  0.3125229 1.430826 1.196171 0.113588

# Random Forest
goof(observed = vDat$pH60_100cm, predicted = OK.preds.V$randomForest, plot.it = FALSE)

##           R2 concordance      MSE      RMSE      bias
## 1 0.1055376  0.2592972 1.512683 1.229912 0.1306959

```

The regression kriging model performs better than the random forest model alone, but only marginally so; though both models are not particularly accurate in any case.

And now to validate the quantification of uncertainty we implement the workflow demonstrated above for the process of determining the optimal cluster parameter settings.

```

## [1] "X"                                "Y"
## [3] "pH60_100cm"                        "Terrain_Ruggedness_Index"
## [5] "AACN"                              "Landsat_Band1"
## [7] "Elevation"                         "Hillshading"
## [9] "Light_insolation"                  "Mid_Slope_Positon"
## [11] "MRVBF"                             "NDVI"
## [13] "TWI"                               "Slope"
## [using ordinary kriging]

```

Now lets evaluate the PICP for each level of confidence

```

bMat <- matrix(NA, nrow = nrow(fuz.me), ncol = 10)
for (i in 1:10) {
  bMat[, i] <- as.numeric(vDat1$pH60_100cm <= uPL.mat[, i] & vDat1$pH60_100cm >=
    lPL.mat[, i])
}

# PICP
colSums(bMat)/nrow(bMat)

## [1] 0.9868421 0.9539474 0.9210526 0.8618421 0.7763158 0.5723684 0.3618421
## [8] 0.2236842 0.1250000 0.1184211

```

Plotting the PICP against the confidence level provides a nice visual. It can be seen on Figure 10 that the PICP follows closely to the 1:1 line.

```

cs <- c(99, 97.5, 95, 90, 80, 60, 40, 20, 10, 5) # confidence level
plot(cs, ((colSums(bMat)/nrow(bMat)) * 100))
abline(a = 0, b = 1, lty = 2, col = "red")

```

From the validation observations the prediction intervals range between 3.2 and 6.4 with a median of about 3.6 pH units when using the Random Forest regression kriging model.

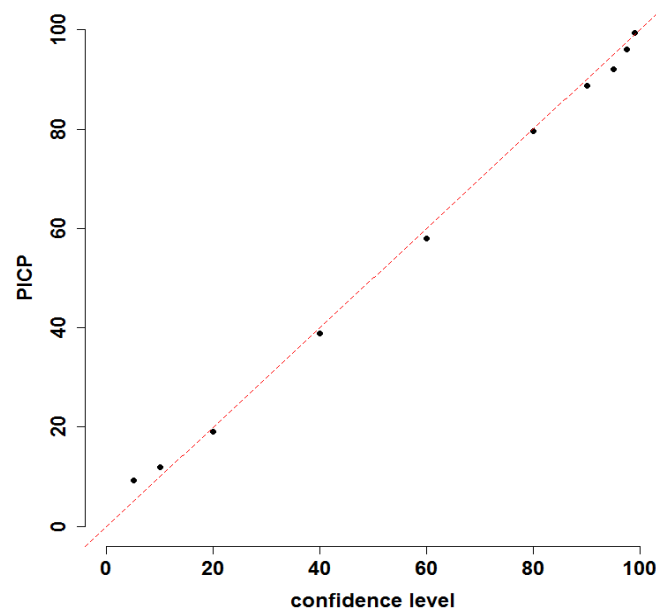


Figure 10: Plot of PICP and confidence level based on validation of Random Forest regression kriging model.

```
quantile(uPL.mat[, 4] - lPL.mat[, 4])
##          0%          25%          50%          75%         100%
## 3.193603 3.470226 3.605872 3.964221 6.416089
```

References

- Bragato, G.
2004. Fuzzy continuous classification and spatial interpolation in conventional soil survey for soil mapping of the lower piave plain. *Geoderma*, 118:1–16.
- Brown, J. D. and G. B. M. Heuvelink
2005. *Encyclopaedia of Hydrological Sciences*, chapter Assessing uncertainty propagation through physically based models of soil water flow solute transport. John Wiley and Sons, Chichester.
- Burrough, P. A., P. F. M. van Gaans, and R. Hootsmans
1997. Continuous classification in soil survey: spatial correlation, confusion and boundaries. *Geoderma*, 77:115–135.
- Efron, B. and R. Tibshirani
1993. *An Introduction to the Bootstrap*. London: Chapman and Hall.

-
- Grimm, R. and T. Behrens
2010. Uncertainty analysis of sample locations within digital soil mapping approaches. *Geoderma*, 155:154–163.
- Lagacherie, P., D. Cazemier, P. vanGaans, and P. Burrough
1997. Fuzzy k-means clustering of fields in an elementary catchment and extrapolation to a larger area. *Geoderma*, 77:197–216.
- Liddicoat, C., D. Maschmedt, D. Clifford, R. Searle, T. Herrmann,
L. Macdonald, and J. Baldock
2015. Predictive mapping of soil organic carbon stocks in south australia's agricultural zone. *Soil Research*, 53:956–973.
- Malone, B. P., A. B. McBratney, and B. Minasny
2011. Empirical estimates of uncertainty for mapping continuous depth functions of soil attributes. *Geoderma*, 160:614–626.
- Malone, B. P., B. Minasny, N. P. Odgers, and A. B. McBratney
2014. Using model averaging to combine soil property rasters from legacy soil maps and from point data. *Geoderma*, 232-234:34–44.
- McBratney, A. and J. de Gruijter
1992. Continuum approach to soil classification by modified fuzzy k-means with extragrades. *Journal of Soil Science*, 43:159–175.
- McBratney, A. B.
1992. On variation, uncertainty and informatics in environmental soil management. *Australian Journal of Soil Research*, 30:913–935.
- McBratney, A. B., M. L. Mendonca Santos, and B. Minasny
2003. On digital soil mapping. *Geoderma*, 117:3–52.
- Minasny, B. and A. B. McBratney
2002. *FuzME version 3.0*. Australian Centre for Precision Agriculture, The University of Sydney, Australia.
- Minasny, B. and A. B. McBratney
2002. Uncertainty analysis for pedotransfer functions. *European Journal of Soil Science*, 53:417–429.
- Odeh, I., A. McBratney, and D. Chittleborough
1992. Soil pattern recognition with fuzzy-c-means: application to classification and soil-landform interrelationships. *Soil Science Society of America Journal*, 56:506–516.
- Shrestha, D. L. and D. P. Solomatine
2006. Machine learning approaches for estimation of prediction interval for the model output. *Neural Networks*, 19:225–235.
- Solomatine, D. P. and D. L. Shrestha
2009. A novel method to estimate model uncertainty using machine learning techniques. *Water Resources Research*, 45:Article Number: W00B11.
- Tranter, G., B. Minasny, and A. B. McBratney
2010. Estimating pedotransfer function prediction limits using fuzzy

k-means with extragrades. *Soil Science Society of America Journal*, 74:1967–1975.

Viscarra Rossel, R. A., C. Chen, M. J. Grundy, R. Searle, D. Clifford, and P. H. Campbell
2015. The australian three-dimensional soil grid: Australias contribution to the globalsoilmap project. *Soil Research*, 53:845–864.

Webster, R.
2000. Is soil variation random? *Geoderma*, 97:149–163.