Empirical uncertainty quantification through fuzzy clustering and cross validation

Soil Security Laboratory

2018

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 1.

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 1 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

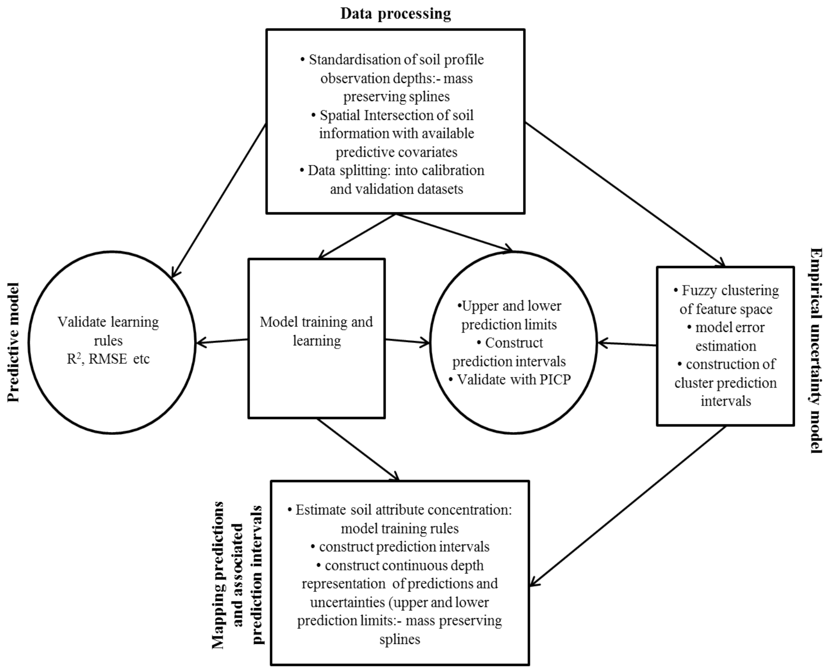
Data processing
• Standardisation of soil profile observation depths:- mass preserving splines
• Spatial Intersection of soil information with available predictive covariates
• Data splitting: into calibration and validation datasets
Predictive model
Validate learning rules
R^2, RMSE etc
Model training and learning
•Upper and lower prediction limits
• Construct prediction intervals
• Validate with PICP
• Fuzzy clustering of feature space
• model error estimation
• construction of cluster prediction intervals
Empirical uncertainty model
Mapping predictions and associated prediction intervals
• Estimate soil attribute concentration: model training rules
• construct prediction intervals
• construct continuous depth representation of predictions and uncertainties (upper and lower prediction limits:- mass preserving splines

Figure 1: 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).

- Creation of maps of both spatial soil predictions and uncertainties.

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

0.1 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.

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

## DATA 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
```

```
# 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 seems 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 0.18822277    0.000
## 2   Sph 0.09624227 1307.864
```

```
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:

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

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; $m_{ij} \epsilon [0,1]$ is the partial membership of the ith observation to the jth cluster. $\psi \geq 1$ is the fuzziness exponent. The square distance between the ith observation to the jth cluster is d_{ij}^{2}. $m_{i\star}^{\psi}$ 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 `Uereq` 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
Uereq.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(Uereq = Uereq.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:

$$PI_i^L = \sum_{j=1}^{c} m_{ij} PIC_j^L and PI_i^U = \sum_{j=1}^{c} m_{ij} PIC_j^U \tag{2}$$

where PI_i^L and PI_i^U correspond to the weighted lower and upper limits for the ith 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 ith 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] 0.9736842 0.9407895 0.9144737 0.8815789 0.7960526 0.5723684 0.3881579
##  [8] 0.2039474 0.1250000 0.1118421
```

```
# 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 to derive and 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       PICP      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 alfa, 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)
```

0.2 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 3.

```
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 2).

```
# 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))
```

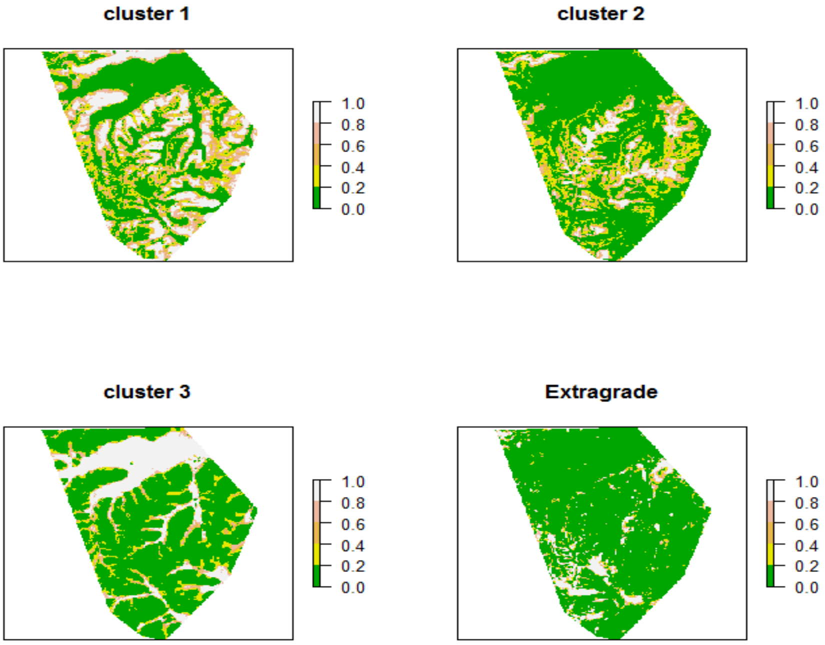

Figure 2: Fuzzy cluster memberships.

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 3).

```
# 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))
```

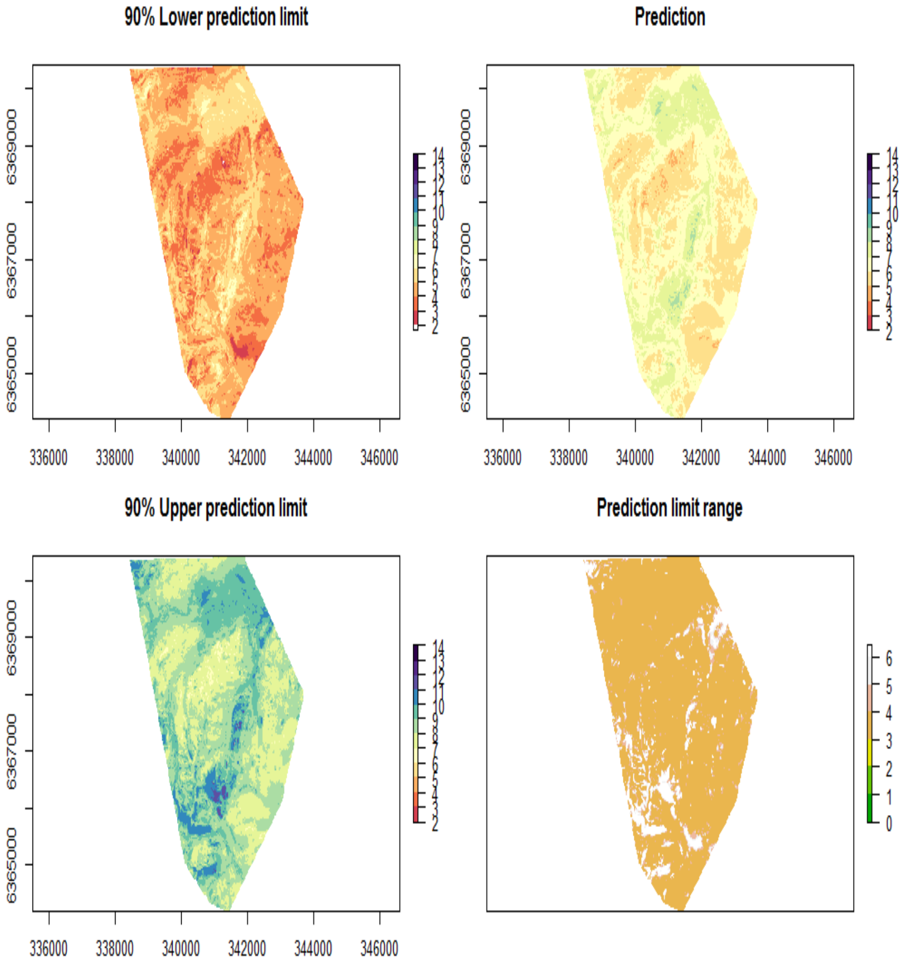

Figure 3: Soil pH predictions and prediction limits derived using a Random Forest regression kriging prediction model together with LOCV and fuzzy classification.

0.3 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 0.190423   0.3733501 1.349081 1.161499 0.08998709

# 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 0.9605263 0.9210526 0.8881579 0.7960526 0.5723684 0.3947368
##  [8] 0.1842105 0.1250000 0.1184211
```

Plotting the PICP against the confidence level provides a nice visual. It can be seen on Figure 4 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")
```

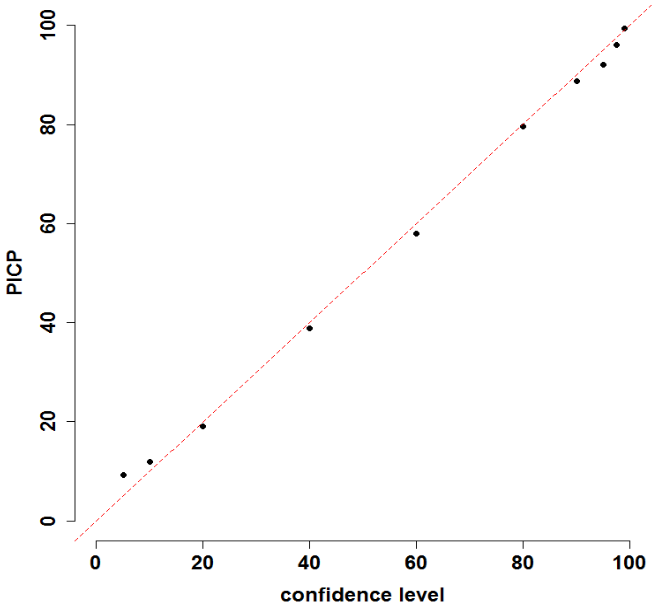

Figure 4: Plot of PICP and confidence level based on validation of Random Forest regression kriging model.

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.

```
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.

Burrough, P. A., P. F. M. van Gaans, and R. Hootsmans
1997. Continuous classifcation in soil survey: spatial correlation, confusion and boundaries. *Geoderma*, 77:115–135.

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.

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.

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.

Minasny, B Mcbratney, A. B.
2002. *FuzME version 3.0.* Australian Centre for Precision Agriculture, The University of Sydney, Australia.

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.

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.