Continuous soil attribute modeling and mapping: Multiple linear regression

Soil Security Laboratory

2017

1 Multiple linear regression

Multiple linear regression (MLR) is where we regress a target variable against more than one covariate. In terms of soil spatial prediction functions, MLR is a least-squares model whereby we want to to predict a continuous soil variable from a suite of covariates. There are a couple of ways to go about this. We could just put everything (all the covariates) in the model and then fit it (estimate the model parameters). We could perform a stepwise regression model where we only enter variables that are statistically significant, based on some selection criteria. Alternatively we could fit what could be termed, an "expert" model, such that based on some pre-determined knowledge of the soil variable we are trying to model, we include covariates that best describe this knowledge. In some ways this is a biased model because we really don't know everything about (the spatial characteristics) the soil property under investigation. Yet in many situations it is better to rely on expert knowledge that is gained in the field as opposed to some other form.

So lets firstly get the data organized. Recall from before in the data preparatory exercises that we were working with the soil point data and environmental covariates for the Hunter Valley area. These data are stored in the `HV_subsoilpH` and `hunterCovariates_sub` objects from the `ithir` package. For the succession of models to be used, we will concentrate on modelling and mapping the soil pH for the 60-100cm depth interval. To refresh, lets load the data in, then intersect the data with the available covariates.

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

# point data
data(HV_subsoilpH)

# Start afresh round pH data to 2 decimal places
HV_subsoilpH$pH60_100cm <- round(HV_subsoilpH$pH60_100cm, 2)
```

```
# remove already intersected data
HV_subsoilpH <- HV_subsoilpH[, 1:3]

# add an id column
HV_subsoilpH$id <- seq(1, nrow(HV_subsoilpH), by = 1)

# re-arrange order of columns
HV_subsoilpH <- HV_subsoilpH[, c(4, 1, 2, 3)]

# Change names of coordinate columns
names(HV_subsoilpH)[2:3] <- c("x", "y")


# grids (covariate raster)
data(hunterCovariates_sub)
```

Perform the covariate intersection.

```
coordinates(HV_subsoilpH) <- ~x + y

# extract
DSM_data <- extract(hunterCovariates_sub, HV_subsoilpH, sp = 1, method = "simple")
DSM_data <- as.data.frame(DSM_data)
str(DSM_data)
```

```
## 'data.frame': 506 obs. of  15 variables:
##  $ id                     : num  1 2 3 4 5 6 7 8 9 10 ...
##  $ 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 7.28 4.95 5.61 5.39 3.44 ...
##  $ 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          : num  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 ...
```

Often it is handy to check to see whether there are missing values both in the target variable and of the covariates. It is possible that a point location does not fit within the extent of the available covariates. In these cases the data should be excluded. A quick way to assess whether there are missing or NA values in the data is to use the `complete.cases` function.

```
which(!complete.cases(DSM_data))
```

```
## integer(0)
```

```
DSM_data <- DSM_data[complete.cases(DSM_data), ]
```

There do not appear to be any missing data as indicated by the `integer(0)` output above i.e there are zero rows with missing information.

With the soil point data prepared, lets fit a model with everything in it (all covariates) to get an idea of how to parametise the MLR models in `R`. Remember the soil variable we are making a model for is soil pH for the 60-100cm depth interval.

```
hv.MLR.Full <- lm(pH60_100cm ~ +Terrain_Ruggedness_Index + AACN + Landsat_Band1 +
    Elevation + Hillshading + Light_insolation + Mid_Slope_Positon + MRVBF +
    NDVI + TWI + Slope, data = DSM_data)
summary(hv.MLR.Full)
```

```
##
## Call:
## lm(formula = pH60_100cm ~ +Terrain_Ruggedness_Index + AACN +
##     Landsat_Band1 + Elevation + Hillshading + Light_insolation +
##     Mid_Slope_Positon + MRVBF + NDVI + TWI + Slope, data = DSM_data)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.2380 -0.7843 -0.1225  0.7057  3.4641
##
## Coefficients:
##                            Estimate Std. Error t value Pr(>|t|)
## (Intercept)                5.372452   2.147673   2.502 0.012689 *
## Terrain_Ruggedness_Index   0.075084   0.054893   1.368 0.171991
## AACN                       0.034747   0.007241   4.798 2.12e-06 ***
## Landsat_Band1             -0.037712   0.009355  -4.031 6.42e-05 ***
## Elevation                 -0.013535   0.005550  -2.439 0.015079 *
## Hillshading                0.152819   0.053655   2.848 0.004580 **
## Light_insolation           0.001329   0.001178   1.127 0.260081
## Mid_Slope_Positon          0.928823   0.268625   3.458 0.000592 ***
## MRVBF                      0.324041   0.084942   3.815 0.000154 ***
## NDVI                       4.982413   0.887322   5.615 3.28e-08 ***
## TWI                        0.085150   0.045976   1.852 0.064615 .
## Slope                     -0.102262   0.062391  -1.639 0.101838
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.178 on 494 degrees of freedom
## Multiple R-squared:  0.2501,Adjusted R-squared:  0.2334
## F-statistic: 14.97 on 11 and 494 DF,  p-value: < 2.2e-16
```

From the summary output above, it seems a few of the covariates are significant in describing the spatial variation of the target variable. To determine the most parsimonious model we could perform a stepwise regression using the `step` function. With this function we can also specify what direction we want step wise algorithm to proceed.

```
hv.MLR.Step <- step(hv.MLR.Full, trace = 0, direction = "both")
summary(hv.MLR.Step)
```

```
##
## Call:
## lm(formula = pH60_100cm ~ AACN + Landsat_Band1 + Elevation +
##     Hillshading + Mid_Slope_Positon + MRVBF + NDVI + TWI, data = DSM_data)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.1202 -0.8055 -0.1286  0.7443  3.4407
##
## Coefficients:
##                    Estimate Std. Error t value Pr(>|t|)
## (Intercept)        7.449369   0.930288   8.008 8.36e-15 ***
## AACN               0.037413   0.006986   5.356 1.31e-07 ***
## Landsat_Band1     -0.037795   0.009134  -4.138 4.12e-05 ***
## Elevation         -0.012042   0.005299  -2.273 0.023481 *
## Hillshading        0.089275   0.018576   4.806 2.04e-06 ***
## Mid_Slope_Positon  0.982066   0.263538   3.726 0.000216 ***
## MRVBF              0.307179   0.083361   3.685 0.000254 ***
## NDVI               5.111642   0.882036   5.795 1.21e-08 ***
## TWI                0.092169   0.045241   2.037 0.042149 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.179 on 497 degrees of freedom
## Multiple R-squared:  0.245,Adjusted R-squared:  0.2329
## F-statistic: 20.16 on 8 and 497 DF,  p-value: < 2.2e-16
```

Comparing the outputs of both the full and stepwise MLR models, there is very little difference in the model diagnostics such as the R^2. Both models explain about 25% of variation of the target variable. Obviously the "full" model is more complex as it has more parameters than the "step" model. If we apply Occam's Razor, the "step" model is preferable.

As described earlier, it is more acceptable to test the performance of a model based upon an external validation. Lets fit a new model using the covariates selected in the step wise regression to a random subset of the available data. We will sample 70% of the available rows for the model calibration data set.

```
set.seed(123)
training <- sample(nrow(DSM_data), 0.7 * nrow(DSM_data))
hv.MLR.rh <- lm(pH60_100cm ~ AACN + Landsat_Band1 + Elevation + Hillshading +
    Mid_Slope_Positon + MRVBF + NDVI + TWI, data = DSM_data[training, ])
# calibration predictions
hv.pred.rhC <- predict(hv.MLR.rh, DSM_data[training, ])

# validation predictions
hv.pred.rhV <- predict(hv.MLR.rh, DSM_data[-training, ])
```

Now we can evaluate the test statistics of the calibration model using the `goof` function.

```
# calibration
goof(observed = DSM_data$pH60_100cm[training], predicted = hv.pred.rhC)
```

```
##           R2 concordance      MSE     RMSE          bias
## 1 0.2437432   0.3936011 1.370124 1.170523 -6.217249e-15
```

```
# validation
goof(observed = DSM_data$pH60_100cm[-training], predicted = hv.pred.rhV)
```

```
##           R2 concordance      MSE     RMSE       bias
## 1 0.2248642   0.3798786 1.376643 1.173304 0.05121269
```

In this situation the calibration model does not appear to be over fitting because the test statistics for the validation are similar to those of the calibration data. While this is a good result, the prediction model performs only moderately well by the fact there is a noticeable deviation between observations and corresponding model predictions. Examining other candidate models is a way to try to improve upon this results.

1.1 Applying the model spatially

From a soil mapping perspective the important question to ask is: What does the map look like that results from a particular model? In practice this can be answered by applying the model parameters to the grids of the covariates that were used in the model. There are a few options on how to do this.

1.1.1 Covariate table

The traditional has been to collate a grid table where there would be two columns for the coordinates followed by other columns for each of the available covariates that were sourced. This was seen as an efficient way to organize all the covariate data as it ensured that a common grid was used which also meant that all the covariates are of the same scale in terms of resolution and extent. We can simulate the covariate table approach using the `hunterCovariates_sub` object as below.

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

```
## '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 ...
```

The result shown above is that the covariate table contains 33252 rows and has 14 variables. It is always necessary to have the coordinate columns, but some saving of memory could be earned if only the required covariates are appended to the table. It will quickly become obvious however that the covariate table approach could be limiting when mapping extents get very large or the grid resolution of mapping becomes more fine-grained, or both.

With the covariate table arranged it then becomes a matter of using the MLR `predict` function.

```
map.MLR <- predict(hv.MLR.rh, newdata = tempD)
map.MLR <- cbind(data.frame(tempD[, c("x", "y")]), map.MLR)
```

Now we can rasterise the predictions for mapping (Figure 1) and grid export. In the example below we set the CRS to WGS84 Zone 56 before exporting the raster file out as a Geotiff file.

```
map.MLR.r <- rasterFromXYZ(as.data.frame(map.MLR[, 1:3]))
plot(map.MLR.r, main = "MLR predicted soil pH (60-100cm)")
# set the projection
crs(map.MLR.r) <- "+proj=utm +zone=56 + south + ellps=WGS84 +datum=WGS84 +units=m +no_defs
writeRaster(map.MLR.r, "soilpH_60_100_MLR.tif", format = "GTiff", datatype = "FLT4S",
    overwrite = TRUE)
# check working directory for presence of raster
```

Some of the parameters used within the `writeRaster` function that are worth noting include: `format`, which is the raster format that we want to write to. Here "GTiff" is being specified — use the `writeFormats` fuction to look at what other raster formats can be used. the parameter `datatype` is specified as "FLT4S" which inidcates that a 4 byte, signed floating point values are to be written to file. Look at the function `dataType` to look at other alternatives, for example for categorical data where we may be interested in logical or integer values.

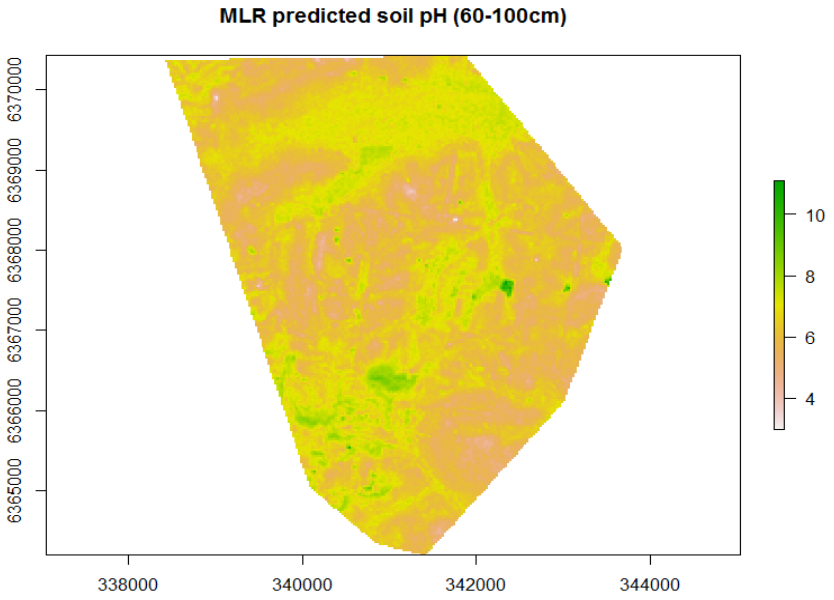

Figure 1: MLR predicted soil pH 60-100cm across the Hunter Valley

1.1.2 Raster predictions

Probably a more efficient way of applying the fitted model is to apply it directly to the rasters themselves. This avoids the step of arranging all covariates into table format. If multiple rasters are being used, it is necessary to have them arranged as a `rasterStack` object. This is useful as it also ensures all the rasters are of the same extent and resolution. Here we can use the `raster predict` function such as below using the `covStack` raster stack as input.

```
map.MLR.r1 <- predict(hunterCovariates_sub, hv.MLR.rh, "soilpH_60_100_MLR.tif",
    format = "GTiff", datatype = "FLT4S", overwrite = TRUE)
# check working directory for presence of raster
```

The prediction function is quite versatile. For example we can also map the standard error of prediction or the confidence interval or the prediction interval even. The script below is an example of creating maps of the 90% prediction intervals for the `hv.MLR.rh` model. We need to explicitly create a function called in this case `predfun` which will direct the raster `predict` function to output the predictions plus the upper and lower prediction limits. In the `predict` function we insert `predfun` for the `fun` parameter and control the output by changing the `index` value to either 1, 2, or 3 to request either the prediction, lower limit, upper limit respectively. Setting the `level` paramter to 0.90 indicates that we want to return the 90% precition interval. The resulting plots are shown in Figure 2.

```
par(mfrow = c(3, 1))
predfun <- function(model, data) {
    v <- predict(model, data, interval = "prediction", level = 0.9)
}

map.MLR.r.1ow <- predict(hunterCovariates_sub, hv.MLR.rh, "soilPh_60_100_MLR_low.tif",
    fun = predfun, index = 2, format = "GTiff", datatype = "FLT4S", overwrite = TRUE)
plot(map.MLR.r.1ow, main = "MLR predicted soil pH (60-100cm) lower limit")

map.MLR.r.pred <- predict(hunterCovariates_sub, hv.MLR.rh, "soilPh_60_100_MLR_pred.tif",
    fun = predfun, index = 1, format = "GTiff", datatype = "FLT4S", overwrite = TRUE)
plot(map.MLR.r.pred, main = "MLR predicted soil pH (60-100cm)")

map.MLR.r.up <- predict(hunterCovariates_sub, hv.MLR.rh, "soilPh_60_100_MLR_up.tif",
    fun = predfun, index = 3, format = "GTiff", datatype = "FLT4S", overwrite = TRUE)
plot(map.MLR.r.up, main = "MLR predicted soil pH (60-100cm) upper limit")
# check working directory for presence of rasters
```

1.1.3 Directly to rasters using parallel processing

An extension of using the raster `predict` function is to apply the model again to the rasters, but to do it across multiple computer nodes. This is akin to breaking a job up into smaller pieces then processing the jobs in parallel rather than sequentially. The parallel component here is that the smaller pieces are passed to more than 1 compute nodes. Most desktop computers these days can have up to 8 compute nodes which can result in some excellent gains in efficiency when applying models across massive extents and or at fine resolutions. The `raster` package has some built in dependencies with other `R` packages that facilitate parallel processing options. For example the `raster` package ports with the `parallel` package for setting up and controlling the compute node processes. The script below is an example of using 4 compute nodes to apply the `hv.MLR.rh` model to the `hunterCovariates_sub` raster stack.

```
library(parallel)
beginCluster(4)
cluserMLR.pred <- clusterR(hunterCovariates_sub, predict, args = list(hv.MLR.rh),
    filename = "soilpH_60_100_MLR_pred.tif", format = "GTiff", progress = FALSE,
    overwrite = T)
endCluster()
```

To set up the compute nodes, you use the `beginCluster` function and inside it, specify how many compute nodes you want to use. If empty brackets are used, the function will use 100% of the compute resources. The `clusterR` function is the work horse function that then applies the model in parallel to the rasters. The parameters and subsequent options are similar to the raster `predict` function, although it would help to look at the help files on this function for more detailed explanations. It is always important after the prediction is completed to shutdown the nodes using the `endCluster`

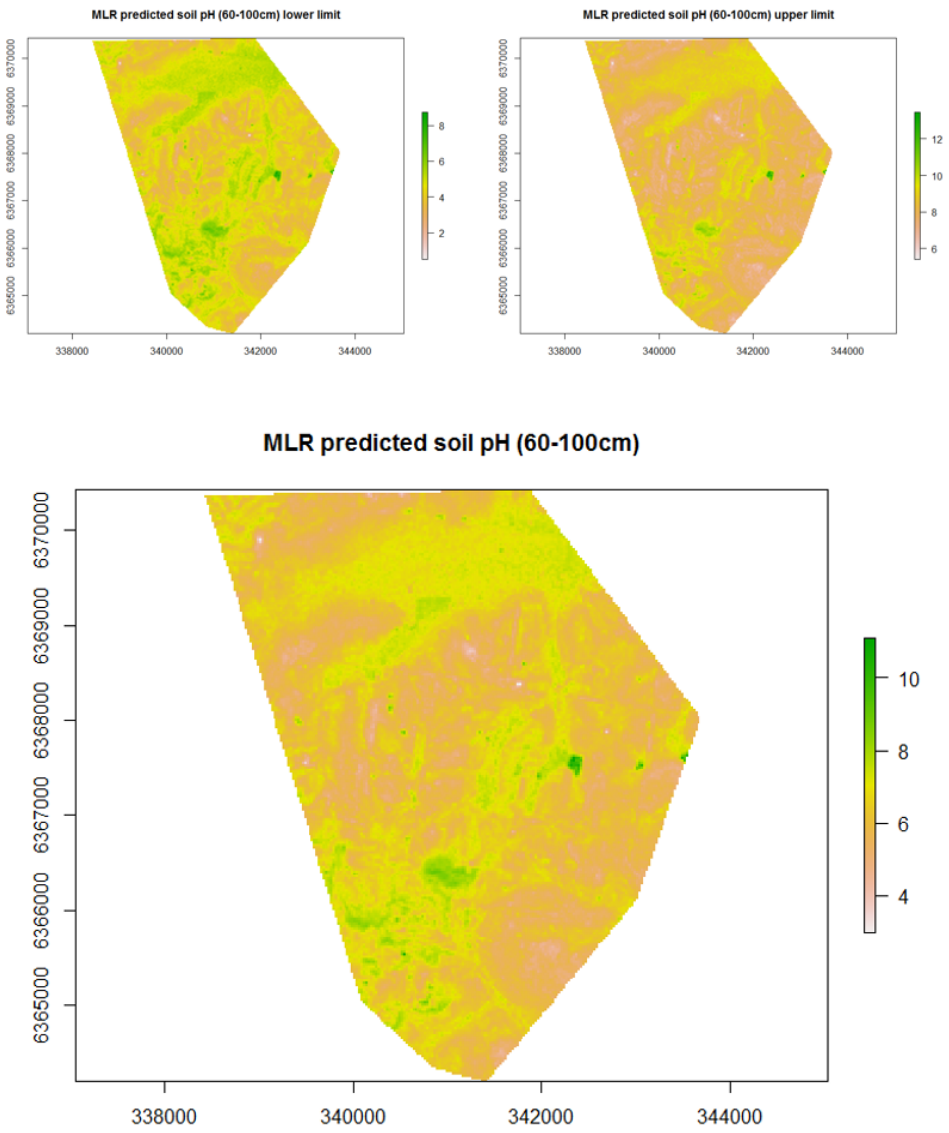

Figure 2: MLR predicted soil pH (60-100cm) across the Hunter Valley with associated lower and upper prediction limits.

function.

The relative ease in setting up the parallel processing for our mapping needs has really opened up the potential for performing DSM using very large data sets and rasters. Moreover, using the parallel processing together with the file pointing ability (that was discussed earlier) `raster` has made the possibility of big DSM a reality, and importantly- practicable.

References