Categorical soil attribute modeling and mapping

Soil Security Laboratory

2017

The other form of soil spatial prediction functions are those dealing with categorical target variables such as soil classes. Naturally the models we will use further on are not specific to soil classes but can be generally applied to any type of categorical data too.

In the examples below we will demonstrate the prediction of soil-landscape classes termed: Terrons. Terron relates to a soil and landscape concept that has an associated model. The concept was first described by Carre and McBratney (2005). The embodiment of Terron is a continuous soil-landscape unit or class which combines soil knowledge, landscape information, and their interactions together. Malone et al. (2014) detailed an approach for defining Terrons in the Lower Hunter Valley, NSW, Australia. This area is a prominent wine-growing region of Australia, and the development of Terrons is a first step in the realization of viticultural terroir — an established concept of identity that incorporates much more than just soil and landscape qualities (Vaudour et al., 2015). In the work by Malone et al. (2014) they defined 12 Terrons for the Hunter Valley, which are distinguished by such soil and landscape characteristics as: geomorphometric attributes (derived from a digital elevation model) and specific attributes pertaining to soil pH, clay percentage, soil mineralogy (clay types and presence of iron oxides), continuous soil classes, and presence or absence of marl. The brief in this chapter is to predict the Terron classes across the prescribed Lower Hunter Valley, given a set of observations (sampled directly from the Terron map produced by Malone et al. (2014)).

By now you will be familiar with the process of fitting models and plotting/mapping the outputs. Thus in many ways, categorical data modeling is similar (in terms of implementation) with prediction models of continuous variables. The example in the previous chapter demonstrating the use of the `caret` package can also be similarly applied for categorial variables too, where you will find many model functions suited to that type of data with that package.

In this chapter we will have a look at a few different classification models:

1. Multinomial logistic regression
2. Data mining with the C5 algorithm
3. Random Forest

1 Model validation of categorical prediction models

The special characteristic of categorical data and its prediction within models, is that a class is either predicted or it is not. For binary variables. the prediction is either a *yes* or *no*, *black* or *white*, *present* or *absent* etc. For multinomial variables, there are more than 2 classes, for example either *black*, *grey*, or *white* etc (which could actually be an ordinal categorical classification, rather than nominal).

There is no in-between; rather discrete entities. Exceptions are that some models do estimate the probability of the existence of a particular class, which will be touched on later. Additionally, there are methods of fuzzy classification which are common in the soil sciences (McBratney and Odeh, 1997), but will not be covered in this section. Discrete categories and models for their prediction require other measures of validation than those that were used for continuous variables. The most important quality measures are described in Congalton (1991) and include:

1. Overall accuracy
2. User's accuracy
3. Producer's accuracy
4. Kappa coefficient of agreement

Using a contrived example, each of these quality measures will be illustrated. We will make a 4×4 matrix, and call it `con.mat`, and append some column and row names — in this case Australian Soil Classification Order codes. We then populate the matrix with some more-or-less random integers.

```
con.mat <- matrix(c(5, 0, 1, 2, 0, 15, 0, 5, 0, 1, 31, 0, 0, 10, 2, 11),
nrow = 4, ncol = 4)
rownames(con.mat) <- c("DE", "VE", "CH", "KU")
colnames(con.mat) <- c("DE", "VE", "CH", "KU")
con.mat
```

```
##    DE VE CH KU
## DE  5  0  0  0
## VE  0 15  1 10
## CH  1  0 31  2
## KU  2  5  0 11
```

`con.mat` takes the form of a confusion matrix, and ones such as this are often the output of a classification model. If we summed each of the columns (using the `colSums` function), we would obtain the total number of observations for each soil class. Having column sums reflecting the number of observations is a widely used convention in classification studies.

```
colSums(con.mat)
```

```
## DE VE CH KU
##  8 20 32 23
```

Similarly, if we summed each of the rows we would retrieve the total number of predictions of each soil class. The predictions could have been made through any sort of model or classification process.

```
rowSums(con.mat)
```

```
## DE VE CH KU
##  5 26 34 18
```

Therefore, the numbers on the diagonal of the matrix will indicate fidelity between the observed class and the subsequent prediction. Numbers on the off-diagonals indicate a mis-classification or error. Overall accuracy is therefore computed by dividing the total correct (i.e., the sum of the diagonal) by the total number of observations (sum of the column sums).

```
ceiling(sum(diag(con.mat))/sum(colSums(con.mat)) * 100)
```

```
## [1] 75
```

Accuracy of individual classes can be computed in a similar manner. However, there is a choice of dividing the number of correct predictions for each class by either the totals (observations or predictions) in the corresponding columns or rows respectively. Traditionally, the total number of correct predictions of a class is divided by the total number of observations of that class (i.e. the column sum). This accuracy measure indicates the probability of an observation being correctly classified and is really a measure of omission error, or the "producer's accuracy". This is because the producer of the model is interested in how well a certain class can be predicted.

```
ceiling(diag(con.mat)/colSums(con.mat) * 100)
```

```
## DE VE CH KU
## 63 75 97 48
```

Alternatively, if the total number of correct predictions of a class is divided by the total number of predictions that were predicted in that category, then this result is a measure of commission error, or "user's accuracy". This measure is indicative of the probability that a prediction on the map actually represents that particular category on the ground or in the field.

```
ceiling(diag(con.mat)/rowSums(con.mat) * 100)
```

```
##  DE  VE  CH  KU
## 100  58  92  62
```

So if we use the `DE` category as an example, the "model" predicts this class correctly 63% of the time, but when it is actually predicted it is correct 100% of the time.

The Kappa coefficient is another statistical measure of the fidelity between observations and predictions of a classification. The calculation is based on the difference between how much agreement is actually present ("observed" agreement) compared to how much agreement would be expected to be present by chance alone ("expected" agreement). The observed agreement is simply the overall accuracy percentage. We may also want to know how different the observed agreement is from the expected agreement. The Kappa

coefficient is a measure of this difference, standardized to lie on a -1 to 1 scale, where 1 is perfect agreement, 0 is exactly what would be expected by chance, and negative values indicate agreement less than chance, i.e., potential systematic disagreement between observations and predictions. The Kappa coefficient is defined as:

$$K = \frac{p_o - p_e}{1 - P_e} \tag{1}$$

where p_o is the overall or observed accuracy, and p_e is the expected accuracy, where:

$$p_e = \sum_{i=1}^{n} (\frac{colSum_i}{TO}) \times (\frac{rowSum_i}{TO}) \tag{2}$$

TO is the total number of observations and n is the number of classes. Rather than scripting the above equations, the kappa coefficient together with the other accuracy measures are contained in a function called `goofcat` in the `ithir` package. As we already have a confusion matrix prepared, we can enter it directly into the function as in the script below.

```
goofcat(conf.mat = con.mat, imp = TRUE)
```

```
## $confusion_matrix
##    DE VE CH KU
## DE  5  0  0  0
## VE  0 15  1 10
## CH  1  0 31  2
## KU  2  5  0 11
##
## $overall_accuracy
## [1] 75
##
## $producers_accuracy
## DE VE CH KU
## 63 75 97 48
##
## $users_accuracy
##  DE  VE  CH  KU
## 100  58  92  62
##
## $kappa
## [1] 0.6389062
```

A rule of thumb as indicated in Landis and Koch (1977) for the interpretation of the Kappa coefficient is:

< Less than chance agreement.
0.01 - 0.20 Slight agreement.
0.21 - 0.40 Fair agreement.
0.41 - 0.60 Moderate agreement.

0.61 - 0.80 Substantial agreement.
0.80 - 0.99 Almost perfect agreement.

We will be using these prediction quality indices for categorical variable prediction models in the following examples.

2 Multinomial logistic regression

Multinomial logistic regression is used to model nominal outcome variables, in which the log odds of the outcomes are modeled as a linear combination of the predictor variables. Because we are dealing with categorical variables, it is necessary that logistic regression take the natural logarithm of the odds (log-odds) to create a continuous criterion. The logit of success is then fit to the predictors using regression analysis. The results of the logit, however are not intuitive, so the logit is converted back to the odds via the inverse of the natural logarithm, namely the exponential function. Therefore, although the observed variables in logistic regression are categorical, the predicted scores are modeled as a continuous variable (the logit). The logit is referred to as the link function in logistic regression. As such, although the output in logistic regression will be multinomial, the logit is an underlying continuous criterion upon which linear regression is conducted. This means that for logistic regression we are able to return the most likely or probable prediction (class) as well as the probabilities of occurrence for all the other classes considered. Some discussion of the theoretical underpinnings of multinomial logistic regression, and importantly its application in DSM is given in Kempen et al. (2009).

In `R` we can use the `multinom` function from the `nnet` package to perform logistic regression. The are other implementations of this model in `R`, so it is worth a look to compare and contrast them. Fitting `multinom` is just like fitting a linear model as seen below.

As described earlier, the data to be used for the following modelling exercises are Terron classes as sampled from the map presented in Malone et al. (2014).The sample data contains 1000 entries of which there are 12 different Terron classes. Before getting into the modeling, we first load in the data and then perform the covariate layer intersection using the suite of environmental variables contained in the `hunterCovariates` data object in the `ithir` package. The small selection of covariates cover an area of approximately $220km^2$ at a spatial resolution of 25m. They include those derived from a DEM: altitude above channel network (AACN), solar light insolation and terrain wetness index (TWI). Gamma radiometric data (total count) is also included together with a surface that depicts soil drainage in form of a continuous index (ranging from 0 to 5). These 5 different covariate layers are stacked together via a `rasterStack`.

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

```
data(hvTerronDat)
data(hunterCovariates)
```

Transform the `hvTerronDat` data to a `SpatialPointsDataFrame`.

```
names(hvTerronDat)
```

```
## [1] "x"      "y"      "terron"
```

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

As these data are of the same spatial projection as the `hunterCovariates`, there is no need to perform a coordinate transformation. So we can perform the intersection immediately.

```
DSM_data <- extract(hunterCovariates, hvTerronDat, sp = 1, method = "simple")
DSM_data <- as.data.frame(DSM_data)
str(DSM_data)
```

```
## 'data.frame': 1000 obs. of  8 variables:
##  $ x                : num  346535 334760 340910 336460 344510 ...
##  $ y                : num  6371941 6375841 6377691 6382041 6378116 ...
##  $ terron           : Factor w/ 12 levels "1","2","3","4",..: 3 4 12 5 6 11 3 10 3 5 ..
##  $ AACN             : num  37.544 25.564 32.865 0.605 9.516 ...
##  $ Drainage.Index   : num  4.72 4.78 2 4.19 4.68 ...
##  $ Light.Insolation : num  1690 1736 1712 1712 1677 ...
##  $ TWI              : num  11.5 13.8 13.4 18.6 19.8 ...
##  $ Gamma.Total.Count: num  380 407 384 388 454 ...
```

It is always good practice to check to see if any of the observational data returned any NA values for any one of the covariates. If there is NA values, it indicates that the observational data is outside the extent of the covariate layers. It is best to remove these observations from the data set.

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

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

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

Now for model fitting. The target variable is `terron`. So lets just use all the available covariates in the model. We will also subset the data for an external validation i.e. random hold back validation.

```
library(nnet)

set.seed(655)
training <- sample(nrow(DSM_data), 0.7 * nrow(DSM_data))
hv.MNLR <- multinom(terron ~ AACN + Drainage.Index + Light.Insolation + TWI +
    Gamma.Total.Count, data = DSM_data[training, ])
```

```
## # weights:  84 (66 variable)
## initial  value 1739.434655
## iter  10 value 1544.694562
## iter  20 value 1368.990795
```

```
## iter  30 value 1236.748119
## iter  40 value 1161.299395
## iter  50 value 1063.598067
## iter  60 value 992.924808
## iter  70 value 968.797316
## iter  80 value 963.189954
## iter  90 value 962.000276
## iter 100 value 961.634415
## final  value 961.634415
## stopped after 100 iterations
```

Using the `summary` function allows us to see the linear models for each Terron class, which are the result of the log-odds of each soil class modeled as a linear combination of the covariates. We can also see the probabilities of occurrence for each Terron class at each observation location by using the `fitted` function.

```
summary(hv.MNLR)

# Estimate class probabilities
probs.hv.MNLR <- fitted(hv.MNLR)

# return top of data frame of probabilites
head(probs.hv.MNLR)
```

Subsequently, we can also determine the most probable Terron class using the the `predict` function.

```
pred.hv.MNLR <- predict(hv.MNLR)
summary(pred.hv.MNLR)
```

```
##   1   2   3   4   5   6   7   8   9  10  11  12
##  21   7  60  62 110  73 169 115  18  29  12  24
```

Lets now perform an internal validation of the model to assess its general performance. Here we use the `goofcat` function, but this time we import the two vectors into the function which correspond to the observations and predictions respectively.

```
goofcat(observed = DSM_data$terron[training], predicted = pred.hv.MNLR)
```

```
## $confusion_matrix
##     1 2  3  4  5  6  7  8  9 10 11 12
## 1  13 3  1  2  0  0  1  0  1  0  0  0
## 2   1 5  0  0  0  0  0  0  1  0  0  0
## 3   0 1 35  2  0  0  8  0  2  3  6  3
## 4   4 0  9 30  1  1  3  3  5  0  6  0
## 5   0 0  0  0 56 10  1 10 13 17  1  2
## 6   0 0  0  1 16 47  0  6  1  2  0  0
## 7   0 0 14  7  4  0 92 16  7  8  9 12
## 8   0 0  0  4 10  7 14 57  4  6 13  0
## 9   0 0  0  5  3  0  2  2  5  0  1  0
```

```
## 10  0 0  0  0  4  1  2  6  3 13  0  0
## 11  0 0  4  3  0  0  0  3  0  0  2  0
## 12  1 0  2  1  0  0  1  0  0  4  0 15
##
## $overall_accuracy
## [1] 53
##
## $producers_accuracy
##  1  2  3  4  5  6  7  8  9 10 11 12
## 69 56 54 55 60 72 75 56 12 25  6 47
##
## $users_accuracy
##  1  2  3  4  5  6  7  8  9 10 11 12
## 62 72 59 49 51 65 55 50 28 45 17 63
##
## $kappa
## [1] 0.4637285
```

Similarly, performing the external validation requires first using the `pred.hv.MNLR` model to predict on the withheld points.

```
V.pred.hv.MNLR <- predict(hv.MNLR, newdata = DSM_data[-training, ])
goofcat(observed = DSM_data$terron[-training], predicted = V.pred.hv.MNLR)
```

```
## $confusion_matrix
##    1 2  3 4  5  6  7  8 9 10 11 12
## 1  1 1  1 2  0  0  0  0 1  0  0  0
## 2  5 0  2 0  0  0  0  0 1  1  0  0
## 3  0 0 13 1  0  0  5  0 1  2  2  2
## 4  2 3  9 8  0  0  0  4 8  0  3  0
## 5  0 0  0 0 21  7  0  8 6  3  0  2
## 6  0 0  0 1  8 18  0  5 1  0  0  0
## 7  0 0  8 4  0  0 38  9 4  5  2  5
## 8  0 0  0 3  3  0  6 15 1  3  1  0
## 9  0 0  0 1  2  0  0  1 0  0  0  0
## 10 0 0  0 0  4  1  1  4 1  9  2  0
## 11 0 0  0 0  0  0  0  1 0  0  0  0
## 12 0 0  1 0  0  0  1  0 0  2  0  4
##
## $overall_accuracy
## [1] 43
##
## $producers_accuracy
##  1  2  3  4  5  6  7  8  9 10 11 12
## 13  0 39 40 56 70 75 32  0 36  0 31
##
## $users_accuracy
##  1  2  3  4  5  6  7  8  9 10 11 12
## 17  0 50 22 45 55 51 47  0 41  0 50
##
```

```
## $kappa
## [1] 0.3476539
```

Using the raster `predict` function is the method for applying the `hv.MNLR` model across the whole area. Note that the `clusterR` function can also be used here too if there is a requirement to perform the spatial prediction across multiple compute nodes. Note also that it is also possible for multinomial logistic regression to create the map of the most probable class, as well as the probabilities for all classes. The first script example below is for mapping the most probable class which is specified by setting the `type` parameter to "`class`". If probabilities are required "`probs`" would be used for the `type` parameter, together with specifying an `index` integer to indicate which class probabilities you wish to map. The second script example below shows the parametisation for predicting the probabilities for Terron class 1.

```
# class prediction
map.MNLR.c <- predict(hunterCovariates, hv.MNLR, type = "class",
filename = "hv_MNLR_class.tif", format = "GTiff",
overwrite = T, datatype = "INT2S")

# class probabilities
map.MNLR.p <- predict(hunterCovariates, hv.MNLR, type = "probs", index = 1,
filename = "edge_MNLR_probs1.tif", format = "GTiff",
overwrite = T, datatype = "FLT4S")
```

Plotting the resulting class map is not as straightforward as for mapping continuous variables. A solution is scripted below which uses an associated package to `raster` called `rasterVis`. You will also note the use of explicit colors for each Terron class as they were the same colors used in the Terron map presented by Malone et al. (2014). The colors are defined in terms of HEX color codes. A very good resource for selecting colors or deciding on color ramps for maps is *colorbrewer* located at `http://colorbrewer2.org/`. The script below produces the map in Figure 1.

```
map.MNLR.c <- as.factor(map.MNLR.c)

## Add a land class column to the Raster Attribute Table
rat <- levels(map.MNLR.c)[[1]]
rat[["terron"]] <- c("HVT_001", "HVT_002", "HVT_003", "HVT_004", "HVT_005",
    "HVT_006", "HVT_007", "HVT_008", "HVT_009", "HVT_010", "HVT_011", "HVT_012")
levels(map.MNLR.c) <- rat

## HEX colors
area_colors <- c("#FF0000", "#38A800", "#73DFFF", "#FFEBAF", "#A8A800", "#0070FF",
    "#FFA77F", "#7AF5CA", "#D7B09E", "#CCCCCC", "#B4D79E", "#FFFF00")
# plot
levelplot(map.MNLR.c, col.regions = area_colors, xlab = "", ylab = "")
```

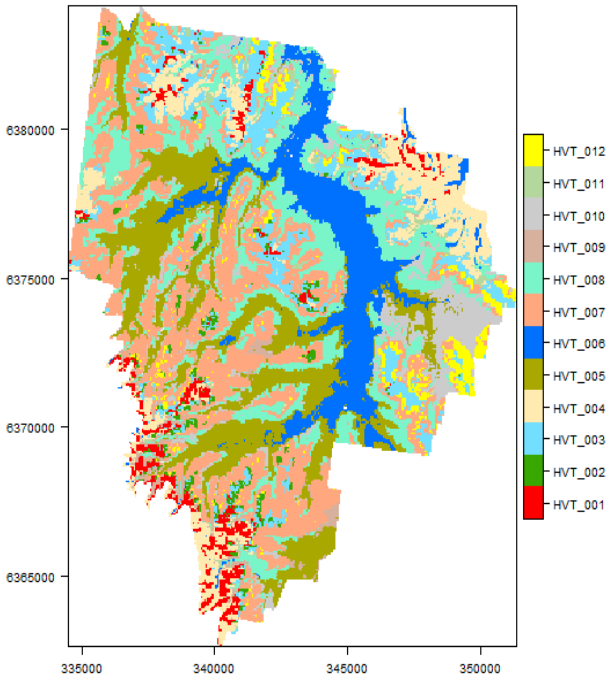

Figure 1: Hunter Valley most probable Terron class map created using multinomial logistic regression model

3 C5 decision trees

The C5 decision tree model is available in the `C50` package. The function `C5.0` fits classification tree models or rule-based models using Quinlans's C5.0 algorithm (Quinlan, 1993).

Essentially we will go through the same process as we did for the multinomial logistic regression. The `C5.0` function and its internal parameters are similar in nature to that of the `Cubist` function for predicting continuous variables. The `trials` parameter lets you implement a "boosted" classification tree process, with the results aggregated at the end. There are also many other useful model tuning parameters in the `C5.0Control` parameter set too that are worth a look. Just see the help files for more information.

Using the same training and validation sets as before, we will fit the C5 model as in the script below.

```
library(C50)
hv.C5 <- C5.0(x = DSM_data[training, c("AACN", "Drainage.Index",
"Light.Insolation", "TWI", "Gamma.Total.Count")],
y = DSM_data$terron[training], trials = 1, rules = FALSE,
control = C5.0Control(CF = 0.95, minCases = 20, earlyStopping = FALSE))
```

By calling the `summary` function, some useful model fit diagnostics are given.

These include the tree structure, together with the covariates used and mis-classification error of the model, as well as a rudimentary confusion matrix. A useful feature of the C5 model is that it can omit in an automated fashion, unnecessary covariates.

The `predict` function can either return the predicted class or the confidence of the predictions (which is controlled using the `type=''prob''` parameter). The probabilities are quantified such that if an observation is classified by a single leaf of a decision tree, the confidence value is the proportion of training cases at that leaf that belong to the predicted class. If more than one leaf is involved (i.e., one or more of the attributes on the path has missing values), the value is a weighted sum of the individual leaves' confidences. For rule-classifiers, each applicable rule votes for a class with the voting weight being the rule's confidence value. If the sum of the votes for class C is W(C), then the predicted class P is chosen so that W(P) is maximal, and the confidence is the greater of (1), the voting weight of the most specific applicable rule for predicted class P; or (2) the average vote for class P (so, W(P) divided by the number of applicable rules for class P).Boosted classifiers are similar, but individual classifiers vote for a class with weight equal to their confidence value. Overall, the confidence associated with each class for every observation are made to sum to 1.

```
# return the class predictions
predict(hv.C5, newdata = DSM_data[training, ])

# return the class probabilities
predict(hv.C5, newdata = DSM_data[training, ], type = "prob")
```

So lets look at the calibration and validation statistics. First, the calibration statistics:

```
C.pred.hv.C5 <- predict(hv.C5, newdata = DSM_data[training, ])
goofcat(observed = DSM_data$terron[training], predicted = C.pred.hv.C5)
```

```
## $confusion_matrix
##     1 2  3  4  5  6  7  8  9 10 11 12
## 1  17 9  4  4  0  0  0  0  1  0  0  1
## 2   0 0  0  0  0  0  0  0  0  0  0  0
## 3   0 0 42  4  0  0 13  0  4  3  7 14
## 4   2 0 12 37  0  1  5  6  9  1  9  1
## 5   0 0  0  0 48  2  1  9 14 18  0  0
## 6   0 0  0  0 22 55  0  3  2  1  1  0
## 7   0 0  7  6  7  0 95 18  9 12  5 14
## 8   0 0  0  4 12  8  2 61  3  4  9  2
## 9   0 0  0  0  0  0  0  0  0  0  0  0
## 10  0 0  0  0  5  0  8  6  0 14  7  0
## 11  0 0  0  0  0  0  0  0  0  0  0  0
## 12  0 0  0  0  0  0  0  0  0  0  0  0
##
## $overall_accuracy
## [1] 53
##
```

```
## $producers_accuracy
##  1  2  3  4  5  6  7  8  9 10 11 12
## 90  0 65 68 52 84 77 60  0 27  0  0
##
## $users_accuracy
##   1   2   3   4   5   6   7   8   9  10  11  12
##  48 NaN  49  45  53  66  55  59 NaN  35 NaN NaN
##
## $kappa
## [1] 0.4618099
```

It will be noticed that some of the Terron classes failed to be predicted by the fitted model. For example Terron classes 2, 9, 11, and 12 were all predicted as being a different class. All observations of Terron class 2 were predicted as Terron class 1. Doing the external validation we return the following:

```
V.pred.hv.C5 <- predict(hv.C5, newdata = DSM_data[-training, ])
goofcat(observed = DSM_data$terron[-training], predicted = V.pred.hv.C5)
```

```
## $confusion_matrix
##    1 2  3 4  5  6  7  8 9 10 11 12
## 1  7 3  5 1  0  0  1  0 2  1  0  1
## 2  0 0  0 0  0  0  0  0 0  0  0  0
## 3  0 0 14 1  0  0  4  0 2  2  2  6
## 4  1 1  5 9  0  0  4  3 7  0  0  0
## 5  0 0  0 0 20  2  0  7 5  2  0  1
## 6  0 0  0 0 10 22  0  7 0  1  0  0
## 7  0 0 10 5  0  0 35 11 2  7  5  3
## 8  0 0  0 3  6  2  4 16 2  5  2  2
## 9  0 0  0 0  0  0  0  0 0  0  0  0
## 10 0 0  0 1  2  0  3  3 4  7  1  0
## 11 0 0  0 0  0  0  0  0 0  0  0  0
## 12 0 0  0 0  0  0  0  0 0  0  0  0
##
## $overall_accuracy
## [1] 44
##
## $producers_accuracy
##  1  2  3  4  5  6  7  8  9 10 11 12
## 88  0 42 45 53 85 69 35  0 29  0  0
##
## $users_accuracy
##   1   2   3   4   5   6   7   8   9  10  11  12
##  34 NaN  46  30  55  56  45  39 NaN  34 NaN NaN
##
## $kappa
## [1] 0.3565075
```

And finally, creating the map derived from the `hv.C5` model using the raster `predict` function (Figure 2). Note that the C5 model returned 0% producer's accuracy for Terron classes 2, 9, 11 and 12. These data account for only a

small proportion of the data set, and/or, they may be similar to other existing Terron classes (based on the available predictive covariates). Consequently, they did not feature in the `hv.C5` model and ultimately, the final map.

```
# class prediction
map.C5.c <- predict(hunterCovariates, hv.C5, type = "class",
filename = "hv_C5_class.tif", format = "GTiff",
overwrite = T, datatype = "INT2S")

map.C5.c <- as.factor(map.C5.c)
rat <- levels(map.C5.c)[[1]]
rat[["terron"]] <- c("HVT_001", "HVT_003", "HVT_004", "HVT_005", "HVT_006",
    "HVT_007", "HVT_008", "HVT_010")
levels(map.C5.c) <- rat

# plot
area_colors <- c("#FF0000", "#73DFFF", "#FFEBAF", "#A8A800", "#0070FF", "#FFA77F",
    "#7AF5CA", "#CCCCCC")
levelplot(map.C5.c, col.regions = area_colors, xlab = "", ylab = "")
```

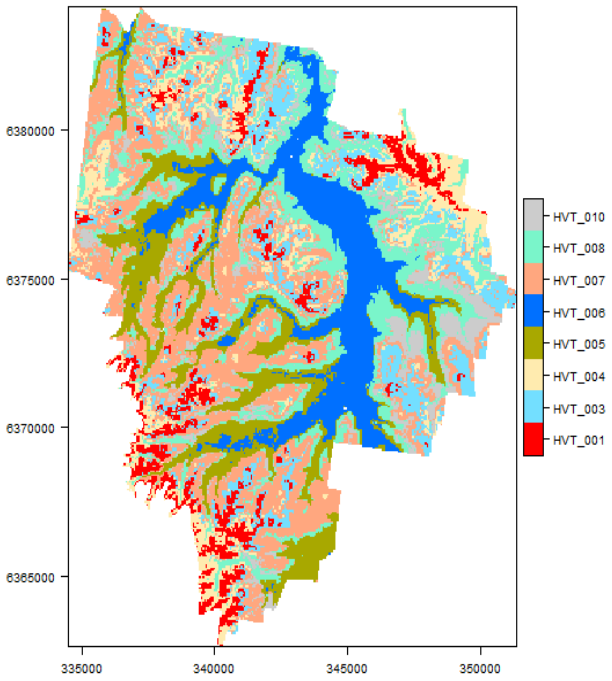

Figure 2: Hunter Valley Terron class map created using C5 decision tree model.

4 Random Forests

The final model we will look at is the Random Forest, which we should be familiar with now as this model type was examined during the continuous variable prediction methods section. It can also be used for categorical variables. Some useful extractor functions like `print` and `importance` give some useful information about the model performance.

```
library(randomForest)

hv.RF <- randomForest(terron ~ AACN + Drainage.Index + Light.Insolation + TWI +
    Gamma.Total.Count, data = DSM_data[training, ], ntree = 500, mtry = 5)

# Output random forest model diognostics
print(hv.RF)

# output relative importance of each covariate
importance(hv.RF)
```

Three types of prediction outputs can be generated from Random Forest models, and are specified via the `type` parameter of the `predict` extractor functions. The different "types" are the `response` (predicted class), `prob` (class probabilities) or `vote` (vote count, which really just appears to return the probabilities).

```
# Prediction of classes
predict(hv.RF, type = "response", newdata = DSM_data[training, ])

# Class probabilities
predict(hv.RF, type = "prob", newdata = DSM_data[training, ])
```

From the diagnostics output of the `hv.C5` model the confusion matrix is automatically generated, except it was a different orientation to what we have been looking for previous examples. This confusion matrix was performed on what is called the OOB or out-of-bag data i.e. it validates the model/s dynamically with observations withheld from the model fit. So lets just evaluate the model as we have done for the previous models. For calibration:

```
C.pred.hv.RF <- predict(hv.RF, newdata = DSM_data[training, ])
goofcat(observed = DSM_data$terron[training], predicted = C.pred.hv.RF)
```

```
## $confusion_matrix
##      1 2  3  4  5  6   7   8  9 10 11 12
## 1  19 0  0  0  0  0   0   0  0  0  0  0
## 2   0 9  0  0  0  0   0   0  0  0  0  0
## 3   0 0 65  0  0  0   0   0  0  0  0  0
## 4   0 0  0 55  0  0   0   0  0  0  0  0
## 5   0 0  0  0 94  0   0   0  0  0  0  0
## 6   0 0  0  0  0 66   0   0  0  0  0  0
## 7   0 0  0  0  0  0 124   0  0  0  0  0
## 8   0 0  0  0  0  0   0 103  0  0  0  0
```

```
## 9   0 0  0  0  0  0   0   0 42  0  0  0
## 10  0 0  0  0  0  0   0   0  0 53  0  0
## 11  0 0  0  0  0  0   0   0  0  0 38  0
## 12  0 0  0  0  0  0   0   0  0  0  0 32
##
## $overall_accuracy
## [1] 100
##
## $producers_accuracy
##   1   2   3   4   5   6   7   8   9  10  11  12
## 100 100 100 100 100 100 100 100 100 100 100 100
##
## $users_accuracy
##   1   2   3   4   5   6   7   8   9  10  11  12
## 100 100 100 100 100 100 100 100 100 100 100 100
##
## $kappa
## [1] 1
```

It seems quite incredible that this particular model is indicating a 100% accuracy. Here it pays to look at the out-of-bag error of the `hv.RF` model for a better indication of the model goodness of fit. FOr the random holdback validation:

```
V.pred.hv.RF <- predict(hv.RF, newdata = DSM_data[-training, ])
goofcat(observed = DSM_data$terron[-training], predicted = V.pred.hv.RF)
```

```
## $confusion_matrix
##    1 2  3  4  5  6  7  8 9 10 11 12
## 1  2 2  1  2  0  0  0  0 1  0  0  0
## 2  4 0  2  0  0  0  0  0 1  1  0  0
## 3  0 0 11  1  0  0  7  0 2  0  2  1
## 4  2 2  4 10  0  0  1  3 6  0  0  0
## 5  0 0  0  0 29  8  2  7 5  2  0  1
## 6  0 0  0  0  2 18  0  6 0  1  0  0
## 7  0 0  8  5  0  0 31  8 3  4  3  5
## 8  0 0  0  0  4  0  8 19 3  1  2  1
## 9  0 0  1  0  0  0  0  0 2  0  0  0
## 10 0 0  2  0  2  0  1  3 1 12  1  0
## 11 0 0  5  2  1  0  0  1 0  1  2  0
## 12 0 0  0  0  0  0  1  0 0  3  0  5
##
## $overall_accuracy
## [1] 47
##
## $producers_accuracy
##  1  2  3  4  5  6  7  8  9 10 11 12
## 25  0 33 50 77 70 61 41  9 48 20 39
##
## $users_accuracy
```

```
##  1  2  3  4  5  6  7  8  9 10 11 12
## 25  0 46 36 54 67 47 50 67 55 17 56
##
## $kappa
## [1] 0.4015957
```

So based on the model validation, the Random Forest performs quite similarly to the other models that were used before, despite a *perfect* performance based on the diagnostics of the calibration model.

And finally the map that results from applying the `hv.C5` model to the covariate rasters in shown on Figure 3 .

```
# class prediction
map.RF.c <- predict(hunterCovariates, hv.RF,
filename = "hv_RF_class.tif", format = "GTiff",
overwrite = T, datatype = "INT2S")

map.RF.c <- as.factor(map.RF.c)
rat <- levels(map.RF.c)[[1]]
rat[["terron"]] <- c("HVT_001", "HVT_002", "HVT_003", "HVT_004", "HVT_005",
    "HVT_006", "HVT_007", "HVT_008", "HVT_009", "HVT_010", "HVT_011", "HVT_012")
levels(map.RF.c) <- rat

# plot
area_colors <- c("#FF0000", "#38A800", "#73DFFF", "#FFEBAF", "#A8A800", "#0070FF",
    "#FFA77F", "#7AF5CA", "#D7B09E", "#CCCCCC", "#B4D79E", "#FFFF00")
levelplot(map.RF.c, col.regions = area_colors, xlab = "", ylab = "")
```

References

Carre, F. and A. B. McBratney
2005. Digital terron mapping. *Geoderma*, 128:340–353.

Congalton, R. G.
1991. A review of assessing the accuracy of classifcations of remotely sensed data. *Remote Sensing of the Environment*, 37:35–46.

Kempen, B., D. J. Brus, G. B. M. Heuvelink, and J. J. Stoorvogel
2009. Updating the 1:50,000 dutch soil map using legacy soil data: A multinomial logistic regression approach. *Geoderma*, 151:311–326.

Landis, R. and G. G. Koch
1977. The measurment of observer agreement for categorical data. *Biometrics*, 33:159–174.

Malone, B. P., P. Hughes, A. B. McBratney, and B. Minsany
2014. A model for the identification of terrons in the lower hunter valley, australia. *Geoderma Regional*, 1:31–47.

McBratney, A. B. and I. O. A. Odeh

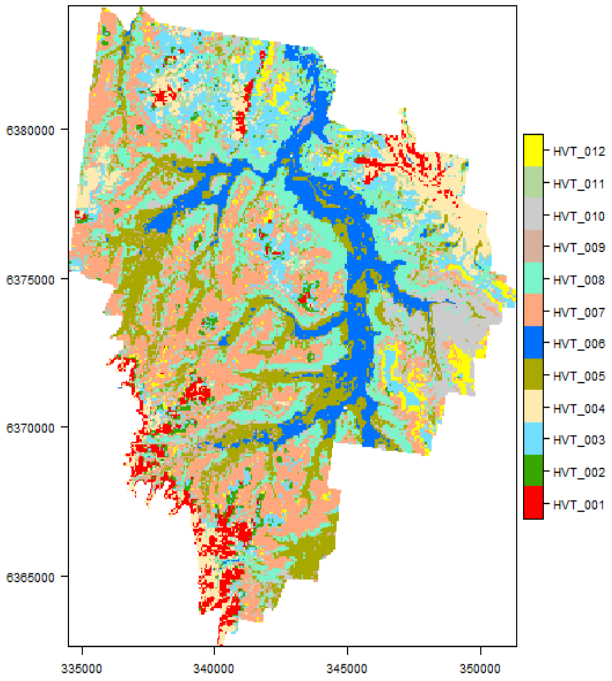

Figure 3: Hunter Valley Terron class map created using random forest model.

1997. Application of fuzzy sets in soil science: Fuzzy logic, fuzzy measurments and fuzzy decisions. *Geoderma*, 77:85–113.

Quinlan, J. R.
1993. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, San Mateo, California.

Vaudour, E., E. Costantini, G. V. Jones, and S. Mocali
2015. An overview of the recent approaches to terroir functional modelling, footprinting and zoning. *SOIL*, 1:287–312.