From VNIR to R: Scanning soils to rapidly predict a world of soil attributes

JP Schmidt

July 14, 2015

Bring in data. Mostly compiled from NRCS soil databases, but some collected on our own.

```
ll.soils.data<-read.table("sdata.csv",header = T, sep=",",na.strings=c("na","NA"))
```

Subset the data to include only profiles and horizons with percent C data.

```
C.data<- subset(ll.soils.data, !(is.na(C)))
```

Transform percent C to logit(percent C) and a column with this name. Because proportions are not normally distributed we convert using the logit - percent/1-percent. There is a built in function to logit transform in the 'car' package.

```
library(car)
logitC<-logit(C.data$C,percents=TRUE)
```

Now we need to deal with different horizons and soil depths to create a single "depth" column. A simple approach for the time being is to get the mean of the top and bottom depths for a horizon, take the log of it, and round only one significant digit. Since we will be using the depth value as a factor in later analyses, limiting the number of classes with very similar depths will increase the robustness of estimates.

```
depth<-round(log10(C.data$bottom.depth-((C.data$bottom.depth-C.data$top.depth)/2)),digits=1)
```

To capture distinct sites which can be expected to be similar to one another in soil properties, we find unique sets of coordinates and then create a site ID based on these.

```
# make a list of unique coordinate pairs
site<-unique(C.data[,1:2])
# get the number of unique sites
number.sites<-dim(site)[1]
# create a series from 1 to the number of unique sites
site.id<-seq(1,1068,1)
# bind site IDs to unique coordinates
site.ids<-cbind(site,site.id)
```

Now we need to put it all together.

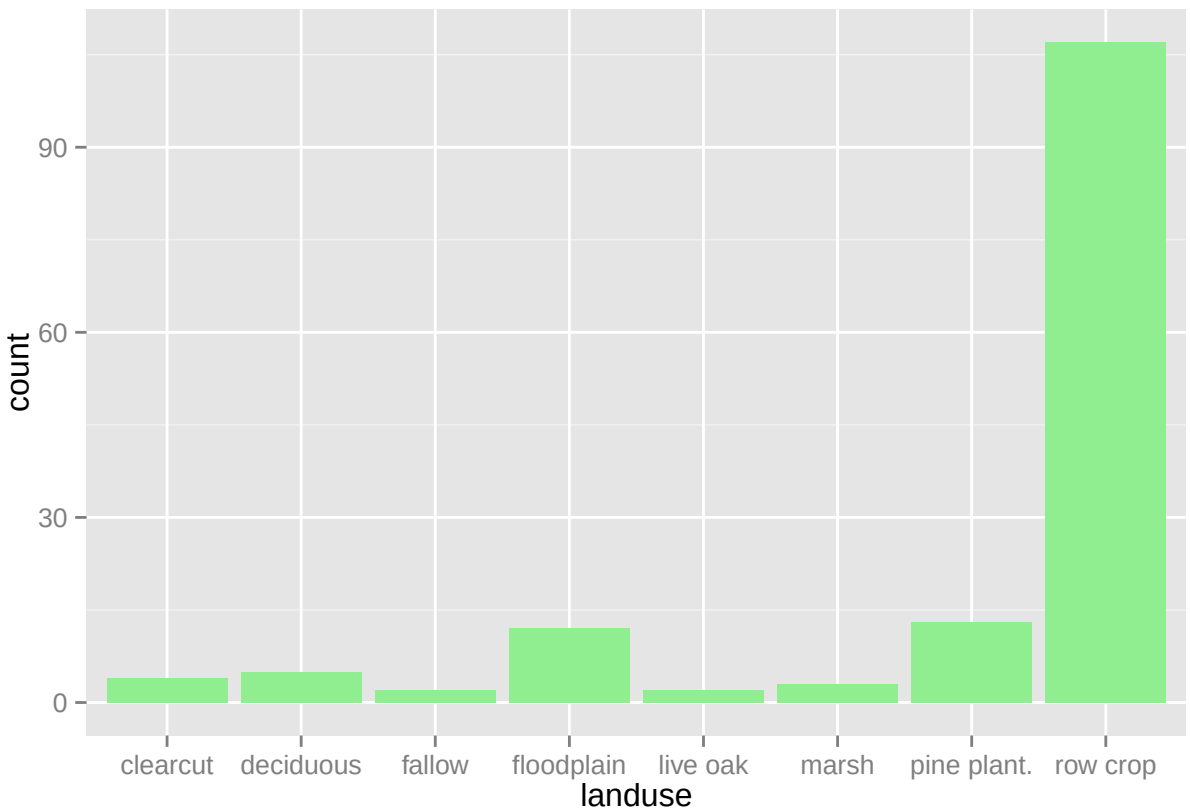
```
# combine soils data with "logitC"" and "depth""
C.data<-cbind(C.data,logitC,depth)
# join site ID to the data set using "lat" and "long" as join fields
C.data<-merge(C.data,site.ids,by=c("lat","long"))
```

To simplify analyses for the time-being,let's now subset the data to look at a single soil series, Pelham, which has good coverage for a variety of land use categories.

```
C.Pelham<-subset(C.data,soil.series=="Pelham")
C.Tifton<-subset(C.data,soil.series=="Tifton")
C.PT<-rbind(C.Pelham,C.Tifton)
```

But how many profiles do we have for each class. Let's get the set of unique profiles (eliminating duplicate for horizons within profiles) and make a histogram of profiles by land use class.

```
unique.profiles<- C.PT[!duplicated(C.PT[,3]),]
library(ggplot2)
lulc.hist <- ggplot(data.frame(unique.profiles),aes(x=landuse))
# counts
lulc.hist + geom_histogram(fill="lightgreen")
```



Since we have little coverage for classes other than "swamp/floodplain","row crop", and "pine plantation", let's subset the data to include only those classes.

```
C.PT<-subset(C.PT,landuse %in% c("row crop","pine plant.","floodplain"))
```

Now let's do a simple linear regression of logitC on depth and landuse. (We go in to detail a little later about how to assemble land use data.)

```
C.PT.lm<-lm(logitC~as.factor(depth) + landuse,data=C.PT)
summary(C.PT.lm)$r.squared
```

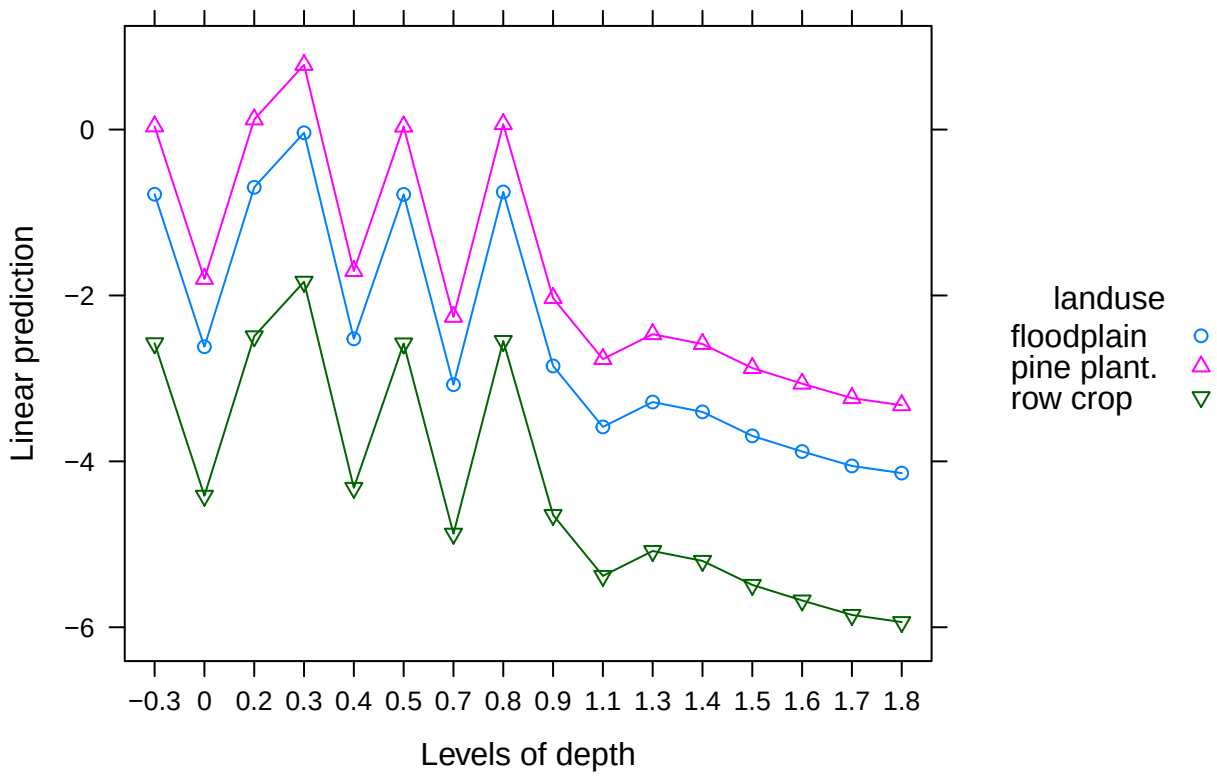
```
## [1] 0.8245931
```

Looking over these results, we see a really good model fit (r2 = 0.82). This is of course misleading since we have not controlled for the spatial autocorrelation of samples.But this is a good starting point. We can compare these results to those from subsequent models that incorporate space.Let's look at the means by land use and soil depth for the Pelham percent C data graphically using the 'lsmeans' package.

```
library(lsmeans)
```

```
## Loading required package: estimability
```

```
lsmip(C.PT.lm, landuse~depth)
```



Notice that C content is highest in pine plantations followed by clearcuts and then floodplains, and that, although row crop is much lower, live oak shows the lowest values. Are there significant differences between land use classes? We can plot the contrasts to see.

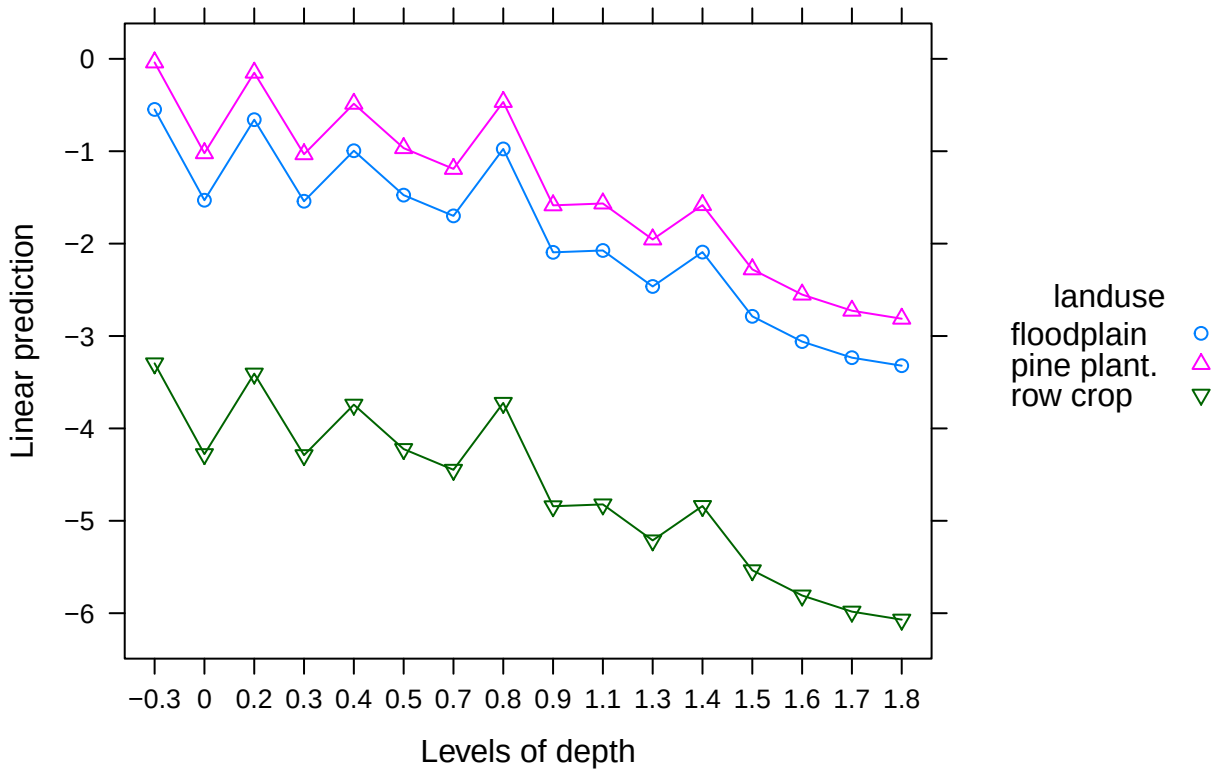
```
C.PT.lm.means<-lsmeans(C.PT.lm, "landuse", by="depth")
# plot command below generates confidence interval
#plot(C.PT.lm.means, comparisons=TRUE, alpha= .10)
```

We see from the plot of means and confidence intervals by depth, that the means for each land use are signficant different from one another. But, of course, we haven't controlled for spatial autocorrelation. Let's now run a mixed model that treats site and profile as random effects and depth and land use as fixed effects. To do this, we need to laod the 'nlme' package.

```
library(nlme)
C.PT.lmm<-lme(logitC~landuse + as.factor(depth), random = ~1 | site.id/profile.id,data=C.PT)
```

Let's plot the mixed model results for logitC by landuse across soil depth.

```r
lsmip(C.PT.lmm, landuse~depth)
```



Notice the graph has changed quite a bit. Let's check whether contrasts between other land uses are still signficant.

```r
C.PT.lmm.means<-lsmeans(C.PT.lmm, "landuse", by="depth")
# plot command below generates confidence interval graphs
#plot(C.PT.lmm.means, comparisons=TRUE, alpha= .10)
```

Once we have corrected for spatial autocorrelation, we see that pine plantation and floodplains do not have significantly higher C content, but that C content is still quite lower in row crop management.If we want to see the values in the graph tabulated, we can simply type:

```r
C.PT.lmm.means
```

```
## depth = -0.3:
##  landuse          lsmean        SE df   lower.CL    upper.CL
##  floodplain  -0.54777364 0.4733769 94 -1.4876746  0.39212735
##  pine plant. -0.03919753 0.4094089 92 -0.8523189  0.77392388
##  row crop    -3.29612560 0.4837247 92 -4.2568446 -2.33540665
##
## depth =  0.0:
##  landuse          lsmean        SE df   lower.CL    upper.CL
##  floodplain  -1.53025833 0.2694455 94 -2.0652486 -0.99526806
##  pine plant. -1.02168221 0.3108455 92 -1.6390483 -0.40431612
##  row crop    -4.27861028 0.1493242 92 -4.5751811 -3.98203944
##
## depth =  0.2:
```

```
##  landuse          lsmean        SE df   lower.CL    upper.CL
##  floodplain  -0.65837920 0.3388587 94 -1.3311911  0.01443274
##  pine plant. -0.14980308 0.2948726 92 -0.7354454  0.43583926
##  row crop    -3.40673116 0.3535889 92 -4.1089892 -2.70447312
##
## depth =  0.3:
##  landuse          lsmean        SE df   lower.CL    upper.CL
##  floodplain  -1.54122235 0.3473138 94 -2.2308220 -0.85162268
##  pine plant. -1.03264623 0.4160249 92 -1.8589077 -0.20638477
##  row crop    -4.28957430 0.3117016 92 -4.9086406 -3.67050800
##
## depth =  0.4:
##  landuse          lsmean        SE df   lower.CL    upper.CL
##  floodplain  -0.99473233 0.2775398 94 -1.5457941 -0.44367057
##  pine plant. -0.48615622 0.3088543 92 -1.0995676  0.12725514
##  row crop    -3.74308429 0.1830850 92 -4.1067070 -3.37946162
##
## depth =  0.5:
##  landuse          lsmean        SE df   lower.CL    upper.CL
##  floodplain  -1.47513709 0.5075935 94 -2.4829759 -0.46729824
##  pine plant. -0.96656097 0.5093469 92 -1.9781678  0.04504586
##  row crop    -4.22348904 0.4622559 92 -5.1415691 -3.30540897
##
## depth =  0.7:
##  landuse          lsmean        SE df   lower.CL    upper.CL
##  floodplain  -1.69968353 0.2806503 94 -2.2569213 -1.14244578
##  pine plant. -1.19110741 0.3215477 92 -1.8297290 -0.55248587
##  row crop    -4.44803549 0.1637739 92 -4.7733045 -4.12276645
##
## depth =  0.8:
##  landuse          lsmean        SE df   lower.CL    upper.CL
##  floodplain  -0.97517623 0.3887606 94 -1.7470695 -0.20328296
##  pine plant. -0.46660011 0.3836024 92 -1.2284676  0.29526734
##  row crop    -3.72352818 0.3771440 92 -4.4725687 -2.97448762
##
## depth =  0.9:
##  landuse          lsmean        SE df   lower.CL    upper.CL
##  floodplain  -2.09434523 0.2647848 94 -2.6200816 -1.56860890
##  pine plant. -1.58576911 0.3163208 92 -2.2140095 -0.95752868
##  row crop    -4.84269719 0.1027456 92 -5.0467588 -4.63863557
##
## depth =  1.1:
##  landuse          lsmean        SE df   lower.CL    upper.CL
##  floodplain  -2.07417320 0.2666459 94 -2.6036049 -1.54474147
##  pine plant. -1.56559708 0.3098870 92 -2.1810594 -0.95013480
##  row crop    -4.82252515 0.1425148 92 -5.1055718 -4.53947853
##
## depth =  1.3:
##  landuse          lsmean        SE df   lower.CL    upper.CL
##  floodplain  -2.46372603 0.2661972 94 -2.9922668 -1.93518523
##  pine plant. -1.95514991 0.3167557 92 -2.5842541 -1.32604571
##  row crop    -5.21207799 0.1027931 92 -5.4162340 -5.00792195
##
## depth =  1.4:
```

```
##  landuse          lsmean        SE df   lower.CL    upper.CL
##  floodplain  -2.09164998 0.3185074 94 -2.7240539 -1.45924606
##  pine plant. -1.58307386 0.3551555 92 -2.2884433 -0.87770447
##  row crop    -4.84000194 0.2223046 92 -5.2815181 -4.39848582
##
## depth =  1.5:
##  landuse          lsmean        SE df   lower.CL    upper.CL
##  floodplain  -2.78703286 0.2636916 94 -3.3105987 -2.26346701
##  pine plant. -2.27845674 0.3142017 92 -2.9024885 -1.65442499
##  row crop    -5.53538482 0.1008208 92 -5.7356237 -5.33514592
##
## depth =  1.6:
##  landuse          lsmean        SE df   lower.CL    upper.CL
##  floodplain  -3.06008877 0.2661972 94 -3.5886296 -2.53154797
##  pine plant. -2.55151266 0.3167557 92 -3.1806169 -1.92240845
##  row crop    -5.80844073 0.1027931 92 -6.0125968 -5.60428470
##
## depth =  1.7:
##  landuse          lsmean        SE df   lower.CL    upper.CL
##  floodplain  -3.23446977 0.2661972 94 -3.7630106 -2.70592896
##  pine plant. -2.72589365 0.3167557 92 -3.3549979 -2.09678944
##  row crop    -5.98282172 0.1027931 92 -6.1869778 -5.77866569
##
## depth =  1.8:
##  landuse          lsmean        SE df   lower.CL    upper.CL
##  floodplain  -3.32118262 0.2662390 94 -3.8498065 -2.79255876
##  pine plant. -2.81260650 0.3167912 92 -3.4417811 -2.18343187
##  row crop    -6.06953457 0.1029125 92 -6.2739277 -5.86514149
##
## Confidence level used: 0.95
```

Since these values are logit-transformed, we can use an anti-logit function to get an actual range. In this case for row crop in the -0.3 (0.5 cm) depth.

```r
antilogit <- function(x) { exp(x) / (1 + exp(x) ) }
mean<-print(antilogit(-6.06953457))
```

```
## [1] 0.002306915
```

```r
lower.CL<-print(antilogit(-6.2739277))
```

```
## [1] 0.001881265
```

```r
upper.CL<-print(antilogit(-5.86514149))
```

```
## [1] 0.002828598
```

We have shown some patterns so far, but we could really use more data on land uses other than row crop. So, using the 'soilDB' package we'll download additional data from the Rapid Assessment of U.S. Soil Carbon (RaCA), which includes many samples that were scanned by VNIR but lack chemometric values such as percent C.

```r
library(soilDB)
```

```
## Loading required package: aqp
## This is aqp 1.8-6
```

```r
pelham<- fetchRaCA(series='pelham',get.vnir=TRUE)
```

```
## Carbon concentration and stock values are probably wrong, or at least suspect. USE WITH CAUTION.
## spectra are large, download may take some time...
## 22 RaCA sites loaded (4.18 Mb transferred)
```

```r
tifton<- fetchRaCA(series='tifton',get.vnir=TRUE)
```

```
## Carbon concentration and stock values are probably wrong, or at least suspect. USE WITH CAUTION.
## spectra are large, download may take some time...
## 10 RaCA sites loaded (2.03 Mb transferred)
```

For each series, we need to pull out several tables of data. First, the site data on land use and vegetation.

```r
raca.site<-rbind(pelham$pedon@site,tifton$pedon@site)
```

Second, data on each horizon for each profile.

```r
raca.horizons<-rbind(pelham$pedons@horizons,tifton$pedons@horizons)
```

Third, sample data which links samples to horizons and spectra.

```r
raca.sample<-rbind(pelham$sample,tifton$sample)
```

Finally, VNIR spectra which we will use to augment our data set.

```r
raca.spectra<-rbind(tifton$spectra,pelham$spectra)
```

Now we need to join them together in order to calibrate the spectra to percent C.

```r
# add sample_id column to spectra
sample_id<-as.numeric(rownames(raca.spectra))
raca.spectra<-cbind(sample_id,raca.spectra)
# join site, horizon, and spectra tables
sh <- merge(raca.sample, raca.horizons, by=c("rcapid","hzname"))
ssh<-merge(sh,raca.site, by="rcapid")
sh.spec <- merge(ssh, raca.spectra, by="sample_id")
# bring in chemo data
chemo.data<-read.table("raca.analyses.sept18.csv",header = T, sep=",",na.strings=c("na","NA"))
PT.spec.chemo<-merge(chemo.data,sh.spec,by="sample_id",all.y=TRUE)
```

The above methods are how we acquire the VNIR spectra. Since the landuse categories within the RaCA data are broad. We will extract values from the National Land Cover Dataset (NLCD) and crosswalk them to existing classes from other data sources to assemble the long leaf soils data set. First we need to bring in the NLCD raster data downloaded from (http://www.mrlc.gov/nlcd11_data.php). This requires loading the 'raster' package in R.

```r
library(raster)
```

```
## Loading required package: sp
##
## Attaching package: 'raster'
##
## The following objects are masked from 'package:aqp':
##
##     metadata, metadata<-
##
## The following object is masked from 'package:nlme':
##
##     getData
```

```r
nlcd<-raster("nlcd_2011_landcover_2011_edition_2014_10_10.img")
```

Now we need to turn the C.PTa table into a spatial points data fram using the x and y coordinates included as columns of the data. We must also set the coordinate projection - in this case unprojected lat-long data, then project it to Albers Equal Area to match the NLCD. All this requires loading several more R GIS packages.

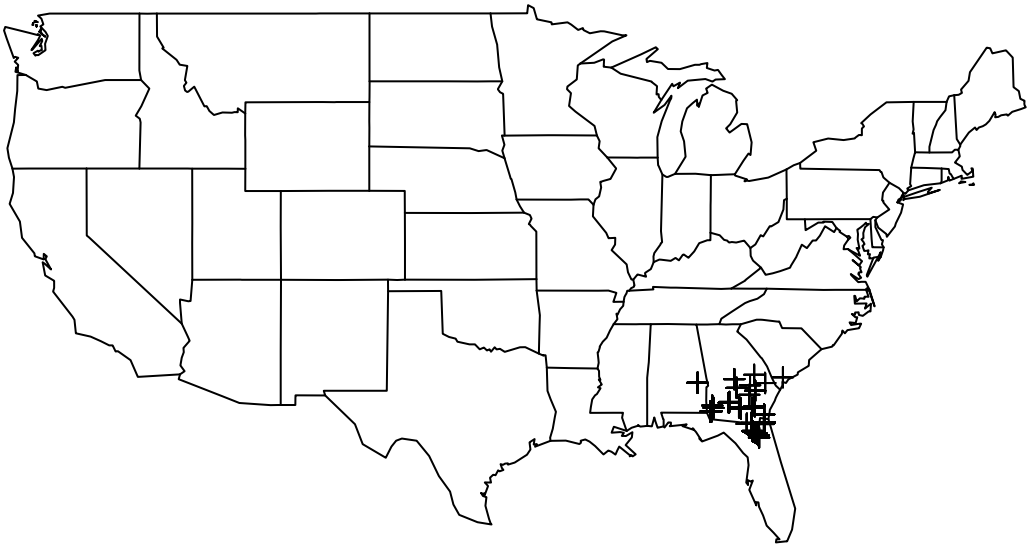
```r
library(sp)
library(rgdal)
```

```
## rgdal: version: 1.0-4, (SVN revision 548)
##  Geospatial Data Abstraction Library extensions to R successfully loaded
##  Loaded GDAL runtime: GDAL 1.11.2, released 2015/02/10
##  Path to GDAL shared files: /Library/Frameworks/R.framework/Versions/3.2/Resources/library/rgdal/gdal
##  Loaded PROJ.4 runtime: Rel. 4.9.1, 04 March 2015, [PJ_VERSION: 491]
##  Path to PROJ.4 shared files: /Library/Frameworks/R.framework/Versions/3.2/Resources/library/rgdal/pr
##  Linking to sp version: 1.1-1
```

```r
coordinates(PT.spec.chemo)<-~x + y
proj4string(PT.spec.chemo)=CRS("+proj=longlat +ellps=WGS84 +datum=WGS84") # set it to lat-long
```

Now let's plot the points to make sure they're where we think they are.

```r
library(maps)
map('state')
plot(PT.spec.chemo,add=TRUE)
```



Since the points are in lat-long and the NLCD in Alber-Equal Area, we need to project the points to that coordinate system.

```
PT.spec.chemo = spTransform(PT.spec.chemo,CRS("+proj=aea +lat_1=29.5 +lat_2=45.5 +lat_0=23 +lon_0=-96 +
```

Now let's extract the landcover values to the Pelham-Tifton points.

```
lulc.nlcd<-extract(nlcd,PT.spec.chemo)
```

Now let's see if it worked.

```
head(as.data.frame(lulc.nlcd))
```

```
##   lulc.nlcd
## 1        90
## 2        90
## 3        90
## 4        90
## 5        90
## 6        90
```

We now need to join these values to the data.

```
PT.spec.chemo.nlcd<-cbind(lulc.nlcd,PT.spec.chemo)
```

Of course these numeric codes will need to be crosswalked to whatever existing LULC classes you may be using. We have already done that for this data so we'll skip ahead.

```
dim(PT.spec.chemo.nlcd)
```

```
## [1] 350   2
```

Bringing back in this data simplified to have just percent C values, profile id and spectra. We can now transform the data and randomly split profiles.

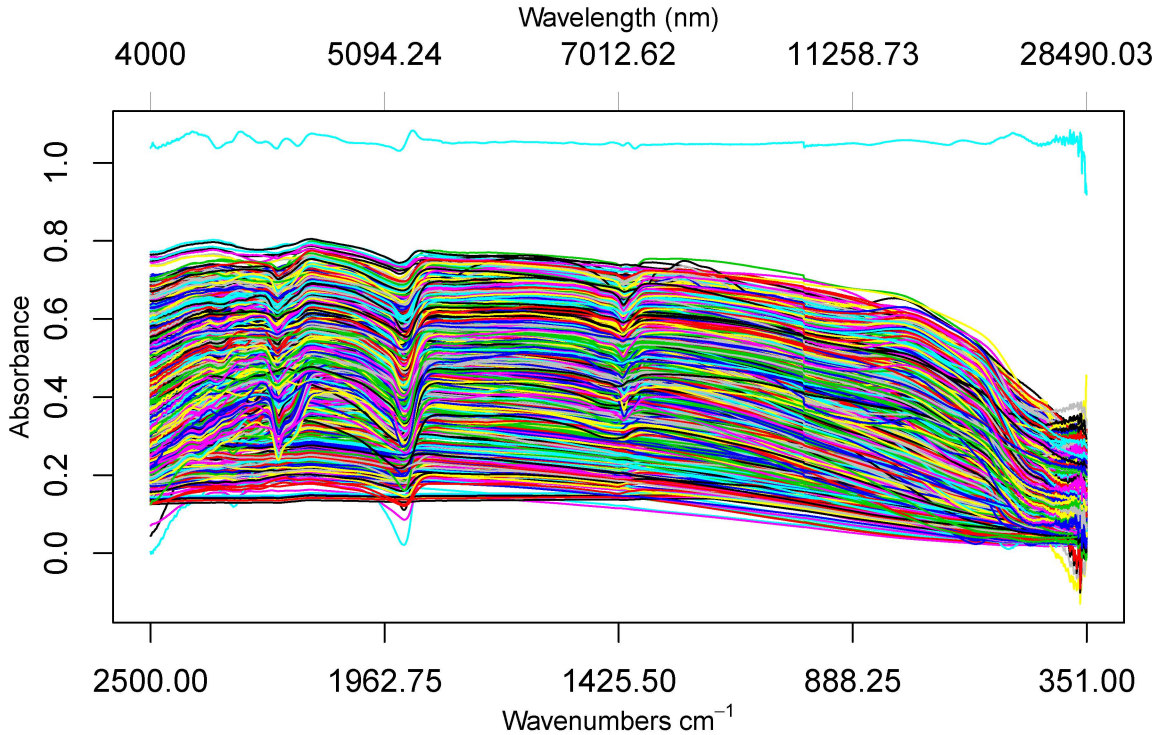
```r
ll.spec.chemo<-read.table("ll.spec.chemo.csv",header = T, sep=",",na.strings=c("na","NA"))
```

First let's separate and plot the spectra using the plot.Spectra function adapted from the 'soil.spec' package.

```r
spectra<-ll.spec.chemo[,3:2153]

plot.Spectra<-function (x, ...)
{
  if (nrow(x) > 0) {
    k <- ncol(x)
    wavenumber <- as.numeric(sapply(names(x[, -1]), function(obj) {
      gsub("[^0-9.]", "", obj)
    }))
    x <- as.matrix(x[, -1])
    nano <- round(1e+07/wavenumber, 2)
    wx <- summary(wavenumber)[-4]
    nx <- round(1e+07/wx, 2)
    xaxp <- c(min(wx), max(wx), length(wx) - 1)
    par(mfrow = c(1, 1))
    plot(wavenumber, x[1, ], pch = "", frame = FALSE, xlab = "",
         ylab = "", xlim = c(max(wavenumber), min(wavenumber)),
         ylim = c(min(na.omit(x)), max(na.omit(x))), xaxp = xaxp,
         ...)
    axis(3, wx, nx, lwd = 0, tick = TRUE, lwd.ticks = 0.1,
         col.ticks = "black")
    for (i in 1:nrow(x)) {
      lines(wavenumber, x[i, ], col = i)
    }
    box()
    mtext(expression("Wavenumbers cm"^-1), side = 1, line = 2,
          cex = 0.8)
    mtext("Wavelength (nm)", side = 3, line = 2, cex = 0.8)
    mtext("Absorbance", side = 2, line = 2, cex = 0.8)
  }
  else {
    stop("Empty object")
  }
}

plot.Spectra(spectra)
```

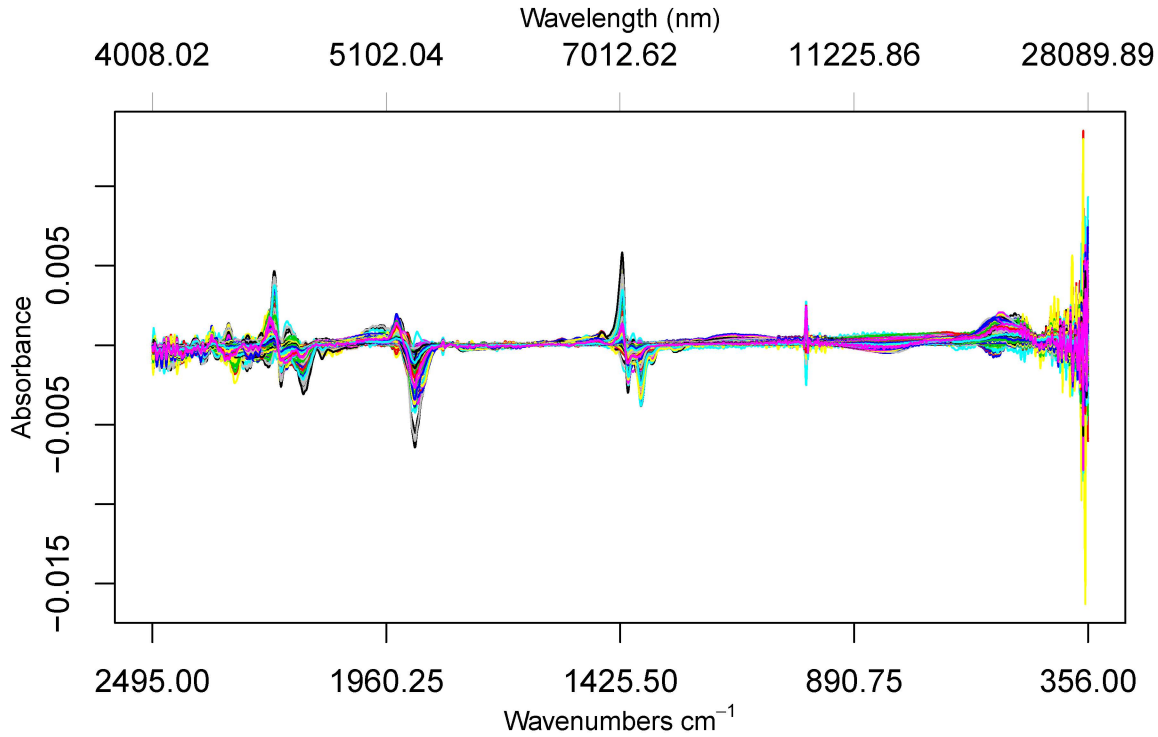


Now let's transform the spectra using the Savitzky-Golay transformation with default values for arguments, using the 'prospectr' package in R, and plot them again.

```
library(prospectr)
```

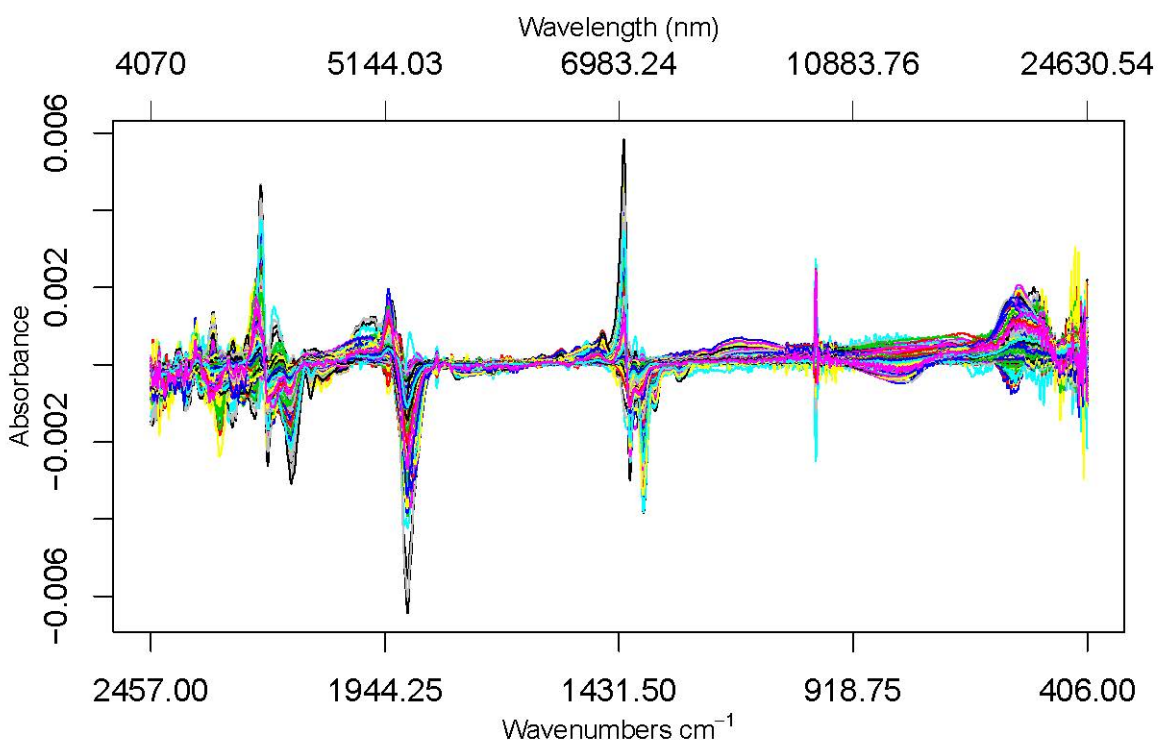
```
## Loading required package: RcppArmadillo
##
## Attaching package: 'prospectr'
##
## The following object is masked from 'package:raster':
##
##     resample
```

```
spectra.sg<-as.data.frame(savitzkyGolay(X = spectra,1,3,11,delta.wav=2))
plot.Spectra(spectra.sg)
```



Notice how the tranformation has removed a lot of noise. Notice also that the ranges 350-380 nm and 2460-2500 nm are regions of low signal to noise, and so we may want to remove them as below.

```
spectra.sg.nr<-as.data.frame(savitzkyGolay(X = spectra[,51:2113],1,3,11,delta.wav=2))
plot.Spectra(spectra.sg.nr)
```



To assess the accuracy of a calibration, sample data should be separated into "training" and "test" sets.

There are several options when doing this. Typically with soils, samples collected at varying depths within the same profile should not be separated between training and test sets as this leads to misleadingly high validation results. At the same time, calibration accuracy is improved when training samples capture the breadth of spectral variation within the data for which the Kennard-Stone algorithm (Kennard and Stone 1969) is often used. Here, we use the implementation of Kennard-Stone in the 'prospectr' package."k" is set to 1700 to split the data set into 60% training and 20% test sets, and "group" is set equal to 'profile' to keep depth increments within the same profile together.

```
trans.data<-cbind(ll.spec.chemo[,1:2],spectra.sg.nr) # dropping the first 30 and last 40 spectra
breaks<-kenStone(spectra,k=1200,metric="euclid",group=ll.spec.chemo$id)
training <- trans.data[breaks$test,]
test <- trans.data[breaks$model,]
```

Now let's pull out the samples that have both spectra and measured values of percent total C, and see how many complete samples we have in training vs. test sets.

```
training<-subset(training[,2:2055],training$logitC !="NA")
dim(training)
```

```
## [1]  194 2054
```

```
test<-subset(test[,2:2055],test$logitC !="NA")
dim(test)
```

```
## [1]   81 2054
```

Now we will use the Partial Least Squares Regression Algorithm (PLSR) to calibrate spectral samples to corresponding percent total C values. As a rule of thumb, it's best to keep the maximum number of components low. Otherwise, the model will tend to "overfit", capturing the variation in the training set well, but not capturing the more general patterns.

```
library(pls)
```

```
##
## Attaching package: 'pls'
##
## The following object is masked from 'package:stats':
##
##     loadings
```

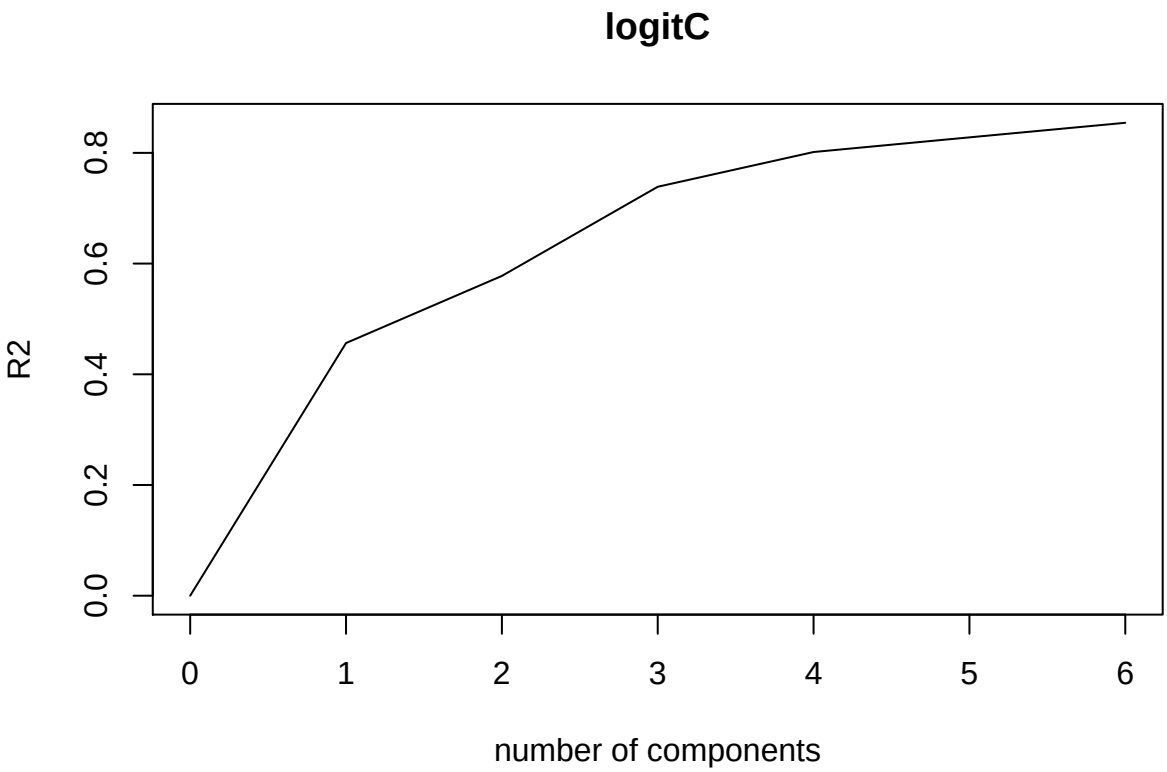
```
Cplsr<-plsr(logitC~.,ncomp=6,data=training, validation = "CV")
summary(Cplsr)
```

```
## Data:    X dimension: 194 2053
##  Y dimension: 194 1
## Fit method: kernelpls
## Number of components considered: 6
##
## VALIDATION: RMSEP
## Cross-validated using 10 random segments.
##        (Intercept)  1 comps  2 comps  3 comps  4 comps  5 comps  6 comps
```

```
## CV          0.8338   0.6245   0.5585   0.4446   0.3920   0.3732   0.3519
## adjCV       0.8338   0.6239   0.5578   0.4437   0.3906   0.3699   0.3466
##
## TRAINING: % variance explained
##         1 comps  2 comps  3 comps  4 comps  5 comps  6 comps
## X         36.60    80.16    85.45    88.82    90.55    91.90
## logitC    45.64    57.77    73.88    80.16    82.80    85.43
```

Notice in the summary that 6 components seems to give the lowest Root Mean Squared Error of Prediction (RMSEP) on training. Now let's plot the more familiar R2 by number of components.

```
plot(Cplsr, "val",val.type = "R2", estimate = "train")
```



Now let's predict on the hold out test data using 5 components.

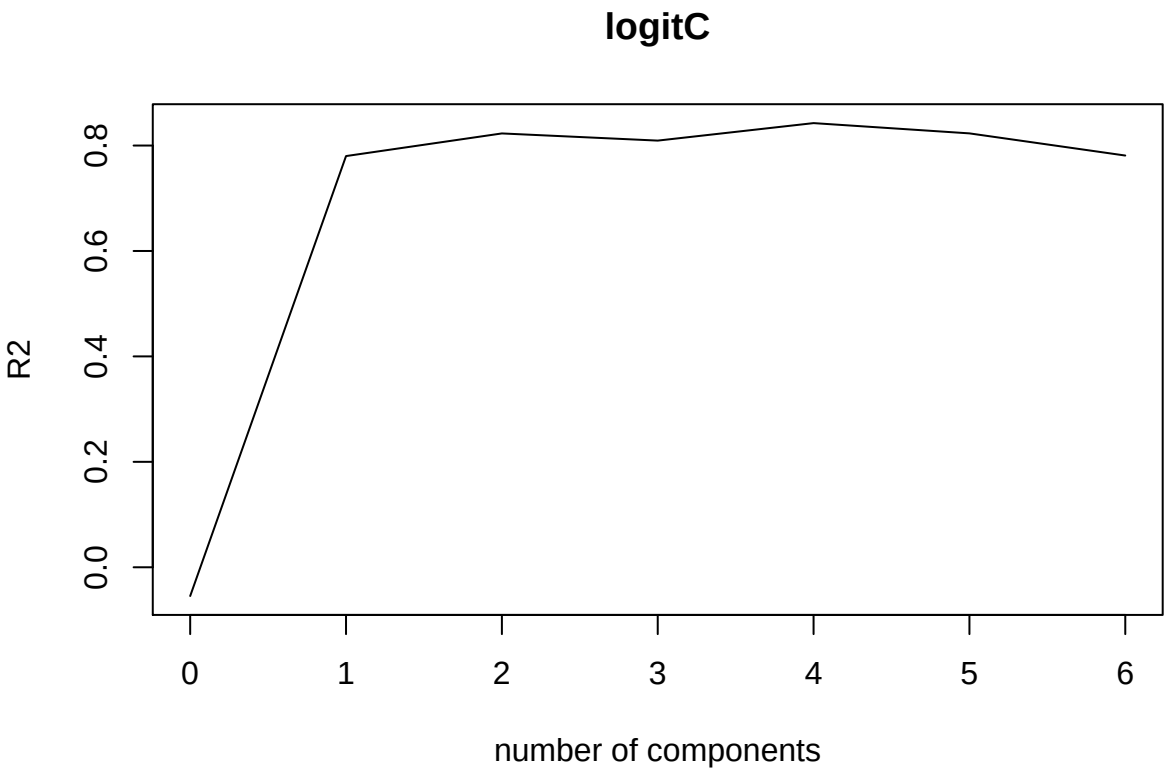
```
pred.test<-predict(Cplsr, test, ncomp = 3, type ="response")
```

Now let's look at error in test set predictions using the same model and looking at RMSEP and R2 plots.

```
RMSEP(Cplsr,newdata=test)
```

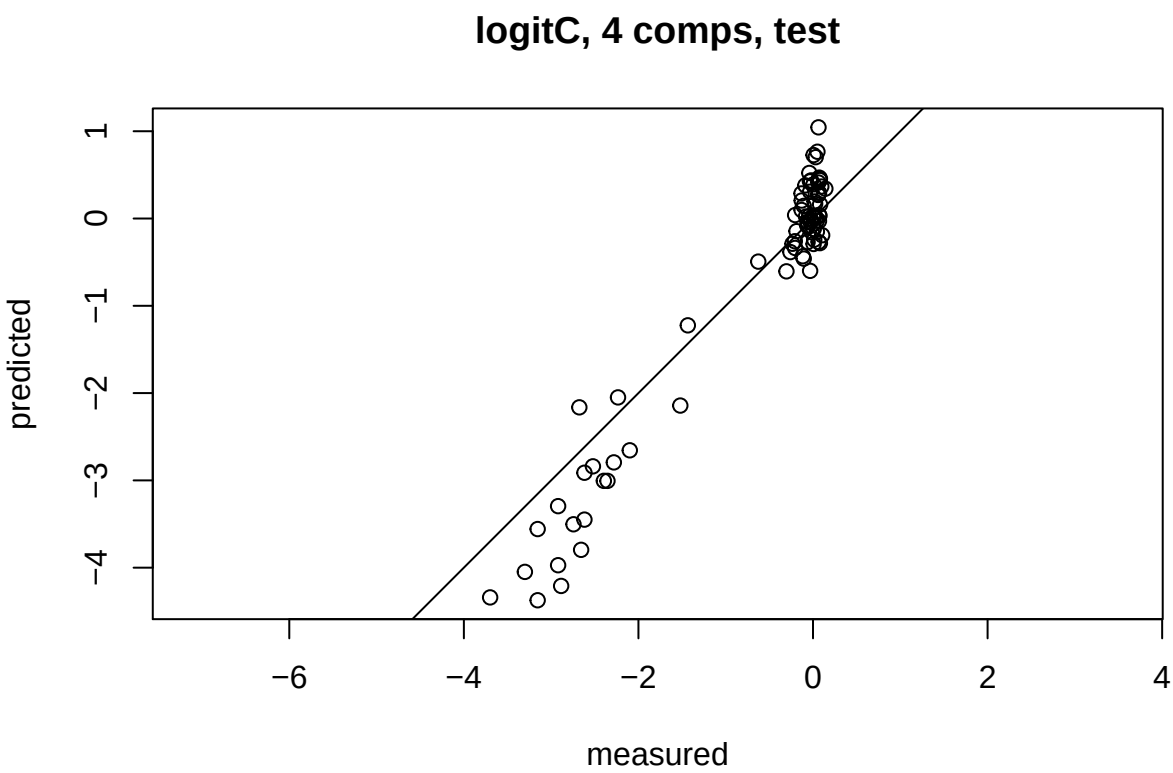
```
## (Intercept)      1 comps      2 comps      3 comps      4 comps
##      1.1785       0.5383       0.4829       0.5011       0.4554
##     5 comps      6 comps
##      0.4827       0.5370
```

```r
plot(Cplsr, "val",val.type = "R2", estimate = "test",newdata=test)
```



Notice that 4 components seem to yield the lowest error. Now let's plot measured vs. predicted values for training and test data.

```r
predplot(Cplsr,ncomp=4,newdata=test,asp=1,line=TRUE)
```



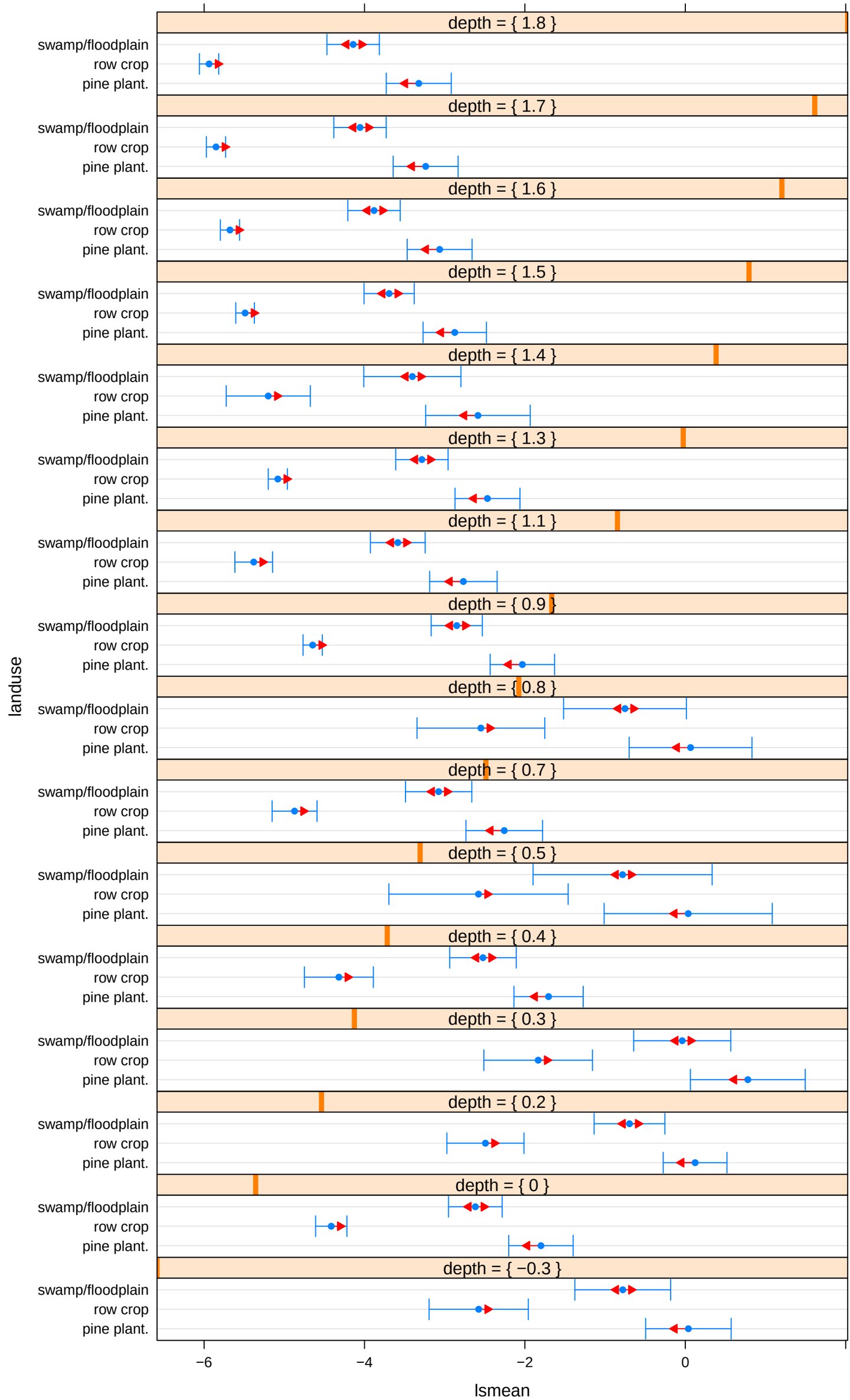
Now let's predict on the hold out test data using 4 components.

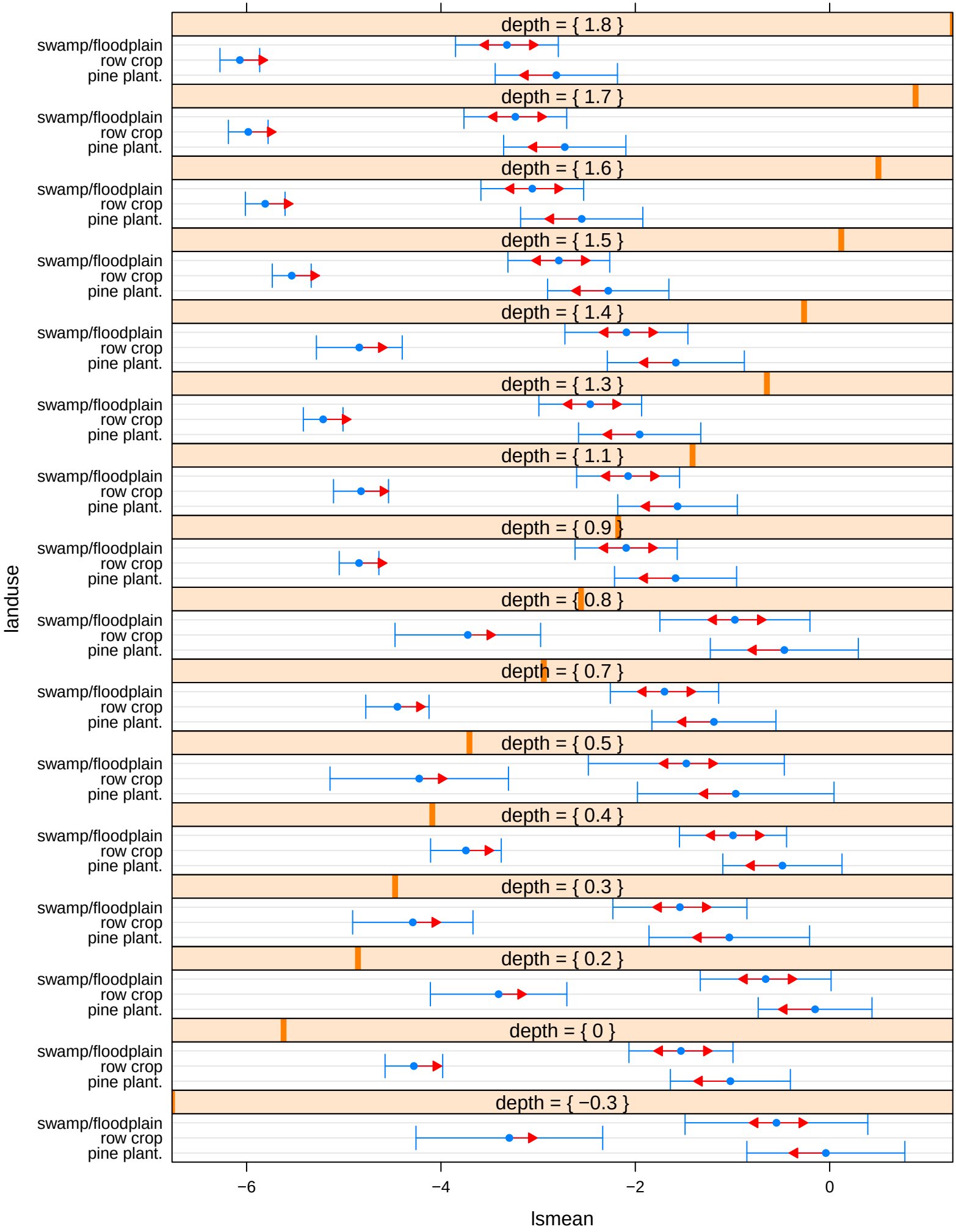
```
predicted.all<-predict(Cplsr, spectra.sg.nr, ncomp = 4, type ="response")
```

This is a cursory introduction to VNIR calibration. In a future webinar, I would like to show more about tuning plsr or other models and bring that data back into regression models to test for differences in soil quality measures by land use.

References

Kennard, R. W., & Stone, L. A. (1969). Computer aided design of experiments. Technometrics, 11(1), 137-148.


depth = { 1.8 }
depth = { 1.7 }
depth = { 1.6 }
depth = { 1.5 }
depth = { 1.4 }
depth = { 1.3 }
depth = { 1.1 }
depth = { 0.9 }
depth = { 0.8 }
depth = { 0.7 }
depth = { 0.5 }
depth = { 0.4 }
depth = { 0.3 }
depth = { 0.2 }
depth = { 0 }
depth = { −0.3 }
swamp/floodplain
row crop
pine plant.
landuse
−6
−4
−2
0
lsmean


depth = { 1.8 }
depth = { 1.7 }
depth = { 1.6 }
depth = { 1.5 }
depth = { 1.4 }
depth = { 1.3 }
depth = { 1.1 }
depth = { 0.9 }
depth = { 0.8 }
depth = { 0.7 }
depth = { 0.5 }
depth = { 0.4 }
depth = { 0.3 }
depth = { 0.2 }
depth = { 0 }
depth = { −0.3 }
swamp/floodplain
row crop
pine plant.
landuse
−6
−4
−2
0
lsmean