USGS LVM

Nicholas Kruskamp

February 15, 2018

## USGS CDI lidar processing overview

This document provides an overview of the basic workflow used to create standardized metrics for the UGSS CDI project “”.

To import data, a path to a LAS or LAZ lidar file must be provided along with a coordinate reference system in standard proj4 format. Additional options are provided to remove values about a certain height that might be considred noise, and to limit the classification of points to exclude possible overlap, water, and noise. NOTE: points must already be height normalized. Reading las data relies on the R package rlas which provides access to lastools and laszip from R. The lidar data is transforms into a spatial points dataframe, providing all the functionality of the SP package for spatial data processing.

USGSlvm::readLidarData

## function (x, inputCRS, maxHAG = 70, returnAll = FALSE)   
## {  
## x <- rlas::readlasdata(x, F, T, T, F, F, T, T, F, F, F)  
## if (returnAll == F) {  
## x <- x[x$Classification %in% c(1, 2, 3, 4, 5, 8, 19),   
## ]  
## }  
## x <- x[x$Z < maxHAG, ]  
## sp::coordinates(x) <- ~X + Y  
## sp::proj4string(x) <- sp::CRS(inputCRS)  
## return(x)  
## }  
## <environment: namespace:USGSlvm>

#example usage  
las\_data <- USGSlvm::readLidarData("C:/Users/nfkruska/Documents/projects/SHEN/va\_shen\_2011/HAG/SHENVAL2011\_744e\_4308n\_HAG.laz", "+proj=utm +zone=17 +datum=NAD83 +units=m +no\_defs")  
  
# since data is expected to be normalized, the z column is changed to reflect  
# normalized heights. Code is written to look for this column name. Future   
# development may include a function to normalize points and provide a new  
# column of "Z\_agl".  
names(las\_data)[1] <- "Z\_agl"  
  
# In addition, points must be classified by height above ground level  
las\_data <- USGSlvm::classifyByHeight(las\_data)  
  
las\_data

## class : SpatialPointsDataFrame   
## features : 2310528   
## extent : 744000, 745000, 4308000, 4309000 (xmin, xmax, ymin, ymax)  
## coord. ref. : +proj=utm +zone=17 +datum=NAD83 +units=m +no\_defs +ellps=GRS80 +towgs84=0,0,0   
## variables : 6  
## names : Z\_agl, gpstime, ReturnNumber, NumberOfReturns, Classification, ScanAngle   
## min values : -27.599, -16890635.3680003, 1, 1, 1, -9   
## max values : 47.239, -16888686.4226961, 4, 4, 110, 10

## Lidar Metrics

A number of predefined lidar metrics are available from the USGSlvm package. While the preprocessed data are served at resolutions of 10 and 25 meters, these functions allow the users to specify any resolution. It is hightly recommended that the resolution provided be an even multiple of the lidar points extent to avoid irregularities in pixels or edge effects of resulting rasters.

resolution <- 25  
USGSlvm::calcPointStatistics

## function (x, resolution = 30, pointClasses = c(100:200))   
## {  
## r <- raster::raster(x, resolution = resolution)  
## x <- x[x$Classification %in% pointClasses, ]  
## qMean <- function(x, ...) {  
## sqrt(mean(x^2))  
## }  
## minRast <- raster::rasterize(x, r, field = x$Z\_agl, fun = min)  
## maxRast <- raster::rasterize(x, r, field = x$Z\_agl, fun = max)  
## meanRast <- raster::rasterize(x, r, field = x$Z\_agl, fun = mean)  
## sdRast <- raster::rasterize(x, r, field = x$Z\_agl, fun = sd)  
## skewRast <- raster::rasterize(x, r, field = x$Z\_agl, fun = moments::skewness)  
## kurtRast <- raster::rasterize(x, r, field = x$Z\_agl, fun = moments::kurtosis)  
## qMeanRast <- raster::rasterize(x, r, field = x$Z\_agl, fun = qMean)  
## s <- raster::stack(minRast, maxRast, meanRast, sdRast, skewRast,   
## kurtRast, qMeanRast)  
## names(s) <- c("hmin", "hmax", "havg", "hstd", "hske", "hkur",   
## "hqav")  
## return(s)  
## }  
## <environment: namespace:USGSlvm>

stats <- USGSlvm::calcPointStatistics(las\_data, resolution)  
  
USGSlvm::calcVertDistRatio

## function (x, resolution = 30, pointClasses = c(100:200))   
## {  
## percLayer <- USGSlvm::calcHeightPercentiles(x, resolution = resolution,   
## percentiles = c(0.5, 0.98, 1))  
## x <- x[x$Classification %in% pointClasses, ]  
## vdr98 <- (percLayer$X98 - percLayer$X50)/percLayer$X98  
## vdr100 <- (percLayer$X100 - percLayer$X50)/percLayer$X100  
## s <- raster::stack(vdr98, vdr100)  
## names(s) <- c("vdr98", "vdr100")  
## return(s)  
## }  
## <environment: namespace:USGSlvm>

vdr <- USGSlvm::calcVertDistRatio(las\_data, resolution)  
  
USGSlvm::calcCanopyCover

## function (x, resolution = 30, pointClasses = c(100:200))   
## {  
## r <- raster::raster(x, resolution = resolution)  
## x <- x[x$ReturnNumber == 1, ]  
## allRast <- raster::rasterize(x, r, x$Z\_agl, fun = "count")  
## x <- x[x$Classification %in% pointClasses, ]  
## vegRast <- raster::rasterize(x, r, x$Z\_agl, fun = "count")  
## canCovRast <- vegRast/allRast \* 100  
## names(canCovRast) <- "ccov"  
## return(canCovRast)  
## }  
## <environment: namespace:USGSlvm>

ccov <- USGSlvm::calcCanopyCover(las\_data, resolution)  
  
USGSlvm::calcCanopyDensity

## function (x, resolution = 30, pointClasses = c(100:200))   
## {  
## r <- raster::raster(x, resolution = resolution)  
## allRast <- raster::rasterize(x, r, x$Z\_agl, fun = "count")  
## x <- x[x$Classification %in% pointClasses, ]  
## vegRast <- raster::rasterize(x, r, x$Z\_agl, fun = "count")  
## canDenRast <- vegRast/allRast \* 100  
## names(canDenRast) <- "cden"  
## return(canDenRast)  
## }  
## <environment: namespace:USGSlvm>

cdens <- USGSlvm::calcCanopyDensity(las\_data, resolution)  
  
USGSlvm::calcHeightPercentiles

## function (x, resolution = 30, percentiles = c(0.1, 0.2, 0.3,   
## 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 0.98), pointClasses = c(100:200))   
## {  
## tile\_raster <- raster::raster(x, resolution = resolution)  
## perc\_stack <- raster::raster(x, resolution = resolution)  
## x <- x[x$Classification %in% pointClasses, ]  
## for (i in percentiles) {  
## perc\_rast <- raster::rasterize(x, tile\_raster, field = x$Z\_agl,   
## fun = function(x, ...) {  
## quantile(x, i, na.rm = T)  
## })  
## perc\_stack <- raster::addLayer(perc\_stack, perc\_rast)  
## }  
## perc\_names <- as.character(percentiles \* 100)  
## names(perc\_stack) <- perc\_names  
## return(perc\_stack)  
## }  
## <environment: namespace:USGSlvm>

hpct <- USGSlvm::calcHeightPercentiles(las\_data, resolution)  
  
USGSlvm::calcHeightPointCounts

## function (x, resolution = 30, binHeight = 5, binCount = NA, pointClasses = c(100:200))   
## {  
## r <- raster::raster(x, resolution = resolution)  
## pcStack <- raster::raster(x, resolution = resolution)  
## x <- x[x$Classification %in% pointClasses, ]  
## maxAgl <- max(x$Z\_agl, na.rm = TRUE)  
## if (is.na(binHeight)) {  
## binHeight <- maxAgl/binCount  
## }  
## else {  
## binCount <- ceiling(maxAgl/binHeight)  
## }  
## bottom <- 0  
## for (i in 1:binCount) {  
## bottom <- bottom  
## top <- bottom + binHeight  
## t <- x[bottom <= x$Z\_agl & x$Z\_agl < top, "Z\_agl"]  
## if (nrow(t) == 0) {  
## countRast <- raster::setValues(r, 0)  
## }  
## else {  
## countRast <- raster::rasterize(t, r, field = t$Z\_agl,   
## fun = "count")  
## }  
## pcStack <- raster::addLayer(pcStack, countRast)  
## bottom <- top  
## }  
## return(pcStack)  
## }  
## <environment: namespace:USGSlvm>

hcnt <- USGSlvm::calcHeightPointCounts(las\_data, resolution)  
  
# Notice this function take an input from calcHeightPointCounts instead of   
# las\_data. While it is capable of processing the las file, CalcHieghtPointPercents  
# would require a call to calcHeightPointCounts. Providing the result in the way  
# avoids calling the function twice.  
USGSlvm::calcHeightPointPercents

## function (x, resolution = 30, binHeight = 5, binCount = NA, pointClasses = c(100:200))   
## {  
## if (class(x) == "SpatialPointsDataFrame") {  
## pcLayer <- USGSlvm::heightPointCounts(x, resolution = resolution,   
## binHeight, binCount, pointClasses)  
## }  
## else {  
## pcLayer <- x  
## }  
## pcSumLayer <- sum(pcLayer, na.rm = T)  
## percentLayer <- pcLayer/pcSumLayer \* 100  
## return(percentLayer)  
## }  
## <environment: namespace:USGSlvm>

hdens <- USGSlvm::calcHeightPointPercents(hcnt, resolution)