Tree_Segmentation

Luke Wertis

2022-10-18

```
library(lidR) #https://r-lidar.github.io/lidRbook/outbox.html#outbox-custom-metrics
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(sp)
```

```
## Warning: package 'sp' was built under R version 4.2.1
```

```
library(sf)
```

```
## Linking to GEOS 3.9.1, GDAL 3.3.2, PROJ 7.2.1; sf_use_s2() is TRUE
```

Set-Up

In this section the LAS file is read into R. The default measurments are in US-ft^2 but the package, lidR, which is the primary package used in this process uses meters so our measurments are converted to meters.

```
LAS = lidR::readLAS("C:\\Users\\wertisml\\Documents\\Hemlocks\\LiDAR\\Valie_Crusis\\las\\merged.las",
                    select = "*")

ft2m = 1200/3937
LAS$X = LAS$X * ft2m
LAS$Y = LAS$Y * ft2m
LAS$Z = LAS$Z * ft2m
st_crs(LAS) <- 6542
```

Normalize Height

Typically LiDAR data is converted into a raster inorder to create a Canopy Height Model, which is the difference between a Digital Terain Model and Digital Surface Model. This step is performed to remove the affect of elevation within the plot, that way an object that is 20m tall at the base of a 200m hill does not apear shorter than a 5m object at the top of the hill. Using the lidR pacakge we are able to normalize the height within the LAS format, saving the need to create a DTM.

Additionally there is an option within this function to use a raster, DSM to normalize height, however, it has been shown that using the knnidw() (k-nearest neighbor) provides better results due to it having a higher spatial resolution than a raster, therefore having no negative elevation measurments which can appear when using a raster.

```
LAS <- normalize_height(LAS, knnidw())
```

```
## Inverse distance weighting: [=--------------------------------------------------] 3% (2 threads)Inver
```

create canopy height model

a raster is created out of the normalize LiDAR data using rasterize_canopy. a matrix is created which acts as a window to search for local maximums which for this case is the tops of trees. A window size of 5x5 was determined to have the highest f-score when comparing how different methods segmented trees.

the focal function is used to smooth out the empty spaces that may appear in the rastized canopy.

the fin function is a linear model representing the canopy diameter as a function of its height (x), several linear and nonlinear models were tested and based on the f-score when comparing the ability to correctly segment trees.

the locate_trees is used with the preivious two models to find the position of the trees within the LAS file. hmin represents the height that an object must be inorder to be picked up in this algorithm. The intention of using 2m as the minimum height is to eliminate underbrush from the study.

```
chm_p2r <- rasterize_canopy(LAS, 0.5, p2r(subcircle = 0.2), pkg = "terra")

kernel <- matrix(1,5,5)
chm_p2r_smoothed <- raster::focal(chm_p2r, w = kernel, fun = median, na.rm = TRUE)

fin <- function(x) {x * 0.02 + 4}

ttops_chm_p2r_smoothed <- locate_trees(chm_p2r_smoothed,
                                       lmf(ws=fin, hmin = 2, shape = "circular"))
```

```
## Local maximum filter: [=================-------------------------------] 35% (2 threads)Local maxi
```

Segment Trees Within the LiDAR Data

the smoothed raster model and the tree positions are applied to a function named silva2016, which is based on the methods used in a previous tree segmentation paper (Silva et al., 2016). The silva method was tested against several other tree segmentation algorithms and based off the f-score, this method was determined to be the superior aproach for our study areas.

The output of the silva2016 function is the information pertaining to individual trees, which is grouped together by unique ids (treeID). This output is then run along with the original normalized LiDAR data to get individual segmented trees.

```
algo1 <- silva2016(chm_p2r_smoothed, ttops_chm_p2r_smoothed, max_cr_factor = 0.6, exclusion = 0.3)

las <- segment_trees(LAS, algo1)
```

Tree Metrics

Once the trees have been indivuudally segmented, the lidR pacakge provides several methods of extracting informatio out of the LiDAR. Namely we are able to get the z (elevation), i (intensity), rn (return number), q (quantile), n (number), and p (percentage), these can be applied to user built functions or to lidR provided functions.

In this step I created a function named stdmetrics_int to get the intensity at each 5th quantile of height for each tree. This was done to later calculate the canopy base height, which is done later in the documentation.

Three other lidR crown_metric provided functions were used to gether as much information about the trees as possible.

```
stdmetrics_int <- function(i, z = NULL, class = NULL, rn = NULL)
{
  itot <- imax <- imean <- isd <- icv <- iskew <- ikurt <- NULL
  icumzq10 <- icumzq30 <- icumzq50 <- icumzq70 <- icumzq90 <- NULL

  n <- length(i)
  itot <- as.double(sum(i))
  imean <- mean(i)

  probs <- seq(0.05, 0.95, 0.05)
  iq      <- as.list(stats::quantile(i, probs))
  names(iq) <- paste("iq", probs*100, sep = "")

  metrics <- list(
    itot = itot,
    imax  = max(i),
    imean = imean,
    isd   = stats::sd(i),
    iskew = (sum((i - imean)^3)/n)/(sum((i - imean)^2)/n)^(3/2),
    ikurt = n * sum((i - imean)^4)/(sum((i - imean)^2)^2)
  )

  if (!is.null(class))
  {
    metrics <- c(metrics, list(ipground = sum(i[class == 2])/itot*100)) #look into adding information a
  }

  if (!is.null(z))
  {
    zq <- stats::quantile(z, probs = seq(0.0, 1, 0.05))

    ipcum <- list(
      ipcumzq05 = sum(i[z <= zq[1]])/itot*100,
      ipcumzq10 = sum(i[z <= zq[2]])/itot*100,
      ipcumzq15 = sum(i[z <= zq[3]])/itot*100,
      ipcumzq20 = sum(i[z <= zq[4]])/itot*100,
```

```
      ipcumzq25 = sum(i[z <= zq[5]])/itot*100,
      ipcumzq30 = sum(i[z <= zq[6]])/itot*100,
      ipcumzq35 = sum(i[z <= zq[7]])/itot*100,
      ipcumzq40 = sum(i[z <= zq[8]])/itot*100,
      ipcumzq45 = sum(i[z <= zq[9]])/itot*100,
      ipcumzq50 = sum(i[z <= zq[10]])/itot*100,
      ipcumzq55 = sum(i[z <= zq[11]])/itot*100,
      ipcumzq60 = sum(i[z <= zq[12]])/itot*100,
      ipcumzq65 = sum(i[z <= zq[13]])/itot*100,
      ipcumzq70 = sum(i[z <= zq[14]])/itot*100,
      ipcumzq75 = sum(i[z <= zq[15]])/itot*100,
      ipcumzq80 = sum(i[z <= zq[16]])/itot*100,
      ipcumzq85 = sum(i[z <= zq[17]])/itot*100,
      ipcumzq90 = sum(i[z <= zq[18]])/itot*100,
      ipcumzq95 = sum(i[z <= zq[19]])/itot*100,
      ipcumzq100 = sum(i[z <= zq[20]])/itot*100
    )

    metrics <- c(metrics, ipcum)
  }

  metrics
}

returns <- crown_metrics(las, func = ~stdmetrics_int(Intensity, Z), geom = "concave")
crowns <- crown_metrics(las, func = .stdtreemetrics, geom = "concave")
base <- crown_metrics(las, func = .stdmetrics, geom = "concave")
shapes <- crown_metrics(las, func = .stdshapemetrics, geom = "concave")
```

Combine Dataframes

In this step the dataframes created from the crown_metrics are combined and duplicate information is removed.

```
base <- base[ -c(45:49) ]

tree_metrics <- cbind(crowns, base)
tree_metrics <- cbind(tree_metrics, shapes)
tree_metrics <- cbind(tree_metrics, returns)
#Need to fix this now that there is a new dataframe added in
tree_metrics[,95:97] <- NULL
tree_metrics[,67:73] <- NULL
tree_metrics[,57] <- NULL
tree_metrics[,5] <- NULL
```

Canopy Base Height

The canopy base height is calculated in this section. Previous research has indicated that the canopy base hight is an important metric when classifying trees. The method used to calculate the canopy base height is to find the inflection point of the intensity of the LiDAR. In principle, the intesity returns will be relativley low as the light passes through the tree canopy and then the intensity return will increase sharply once it

passes through the canopy and is in a relativley sparse space and will then go back to low intensity returns once it begins to pass through under canopy growth. The method being used below identifys the quantile of height when the intensity of return spikes, indicating the sparse space. That quantile is then multiplied by the height of the tree to indicate roughly where the canopy base begins.

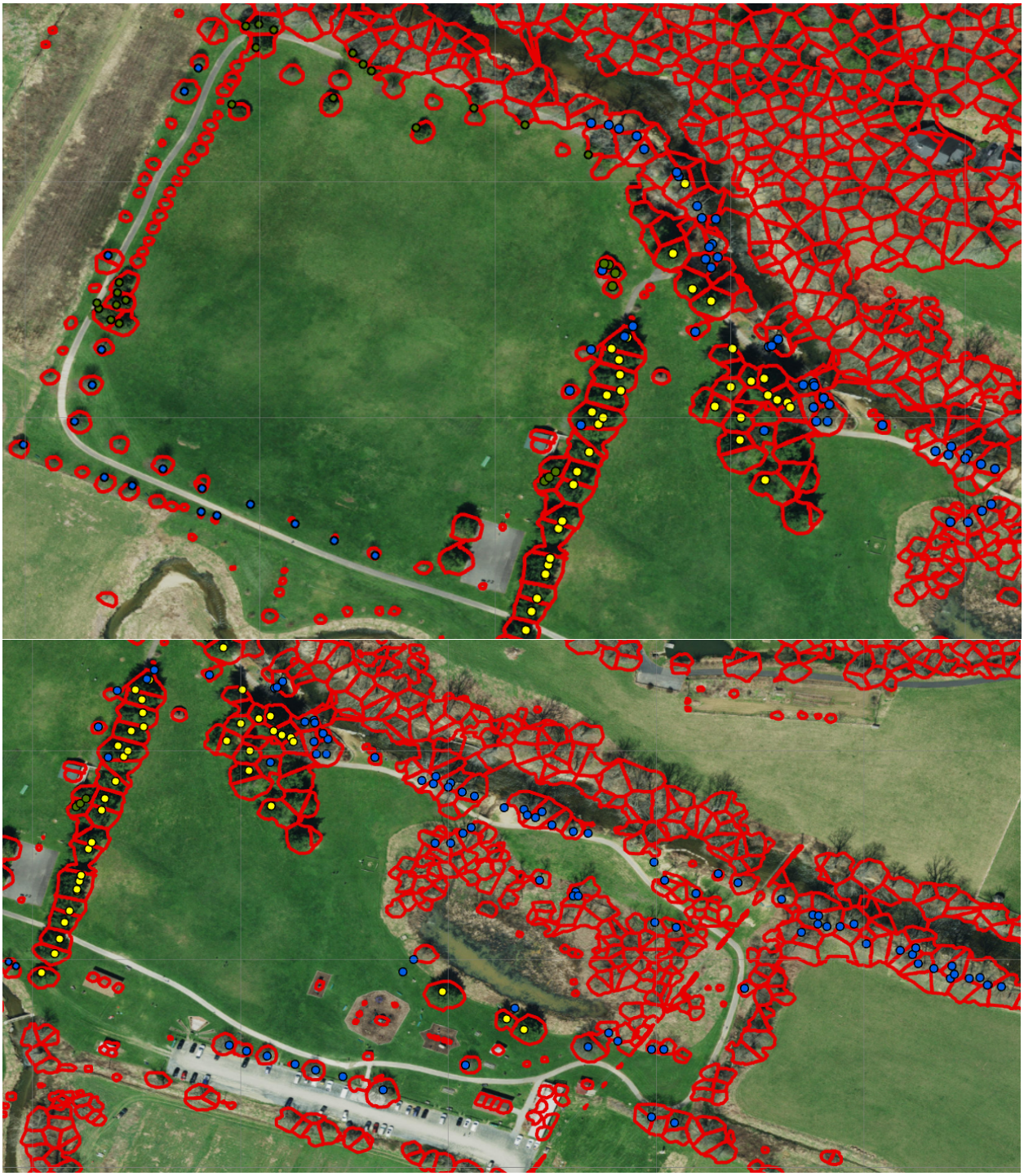
```
tree <- as.data.frame(tree_metrics)

# The columns containing the intensity percentage cumulative height percentile
intes_height <- tree[,65:84]

# Calculating the intensity differnce between each height percentile
n = 10
for(i in 1:ncol(intes_height)){
  if(i < 20) {

    intes_height[[paste0("diff_", n)]] = intes_height[,i+1] - intes_height[,i]

    n = n+5
    i = i+1
  }
}

# Determine the column with the biggest intensity change and get the height percentile
intes_height <- intes_height %>%
  mutate(Largest_Column = colnames(.[,21:39])[apply(.[,21:39],1,which.max)],
         height = as.numeric(sub(".*_", ".", Largest_Column)))

# Multiply the height percentile by the height of the tree
tree_metrics$Canopy_Base <- tree_metrics$Z * intes_height$height
```

Post Information

after the previous step the data is writen off as a shapefile for further analysis. However we can look back at what has been performed to get an understanding of what the data outputs look like.

Based on the field true point locations and having already doen this earlier we can plot the outcomes of our three tree types that we are interested in, Hemlocks, Pines, and Deciduous.

Below are two images of Valle Crusis park with red outlines indicating the tree bounds from the individual tree segmentation which is created from the steps above. There are also points of three different colors, the dark green are the locations of Hemlocks, the yellow are the locations of pine trees, and the blue points are the locations of deciduous trees.



View segmented trees

Each outline of a tree comes with a unique treeID. The treeID and the true point locations are then used to assign which tree outline corresponds to what type of tree. Below is the method used to display this information, the images show the segmented trees in LiDAR.

```
Hemlock_Trees <- c(6385, 6358, 6329, 6278, 5746, 5495, 5424, 5727, 5495, 5811,
                   6251, 6279, 6318, 6632, 6642)

Hemlocks <- las %>%
  filter_poi(treeID %in% Hemlock_Trees)

plot(Hemlocks, size = 8, bg = "white", color = "treeID")

# Display Pines
```

```
Pine_Trees <- c(6823, 6843, 6889, 6878, 6868, 6718, 6312, 6362, 6495, 6541, 6525,
                6572, 6578, 6588, 6561, 6585, 6624, 6601, 6621, 6645, 6651, 6991,
                6977, 6928, 6787, 6752, 6699, 6683, 6649, 6638, 6619, 6612, 6579,
                6557, 6578, 6496, 6765)

Pines <- las %>%
  filter_poi(treeID %in% Pine_Trees)

plot(Pines, size = 8, bg = "white", color = "treeID")

# Display Decidous

Deci_Trees <- c(6640, 6622, 6644, 6637, 6573, 6443, 6198, 6212, 6599, 5552, 5448,
                5435, 5409, 5375, 5441, 5644, 5484, 5671, 5553, 5624, 5696, 5768,
                5866, 5886, 5984, 6077, 6252, 6307, 6369, 6580, 6569, 6576, 6544,
                6623, 6614, 6602, 6613, 6639, 6627, 6646, 6663, 6676, 6687, 6704,
                6761, 6757, 6777, 6814, 6834, 6841, 6841, 6786, 6817, 6791, 6826,
                6842, 6897, 6920, 6941, 7031, 7020, 7131, 7141, 7150, 7019, 7043,
                7090, 7046, 7023, 7012, 7000)

Deciduous <- las %>%
  filter_poi(treeID %in% Deci_Trees)

plot(Deciduous, size = 8, bg = "white", color = "treeID")
```

Getting tree pixel values

```
library(data.table)
```

```
##
## Attaching package: 'data.table'

## The following objects are masked from 'package:dplyr':
##
##     between, first, last
```

```
library(dplyr)
library(rgdal)
```

```
## Please note that rgdal will be retired by the end of 2023,
## plan transition to sf/stars/terra functions using GDAL and PROJ
## at your earliest convenience.
##
## rgdal: version: 1.5-31, (SVN revision 1171)
## Geospatial Data Abstraction Library extensions to R successfully loaded
## Loaded GDAL runtime: GDAL 3.3.2, released 2021/09/01
## Path to GDAL shared files: C:/Users/wertisml/AppData/Local/R/win-library/4.2/rgdal/gdal
## GDAL binary built with GEOS: TRUE
## Loaded PROJ runtime: Rel. 7.2.1, January 1st, 2021, [PJ_VERSION: 721]
## Path to PROJ shared files: C:/Users/wertisml/AppData/Local/R/win-library/4.2/rgdal/proj
```

```
## PROJ CDN enabled: FALSE
## Linking to sp version:1.4-7
## To mute warnings of possible GDAL/OSR exportToProj4() degradation,
## use options("rgdal_show_exportToProj4_warnings"="none") before loading sp or rgdal.
```

```
library(sf)
library(raster)
```

```
## Warning: package 'raster' was built under R version 4.2.1
```

```
##
## Attaching package: 'raster'
```

```
## The following object is masked from 'package:dplyr':
##
##     select
```

```
## The following objects are masked from 'package:lidR':
##
##     projection, projection<-
```

```
library(tictoc)
library(lidR)

Tree_Shapefile <- st_read("C:\\Users\\wertisml\\Documents\\Hemlocks\\Scripts_and_Tools\\tree_crowns\\Fi
```

```
## Reading layer 'Valie_Crusis_Silva_fin_5x5' from data source
##   'C:\Users\wertisml\Documents\Hemlocks\Scripts_and_Tools\tree_crowns\Final\Valie_Crusis_Silva_fin_5
##   using driver 'ESRI Shapefile'
## Simple feature collection with 11031 features and 94 fields
## Geometry type: POLYGON
## Dimension:     XY
## Bounding box:  xmin: 359664.7 ymin: 275844.6 xmax: 360426.5 ymax: 277368.5
## Projected CRS: NAD83(2011) / North Carolina
```

```
NAIP <- stack("C:\\Users\\wertisml\\Documents\\Hemlocks\\Aerial_Photography\\NC_OneMap\\tif\\Valle_Crusi
NAIP_br <- brick(NAIP)
```

Extract tree pixel values

The shapefile of the trees is used to determine the mean pixel value for each band within the tree region of space. This is meant to get a general understanding of the type of tree being observed. The strength of this method in our study is that everything is done during leaf off period, meaning that only Pines and Hemlocks will be green, hopefully making it easier to classify them.

```
Tree_Shape <- st_transform(Tree_Shapefile, st_crs(NAIP_br))
#Crop to the study area
r.crop <- crop(NAIP_br, extent(Tree_Shape))
#get the mean pixel value within each polygon
r.mean = extract(r.crop, Tree_Shapefile, method="simple", fun=mean, sp=T)
```

```
## Warning in .local(x, y, ...): Transforming SpatialPolygons to the crs of the
## Raster
```

```
# Write results
Tree_Stats <- as.data.frame(r.mean)

fwrite(Tree_Stats, "~/Hemlocks/Scripts_and_Tools/tree_crowns/Final/csv/Valle_Crusis.csv")
```