1D Post-Hoc Binning

Isolde Lane-Shaw

03/31/2021

Introduction

In order to manage and conserve bird species within the boreal forest is is necessary to be able to predict bird species abundance across future/hypothetical landscapes, based off of reliable models and data. To this end, a series of national models of bird density have been created based off the Boreal Avian Modelling (BAM) Project dataset (Boreal Avian Modelling Project, 2020). The BAM dataset is a collation of over 100 avian studies from across the Canadian boreal region (the majority of which comprise point-count data). In the most recent version of these models, a series of bootstrapped bird density predictions for the Canadian boreal region are output as rasters, at a resolution of 250m2.

Due to the heterogenous nature of the environment and the extensive quantity of data available in these rasters a degree of simplification is necessary in order to make predictions about bird abundance on future or hypothetical landscapes. To this end, this module serves to distill the bird density predictions data, grouping it according to broad forest and land cover types.

This module calculates predicted bird densities from the mean of the most recent bootstrapped BAM national models of bird density rasters, for each of a given set of cover and age classes according to two different methods: - 1D, where a single variable, underlying cover class, is used to bin data points of predicted bird density, by calculating the mean. This method is used for both forested and non-forested cover classes. - 2D, where a gbm is used to give a predicted bird density according the combination of two variables, cover class and age. The predicted bird densities are then further binned according to the mean value of the desired age classes. This method is used for forested cover classes only.

In this document we demonstrate the 1D method

Get rasterToMatch and study area files

Here the LLC2005 raster layer of Canada is read in using the `LandR::prepInputsLCC` function. It will be used as the `rasterToMatch` layer for other spatial layers to reproject, crop and mask to.

```
#rasterToMatch <- LandR::prepInputsLCC(destinationPath = inputsDir)
```

##Define Study Area

The study area that we are interested in is then downloaded and read in using the prepInputs function. In this example we are using an area of North-East British Columbia.

```
#specify file name
nameAreaShapefile <- "BCR6_BC.shp"
folderUrlArea <- "https://drive.google.com/file/d/1SjUKXOcNqDUkRxRTjwgLlLFjb5LRzbHy"
#give archive name
archiveArea <- "BCR6_BC.zip"

studyArea <- prepInputs(targetFile = nameAreaShapefile,
                        url = folderUrlArea,
                        archive = archiveArea,
                        #Extract other files with similar names
                        alsoExtract = "similar",
                        #save the file to a folder in the
                        #working directory called studyArea
                        destinationPath = downloadFolderArea,
                        #use the function shapefile
                        fun = "raster::shapefile",
                        targetCRS = crs(rasterToMatch),
                        #use the specified rasterToMatch to reproject to
                        #rasterToMatch = rasterToMatch,
                        overwrite = TRUE,
                        verbose = TRUE)
# studyArea <- spTransform(studyArea, crs(rasterToMatch))

#create landscapeRaster by cropping rasterToMatch to studyArea
rasterToMatch <- raster::crop(rasterToMatch, studyArea)
#mask landscapeRaster to studyArea
rasterToMatch <- raster::mask(rasterToMatch, studyArea)
#landscapeRasters <- unstack(landscapeRastersStack)
```

Get Landscape Data

Here we download and load the rasters giving data on the landscape.

Input land cover class rasters

First we download a raster that gives the forest cover types, in forested areas.

```
#specify file name
nameForClassRaster <- "VegTypeABBC.tif"
folderUrlForClass <- "https://drive.google.com/file/d/1cfrb-RhiwMD4XlS_yzRSQYPVh8ffwCOL/view?usp=sharin
#give archive name
archiveForClass <- "vegTypeABBC.zip"

forClassRaster <- prepInputs(targetFile = nameForClassRaster,
                        url = folderUrlForClass,
                        archive = archiveForClass,
                        #Extract other files with similar names
                        alsoExtract = "similar",
                        #save the file to a folder in the
                        #working directory called forestClassRasters
                        destinationPath = downloadFolderForestClass,
                        #use the function raster
                        #targetCRS = crs(rasterToMatch),
                        fun = "raster::raster",
                        #use the specified rasterToMatch to reproject to
                        rasterToMatch = rasterToMatch,
                        studyArea = studyArea,
                        useCache = getOption("reproducible.useCache", FALSE),
                        overwrite = TRUE,
                        verbose = TRUE)
```

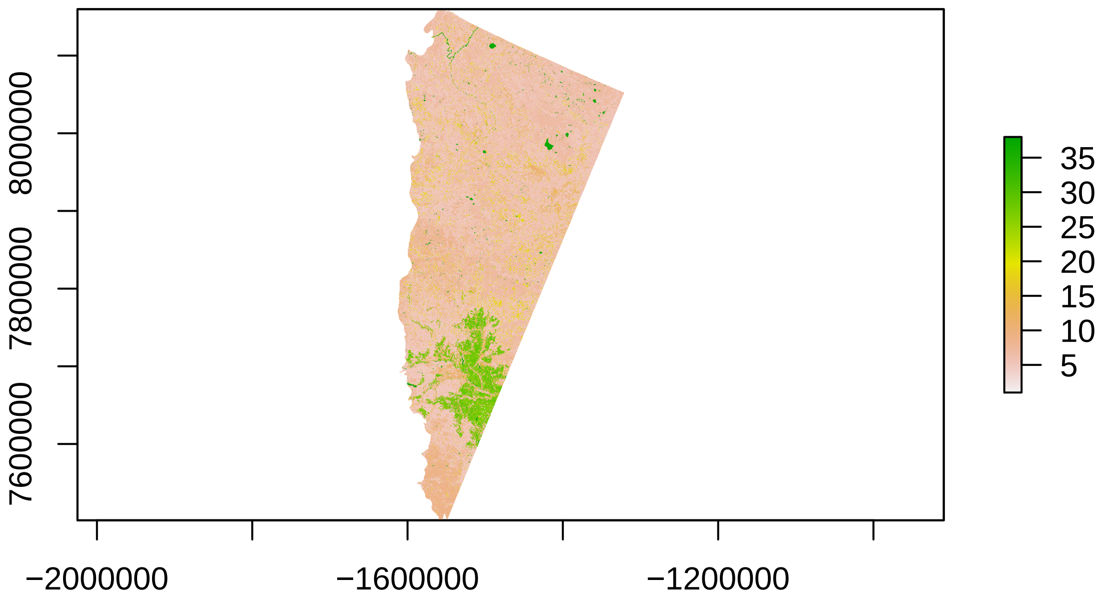
```r
names(forClassRaster) <- c("forClassRaster")

forClassRaster[forClassRaster == 0] <- NA

# #rename the categories found in the rasters
# namesValues <- as.factor(unique(forClassRaster))
# newVals <- unlist(lapply(X = namesValues, FUN = function(x) {
# newVal <-  as.factor(paste0("FR", x))
#     return(newVal)
#   }))
# reclassVector <- c(rbind(namesValues, newVals))
# reclassMatrix <- matrix(reclassVector,
#                                ncol=2, byrow = TRUE)
# forClassRaster <- reclassify(forClassRaster,
#                       reclassMatrix)
```

Then we input a raster that gives cover types in non forested areas. In this example the LCC05 map of Canada is used.

```r
#specify file name
nameNonForRaster <- "LCC2005_V1_4a.tif"
folderUrlNonFor <- "https://drive.google.com/file/d/1E3iQRfaSPx0-b2GTgbLZqjo2Nhom39EK/view?usp=sharing"

#give archive name
archiveNonFor <- "LCC2005_V1_4a.zip"
nonForRaster <- prepInputs(targetFile = nameNonForRaster,
                        url = folderUrlNonFor,
                        archive = archiveNonFor,
                        #Extract other files with similar names
                        alsoExtract = "similar",
                        #save the file to a folder in the
                        #working directory called forestClassRasters
                        destinationPath = downloadFolderForestClass,
                        #use the function raster
                        fun = "raster::raster",
                        #targetCRS = crs(rasterToMatch),
                        #use the specified rasterToMatch to reproject to
                        rasterToMatch = rasterToMatch,
                        studyArea = studyArea,
                        useCache = getOption("reproducible.useCache", FALSE),
                        overwrite = TRUE,
                        verbose = TRUE)

names(nonForRaster) <- c("nonForRaster")

nonForRaster <- overlay(x = nonForRaster,
                        y = forClassRaster,
                        fun = function(x, y) {
                         x[!is.na(y[])] <- NA
                          return(x)
                            })

#rename the categories found in the rasters
# namesValues <- unique(nonForRaster)
# newVals <- unlist(lapply(X = namesValues,
# FUN = function(x) {
# newVal <-  as.factor(paste0("NF", x))
#     return(newVal)
#   }))
# reclassVector <- c(rbind(namesValues, newVals))
# reclassMatrix <- matrix(reclassVector,
#                                 ncol=2, byrow = TRUE)
# nonForRaster <- reclassify(nonForRaster,
#                       reclassMatrix)
```

We then combine the nonForRaster and the forClassRaster so that we have a single raster giving the cover types (both forested and non-forested) over the study region.

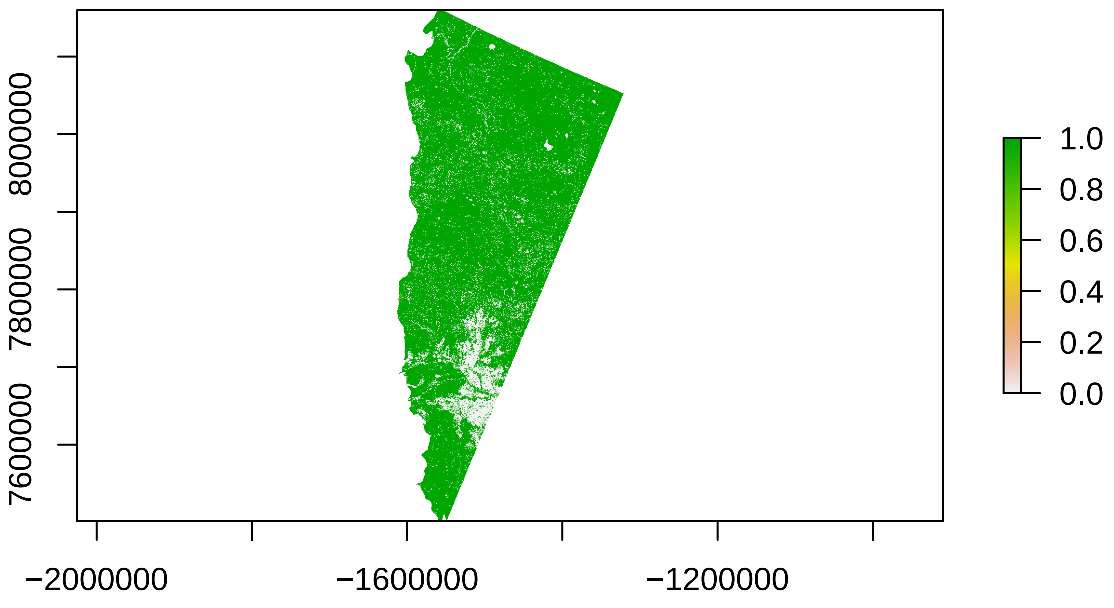
```
landscapeRaster <- cover(x = forClassRaster,
                         y = nonForRaster,
                         filename = "landscapeRaster",
                         overwrite = TRUE )

names(landscapeRaster) <- c("landscapeRaster")

#visually check nonForRaster
plot(landscapeRaster)
```



Create raster giving status of the landcover as forested or non-forested

This is based on the presence or abscence of data in the forClassRaster.

```r
#create a raster that gives if an area is forest or not
#(0 for non-forest, 1 for forest)
#This will allow me to have a value in the birdDataset
#that says if a cell was forested or not

#make forest 1
valsFR <- unique(forClassRaster)
newValFR <-  as.factor(rep("1", length(valsFR)))
newValsFR <- cbind(valsFR, newValFR)
reclassMatrixFR <- matrix(newValsFR,
                              ncol=2, byrow = FALSE)
rasterFR <- reclassify(forClassRaster,
                     reclassMatrixFR)

#make the rest 0
replaceNA <- function(x, na.rm, ...){
  if(is.na(x[1]))
    return(0)
  else
    return(x)
}

FNFRaster <- calc(rasterFR, fun = replaceNA)
#mask back down to size again
FNFRaster <- raster::mask(FNFRaster,rasterToMatch)

names(FNFRaster) <- c("FNFRaster")

#visually check FNFRaster
plot(FNFRaster)
```



Age Raster Input

Finally, we input a raster that gives the ages of the forested areas in age classes.

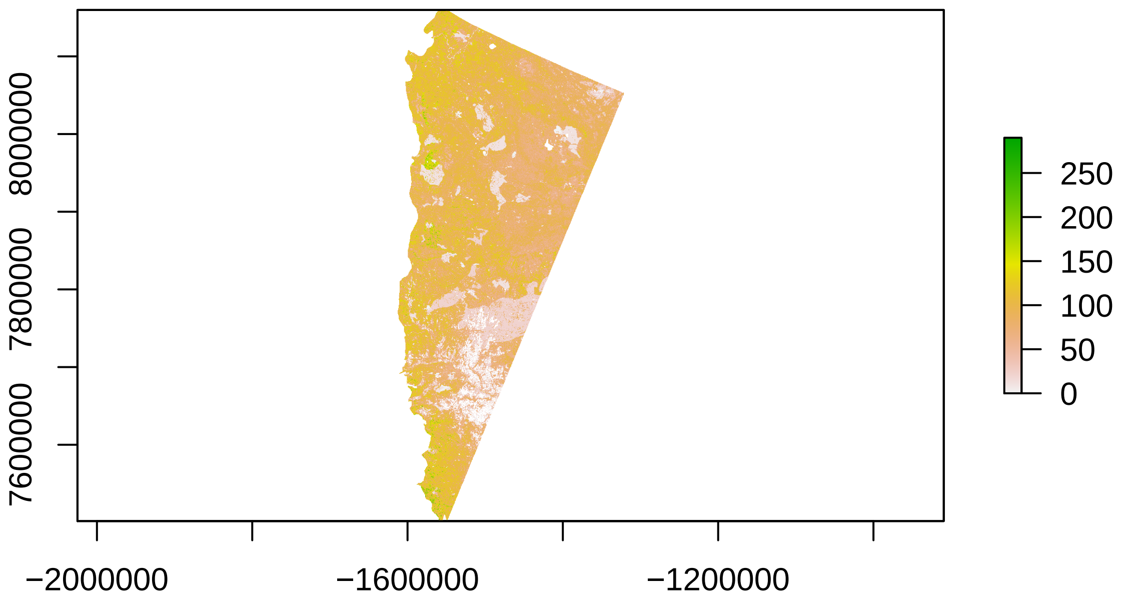
```
#specify file name
nameForAgeRaster <- "ageRas.tif"
folderUrlAge <- "https://drive.google.com/file/d/1Ih4YLfIYX6nQcotdd41j4yRtZKRQxYo7/view?usp=sharing"
#give archive name
archiveAge <- "ageRaster.zip"

forAgeRaster <- prepInputs(targetFile = nameForAgeRaster,
                        url = folderUrlAge,
                        archive = archiveAge,
                        #Extract other files with similar names
                        alsoExtract = "similar",
                        #save the file to a folder in the working directory
                        #called forestClassRasters
                        destinationPath = downloadFolderForestClass,
                        #use the function raster
                        fun = "raster::raster",
                        #targetCRS = crs(rasterToMatch),
                        #use the specified rasterToMatch to reproject to
                        rasterToMatch = rasterToMatch,
                        studyArea = studyArea,
                        overwrite = TRUE,
                        verbose = TRUE)
```

```
names(forAgeRaster) <- c("forAgeRaster")
```

Get KNN age raster to fill in non forested ages

```
#specify file name
namekNNAgeRaster <- "NFI_MODIS250m_2011_kNN_Structure_Stand_Age_v1.tif"
folderUrlkNNAge <- "https://drive.google.com/file/d/1Ih4YLfIYX6nQcotdd41j4yRtZKRQxYo7/view?usp=sharing"
#give archive name
archivekNNAge <- "ageRaster.zip"

kNNAgeRaster <- prepInputs(targetFile = namekNNAgeRaster,
                        url = folderUrlkNNAge,
                        archive = archivekNNAge,
                        #Extract other files with similar names
                        alsoExtract = "similar",
                        #save the file to a folder in the working directory
                        #called forestClassRasters
                        destinationPath = downloadFolderForestClass,
                        #use the function raster
                        fun = "raster::raster",
                        #targetCRS = crs(rasterToMatch),
                        #use the specified rasterToMatch to reproject to
                        rasterToMatch = rasterToMatch,
                        studyArea = studyArea,
                        overwrite = TRUE,
                        verbose = TRUE)

names(kNNAgeRaster) <- c("kNNAgeRaster")
```

```
ageRaster <- cover(x = forAgeRaster,
                         y = kNNAgeRaster,
                         filename = "ageRaster",
                         overwrite = TRUE )

names(ageRaster) <- c("ageRaster")

#visually check nonForRaster
plot(ageRaster)
```



Get Bird Density Data

Download/load 250m bootstrapped bird rasters for mean and variance, and the DF giving numbers of bootstrap replicates the rasters were derived from.

Multiple rasters of predicted bird density are downloaded, postProcessed, loaded and output as a raster stack, alongside rasters giving the variance between bootstraps at each pixel.

To do this we first define a function to download rasters for the chosen bird species, and output a list of the rasters that have been downloaded.

```
download250mBootRasters <-
  function(folderUrl, birdsList, rastersPath) {
    ## drive_ls function is used to list all the
    ##files it finds using the folder url with the given pattern
    filesToDownload <-
      googledrive::drive_ls(path = as_id(folderUrl))
    #note: has to be changed if filenaming system changes
    ## grepl function searches for all items in the
    ##filesToDownload that are on birdList & stores their
    ##names in rastersforBirdList
    rastersForBirdList <-
      filesToDownload$name[grepl(pattern = paste(birdsList,
                                                 collapse = "|"),
                                 x = filesToDownload$name)]

    ## for each item in turn from rastersForBirdlist
```

```r
  #the following function is applied:
  downloadedRasters <-
    lapply(
      X = rastersForBirdList,
      FUN = function(rasterFile) {
        ## if the item in rastersForBirdList is
        #not already present at rastersPath, googledrive
        #package downloads it
        if (!file.exists(file.path(rastersPath, rasterFile))) {
          googledrive::drive_download(
            file = as_id(filesToDownload[filesToDownload$name %in% rasterFile,]$id),
            #rasterFile,
            path = file.path(rastersPath,
                             rasterFile),
            overwrite = TRUE
          )
        }

        ## otherwise, if it is already present
        ##and downloaded, just get the name of the item
        return(raster(file.path(rastersPath, rasterFile),
                      verbose = TRUE))
      }
    )

  #get the species codes as names for the
  ##downloadedRasters object, rather than using the whole filepath
  X <-
    lapply(rastersForBirdList, substr, 1, 8)
   #works for strings of the form "varsXXXX" or "meanXXXX"
  names(downloadedRasters) <- X
  #the downloadBirdDensityRasters function returns/loads
  #the downloaded rasters
  return(downloadedRasters)
}
```

The download bird function is called by a second defined function, which postprocesses (reprojects, crops and masks according to the given rasterToMatch and studyArea) and loads the rasters in a raster stack.

```r
load250mBootRasters <- function(birdsList,
                                folderUrl,
                                rastersPath,
                                rasterToMatch,
                                studyArea) {
  reproducible::Require("raster")
  ## check that there is a folder at the given rastersPath.
  #if not, create it.
  rastersPath <- checkPath(file.path(rastersPath),
                           create = TRUE)
  ## download the rasters on the birdList.
  #Return List of downloaded files.
  downloadedRasters <- download250mBootRasters(folderUrl = folderUrl,
                                               birdsList = birdsList,
                                               rastersPath = rastersPath)
  ## lapply applys the custom function to each raster in turn
  postprocessedRasters <- lapply(X = downloadedRasters,
                                 FUN = function(RasterLayer) {
    ## the function postProcesses the layer, cropping
    ## and masking it to a given study area and rasterToMatch,
    ## and saving it to a given destination path
    proRaster <- postProcess(RasterLayer,
                             studyArea = studyArea,
                             rasterToMatch = rasterToMatch,
                             destinationPath = rastersPath)
    ## each layer is returned into the object proRaster
    return(proRaster)
  })

  ## Finally the whole function returns the birdRasterStacks
  return(postprocessedRasters)
}
```

Here, we define which bird species we want, where to download the bird density rasters from, and where to save them. It then calls the `loadBirdDensityRaster` function.

We also download a table giving the number of bootstrapped replicates that existed for each bird species.

```r
#Specify which bird species to examine/input
birdsList <- c("BAWW", "OVEN")
# give file location
folderUrlBird <- "https://drive.google.com/drive/folders/1fCTr2P-3Bh-7Qh4W0SMJ_mT9rpsKvGEA"

birdRasterStack <- load250mBootRasters(folderUrl = folderUrlBird,
                                       birdsList = birdsList,
                                       rastersPath = downloadFolderBird,
                                       rasterToMatch = rasterToMatch,
                                       studyArea = studyArea)

noBootsDFLocation <- "https://drive.google.com/file/d/1ldESo9gb6icRD8ZsuPgaEDSIwFjJEe4W"
noBootsDF <- drive_download(file = noBootsDFLocation,
                            path = file.path(downloadFolderBird,
                                             "noBootsDF"),
                            type = "spreadsheet",
                            overwrite = TRUE)
```

Create data table of bird density by landscape

A `data.table` is created, containing the number of cells of each land cover class that are found in the `landscapeRaster`, and the variance between bootstrap replicates, mean bird density for each land cover class. The variance, and the standard error of this mean density is also calculated and included.

The first step is to gather the values from the rasters and create a clean dataset. We then calculate, for each cover class, the number of cells in this class, the mean bird density, and the variance and standard error for bird density.

Get bird dataset

Here the data for each cell's mean bird density and variance between bootstrap replicates is collected in a `data.table`, alongside the corresponding landscape and age raster cells.

```r
getBirdDataset <- function(birdRasterStack,
                           landscapeClassesRasterStack) {
  reproducible::Require("raster")

  meanBirdRasters <- names(birdRasterStack) %>%
                          str_detect('mean') %>%
                          keep(birdRasterStack, .)
  namesMeanBirdRasters <- names(meanBirdRasters)

  birdDatasets <- lapply(X = meanBirdRasters,
                         FUN = function(birdRasterLayer) {

    landBirdRasterStack <- raster::addLayer(birdRasterLayer, landscapeClassesRasterStack)


    ## take the values from the rasters and input
    ## them to a data table called cellValues
    cellValues <- data.table(getValues(landBirdRasterStack))
    cellValues <- setnames(cellValues, c( "birdDensity",
                                          "landForClass",
                                          "age",
                                          "forestedStatus"))
    cellValues <- unite(cellValues,
                        uniqueClasses,
                        c(forestedStatus,
                          landForClass),
                        remove=FALSE)

    #get rid of any rows with NA values
    cellValues <- na.omit(cellValues)

    ## make sure landForClass and forestedStatus
    ## are categorical rather than numerical
    cellValues$landForClass <- as.factor(cellValues$landForClass)
    cellValues$forestedStatus <- as.factor(cellValues$forestedStatus)
    cellValues$uniqueClasses <- as.factor(cellValues$uniqueClasses)

    return(cellValues)
  })

  names(birdDatasets) <- namesMeanBirdRasters

  return(birdDatasets)
}
```

```r
landscapeRastersStack <- raster::stack(landscapeRaster,
                                       ageRaster,
                                       FNFRaster)

birdDatasets <- getBirdDataset(birdRasterStack = birdRasterStack,
                               landscapeClassesRasterStack = landscapeRastersStack )

for (i in names(birdDatasets)) {
  attr(birdDatasets[[i]],"Species") <- i
  ## attr(birdDatasets$OVEN, "Species")
}

## demonstrate what the first few lines of
## one of the birdDatasets looks like
head(birdDatasets$meanOVEN)
```

```
##    birdDensity uniqueClasses landForClass       age forestedStatus
## 1:  0.04313287           0_1            1  70.15667              0
## 2:  0.04370008           0_1            1  76.91000              0
## 3:  0.04694378           0_1            1  95.23333              0
## 4:  0.04035857           0_1            1 105.46000              0
## 5:  0.04479010           0_6            6 116.14000              0
## 6:  0.04653142           0_6            6 130.67999              0
```

1D post-hoc binning predictions of bird densities by cover type alone

Get bird densities by cover class and calculate statistics

```
getBirdStatsByClass <- function(birdDatasets) {
  namesBirdsAnalysed <- names(birdDatasets)
  #base::attr()
  birdStatsByClass <- lapply(X = birdDatasets,
                             FUN = function(singleBirdDataset) {
    print(attr(singleBirdDataset,
               "Species"))
    ## TODO: use messages
    flush.console()
    ## TODO: remove!
    birdStats <- singleBirdDataset[order(-forestedStatus,
                                         landForClass)
                                   # order the rows by the land cover class
                                   ][,list(classCount = .N,
                                           # get the number of cells
                                           # each cover class
                                           #meanBirdDensity = mean(birdDensity),
                                           # get the mean bird density
                                           #for each cover class
                                           meanBirdDensity = mean(birdDensity),
                                           #try log of mean bird density
                                           varBirdDensity = var(birdDensity),
                                           # get the variance for bird density
                                           # for each cover class
                                           seBirdDensity = std.error(birdDensity),
                                           # get the standard error
                                           #for bird density
                                           # for each cover class
```

```
                                          normality = tryCatch(ad.test(birdDensity)$p.value,
                                                      error = function(cond) { return(NaN) })
#ifelse(mean(birdDensity) > 0,
#tryCatch(ad.test(birdDensity)$p.value,
#error = function(cond){return(NA)}), NA),
                                          unimodality =   dip.test(birdDensity)$p.value),
                                    by = list(forestedStatus,
                                              landForClass)]

    birdStats <- unite(birdStats,
                       uniqueClasses,
                       c(forestedStatus,
                         landForClass),
                       remove=FALSE)

    return(birdStats)
  })

  names(birdStatsByClass) <- namesBirdsAnalysed

  return(birdStatsByClass)
}
```

```
birdStatsByClass <- getBirdStatsByClass(birdDatasets = birdDatasets)
```

```
## [1] "meanOVEN"
## [1] "meanBAWW"
```

```
for (i in names(birdStatsByClass)) {
  attr(birdStatsByClass[[i]],
       "Species") <- i
T}

#demonstrate what first lines of a birdStatsByClass
# df looks like
head(birdStatsByClass$meanOVEN)
```

```
##    uniqueClasses forestedStatus landForClass classCount meanBirdDensity
## 1:           1_1              1            1         20      0.02948067
## 2:           1_2              1            2       1324      0.03744262
## 3:           1_3              1            3        677      0.06897586
## 4:           1_4              1            4       1860      0.06571962
## 5:           1_5              1            5     765230      0.07852353
## 6:           1_6              1            6      47361      0.05724678
##    varBirdDensity seBirdDensity    normality unimodality
## 1:   2.029391e-05  1.007321e-03 7.052963e-05  0.79139657
## 2:   2.006014e-04  3.892449e-04 3.700000e-24  0.00000000
## 3:   1.810380e-03  1.635274e-03 3.700000e-24  0.02787774
## 4:   5.980507e-04  5.670385e-04 3.700000e-24  0.31195313
## 5:   1.767870e-03  4.806503e-05 3.700000e-24  0.00000000
## 6:   9.193674e-04  1.393266e-04 3.700000e-24  0.00000000
```

Explore the produced 1D bird density predictions

Are the normality and unimodality tests passed?

```r
getAssumptionsSummary <- function(birdStatsTables) {
  namesBirdsAnalysed <- names(birdStatsTables)

  byBirdAssumptions <- lapply(birdStatsTables,
                              FUN = function(x) {
    print(attr(x, "Species"))
    ## TODO: use messages
    flush.console()
    ## TODO: remove

    ## Normality Proportion
    norm <- length(x$normality[!is.na(x$normality)])
    if (norm == "0") {
      propNormal <- NA
    } else {
      noNormal <- sum(!(x$normality[!is.na(x$normality)]) > 0.05)
      totalClassesNorm <- length(x$normality[!is.na(x$normality)])
      propNormal <- noNormal/totalClassesNorm
    }

    ## unimodal proportion
    noUnimodal <- sum(!(x$unimodality > 0.05))
    totalClassesUni <- length(x$unimodality)
    propUnimodal <- noUnimodal/totalClassesUni

    # assumptions <- data.table(propUnimodal)
    assumptions <- data.table(propNormal, propUnimodal)
    return(assumptions)
  })

  names(byBirdAssumptions) <- namesBirdsAnalysed
  numberOfSpecies <- length(byBirdAssumptions)

  birdAssumpTab <- rbindlist(byBirdAssumptions,
                             use.names = TRUE,
                             idcol = "birdSp")

  ## proportion of species where > 50% cover types are normal
  numberOfSpeciesNorm <- length(birdAssumpTab$propNormal[!is.na(birdAssumpTab$propNormal)])
  numberSpNormal <-  sum(!(birdAssumpTab$propNormal[!is.na(birdAssumpTab$propNormal)] < 0.5))
  propSpNormal <- numberSpNormal/numberOfSpeciesNorm

  ## proportion of species where > 50% cover types are unimodal
  numberSpUnimodal <- sum(!(birdAssumpTab$propUnimodal < 0.5))
  propSpUnimodal <- numberSpUnimodal / numberOfSpecies
  birdAssumptions <- data.table(propSpNormal, propSpUnimodal)

  return(birdAssumptions)
}
```

```
assumptionsSummary <- getAssumptionsSummary(birdStatsTables = birdStatsByClass)
```

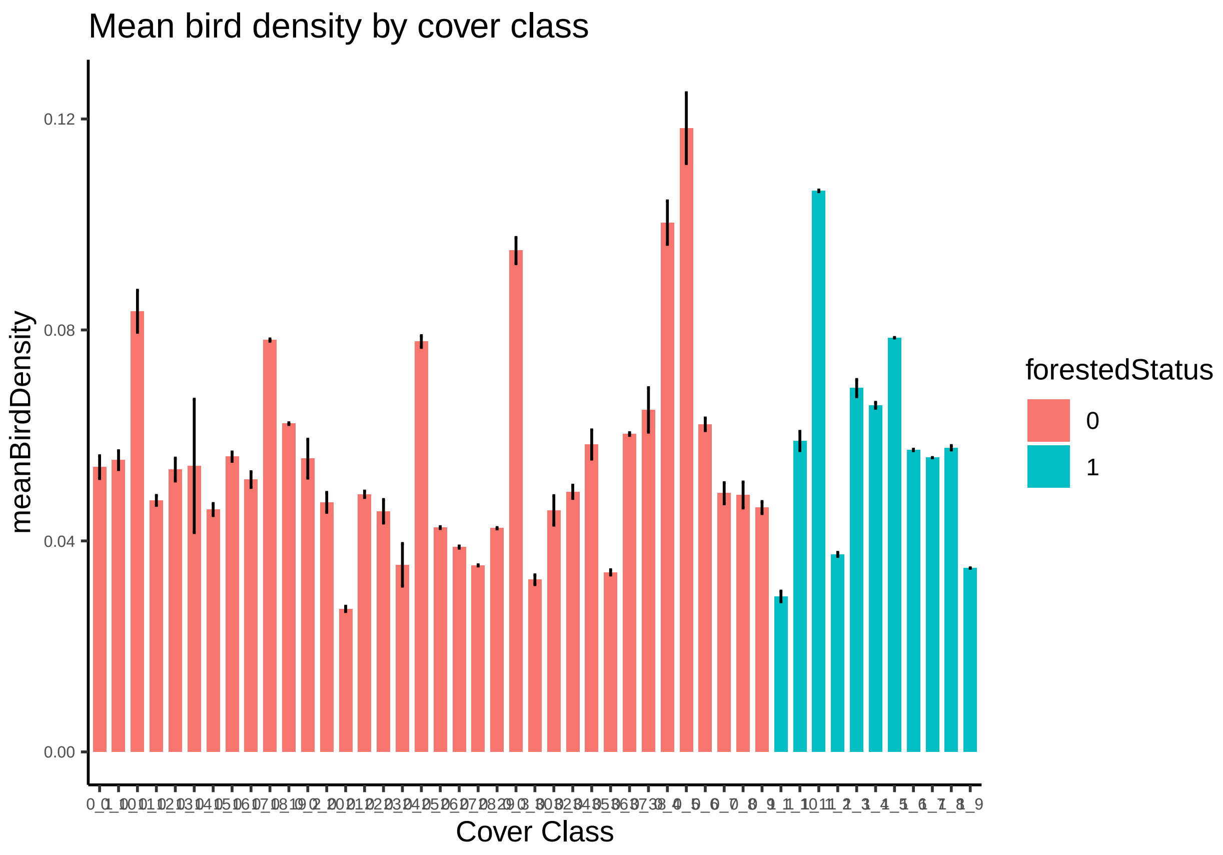
```
## [1] "meanOVEN"
## [1] "meanBAWW"
```

```
head(assumptionsSummary)
```

```
##    propSpNormal propSpUnimodal
## 1:            1              1
```

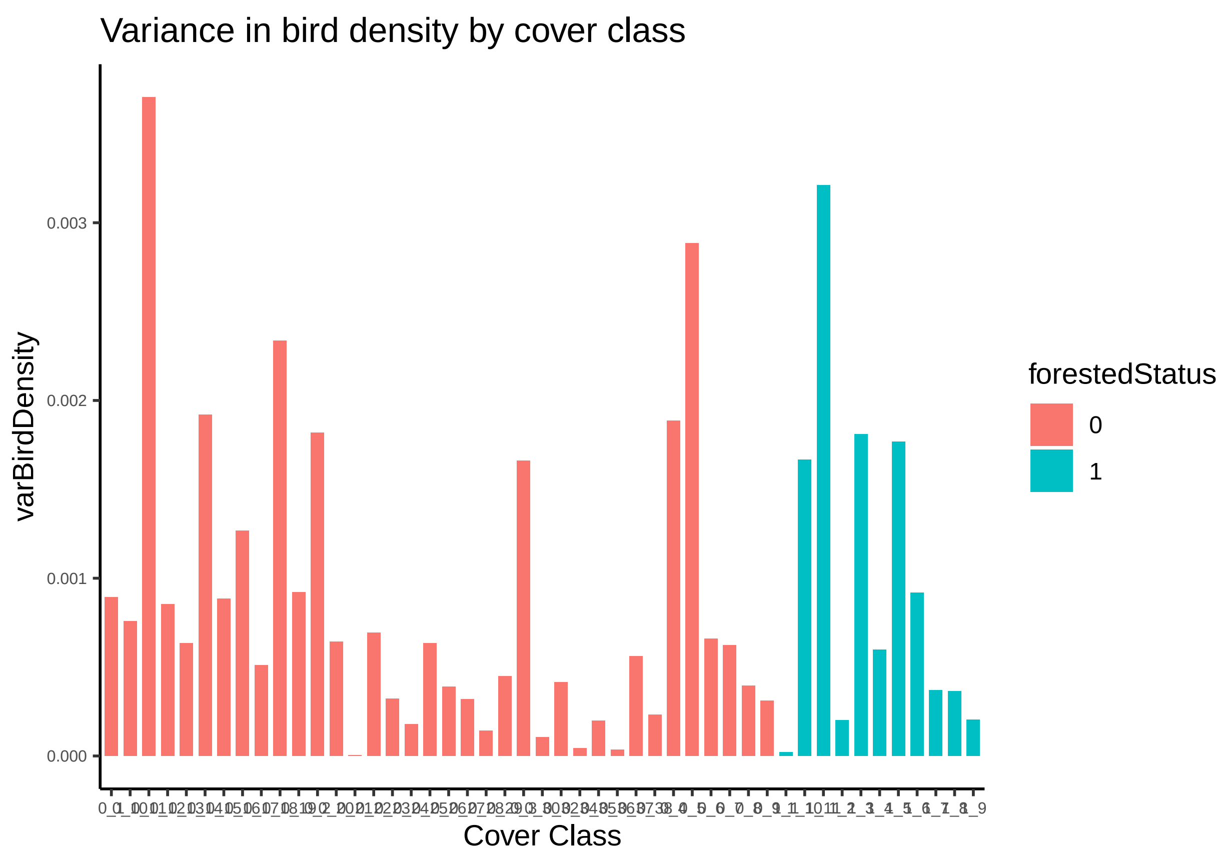
Plot mean bird density by cover class with SE bars

```
plotsMeanBirdDensity <- lapply(X = birdStatsByClass,
                              FUN = function(singleBirdStats){

plotMeanBirdDensity <- ggplot(data = singleBirdStats,
                              aes(x = uniqueClasses,
                                  y = meanBirdDensity,
                                  fill = forestedStatus)) +
    geom_bar(stat = "identity",
             width = 0.7) +
    theme_classic() +
    ggtitle(paste0("Mean bird density by cover class")) +
    xlab("Cover Class") +
    theme(axis.text = element_text(size = 6)) +
    geom_errorbar(aes(ymin = meanBirdDensity - seBirdDensity,
                      ymax = meanBirdDensity + seBirdDensity),
                  width = .15)

return(plotMeanBirdDensity)

})

#show an example
plotsMeanBirdDensity$meanOVEN
```



Plot variance in bird density by cover class

```
plotsVarBirdDensity <- lapply(X = birdStatsByClass,
                              FUN = function(singleBirdStats){

plotVarBirdDensity <- ggplot(data = singleBirdStats,
                             aes(x =uniqueClasses,
                                 y = varBirdDensity,
                                 fill = forestedStatus)) +
   theme_classic() +
  ggtitle(paste0("Variance in bird density by cover class")) +
  xlab("Cover Class") +
    theme(axis.text = element_text(size = 6)) +
  geom_bar(stat = "identity",
           width = 0.7)

return(plotVarBirdDensity)

})

#show an example
plotsVarBirdDensity$meanOVEN
```



Examine the data using a kernel density plot

```
getKernelDensityData <- function(birdDatasets) {
  namesBirdsAnalysed <- names(birdDatasets)

  birdKernelDensities <- lapply(X = birdDatasets,
                                FUN = function(singleBirdDataset) {
    namesClasses <- levels(singleBirdDataset$uniqueClasses)
      classKernelDensities <- lapply(X = namesClasses,
                                     FUN = function(coverType) {
      dataForDensity <- singleBirdDataset[uniqueClasses == coverType]
      singleClassDensity <- density(dataForDensity[,birdDensity])

      return(singleClassDensity)
    })

    names(classKernelDensities) <- namesClasses
    return(classKernelDensities)
  })

  names(birdKernelDensities) <- namesBirdsAnalysed
  return(birdKernelDensities)
}
```

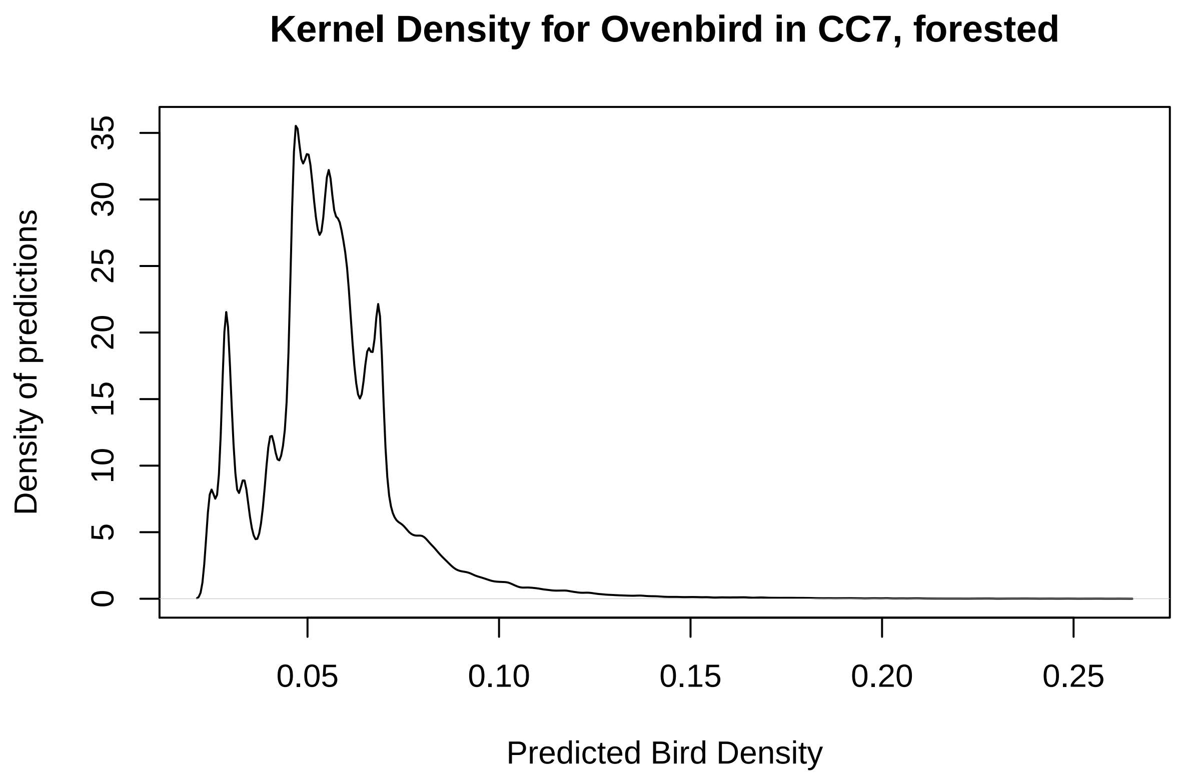
```r
kernelDensityData <- getKernelDensityData(birdDatasets = birdDatasets)
```

Get Kernel Density Data

Plot an Example Kernel Density Data

```r
getKernelDensityPlot <- function(birdCoverDensity,
                                 birdName,
                                 coverType,
                                 meanData) {
  densityPlot <- plot(birdCoverDensity,
                      main = paste0("Kernel Density for ",
                                    paste0(birdName),
                                    " in CC",
                                    coverType),
                      xlab = "Predicted Bird Density",
                      ylab = "Density of predictions")
  return(densityPlot)
}
```

```r
#which kernel density cover
#type and bird species to plot
kernelDensityPlot <- getKernelDensityPlot(birdName = "Ovenbird",
                                          #sp name for title
                                          coverType = "7, forested",
                                          #cover type name for title
                                          birdCoverDensity = kernelDensityData$meanOVEN$"1_7")
```



References

Boreal Avian Modelling Project, 2020. BAM Generalized National Models Documentation, Version 4.0. Available at https://borealbirds.github.io/. DOI: 10.5281/zenodo.4018335.