North Carolina Woodpecker Species Distribution Modeling using Presence-Only Prediction

Benton Tripp

GIS540 Spring 2023 Final Project

Contents

[Synopsis 2](#_Toc133354401)

[Project Keywords 2](#_Toc133354402)

[Abstract 2](#_Toc133354403)

[Project Details 2](#_Toc133354404)

[Running the Analysis 2](#_Toc133354405)

[Mapping 3](#_Toc133354406)

[Graphical User Interface 4](#_Toc133354407)

[Project Data 5](#_Toc133354408)

[Acknowledgements 5](#_Toc133354409)

[Different methods of handling file de-compression 5](#_Toc133354410)

[Reading Data from a Public Google Drive URL 6](#_Toc133354411)

[References 6](#_Toc133354412)

[Appendix I: Project Pseudocode 7](#_Toc133354413)

[Classes 7](#_Toc133354414)

[birds.py 7](#_Toc133354415)

[Functions 7](#_Toc133354416)

[get\_nc\_boundary.py 7](#_Toc133354417)

[get\_bird\_data.py 7](#_Toc133354418)

[process\_bird\_data.py 9](#_Toc133354419)

[get\_land\_cover\_data.py 9](#_Toc133354420)

[get\_dem\_data.py 10](#_Toc133354421)

[get\_weather\_data.py 10](#_Toc133354422)

[presence\_only.py 11](#_Toc133354423)

[presence\_only\_mapping.py 14](#_Toc133354424)

[Workflow 15](#_Toc133354425)

[woodpeckers\_nc.py 15](#_Toc133354426)

[woodpeckers\_nc\_tool.py 16](#_Toc133354427)

# Synopsis

A major challenge in understanding the distribution of species is that observational point data tends to be primarily recorded in locations easily accessible to people. One method of estimating the distribution in a broader area is through a spatial statistics technique called Presence Only Prediction using Maximum Entropy (MaxEnt)1. The purpose of this project is to use the MaxEnt approach to estimate woodpecker species’ distributions in North Carolina from crowd-sourced bird watching dataset.

## Project Keywords

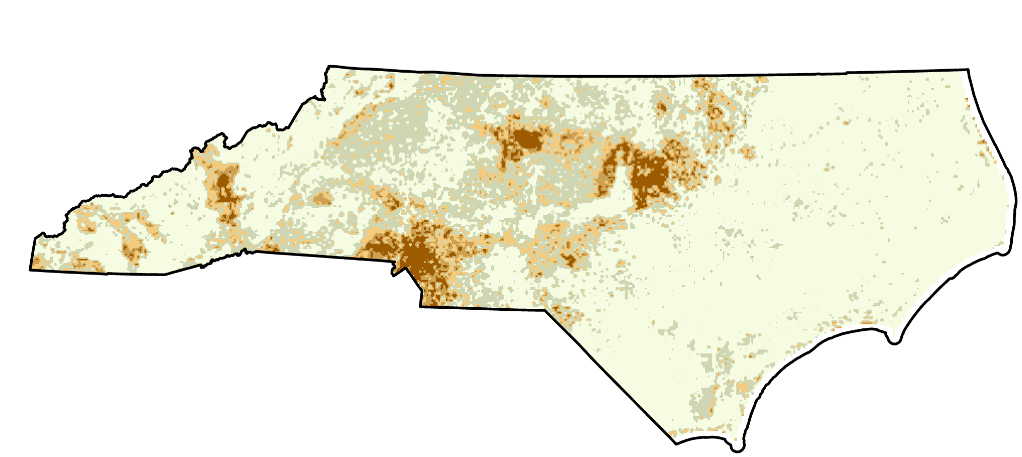
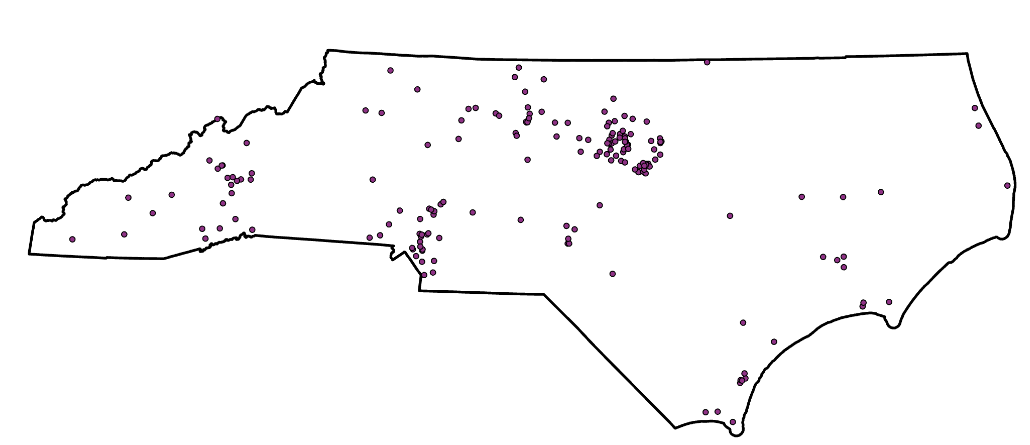
*Species distribution, raster, point data, digital elevation model, land cover, presence-only prediction, MaxEnt, statistics, birds*

# Abstract

Understanding the distribution of species is essential for biodiversity conservation and wildlife management. However, observational point data often has inherent spatial biases due to human accessibility, making it difficult to infer species distributions over a broader landscape. This study implements Presence-Only-Prediction, based on the Maximum Entropy (MaxEnt) modeling approach, to estimate the distribution of bird species in the woodpecker family within North Carolina. The estimates were produced using point data from the crowd-sourced Feederwatch dataset, collected between 2017 and 2019. MaxEnt is a robust spatial statistics technique that accounts for the limitations of presence-only data and has been widely used for species distribution modeling.2-5

To improve the predictive accuracy of the MaxEnt model, explanatory raster data layers were incorporated, including land cover, digital elevation model (DEM), minimum temperature, maximum temperature, and average temperature. These environmental variables provide critical information about the habitat preferences of woodpeckers, influencing their distribution across the study area *(see the Project Data section for additional details)*.

The goal of this study is to provide insights into the spatial distribution of woodpecker species in North Carolina, accounting for potential biases in the Feederwatch dataset. The application of Maximum Entropy modeling in this context not only offers a reliable method for assessing species distributions but also highlights the importance of using crowd-sourced data and advanced spatial analysis techniques in wildlife conservation and management efforts.6,7­­



*Figure 1: Yellow-Bellied Sapsucker estimated distribution in North Carolina, output as a raster from 2017-2019 Feederwatch observation data*

# Project Details

## Running the Analysis

Prior to running the analysis, note some important details to be aware of:

1. There is simply a lot of data. It's unavoidable.
2. If the project is run with missing data it will automatically be downloaded, processed, and added to the geodatabase. The data is downloaded from several sources, and so depending on your internet speed this can take a while. Note that you also must be connected to the NCSU network to download the explanatory DEM data. After data has been saved to the project directory/database, it will no longer be downloaded each time the project is run unless it is removed.
3. The actual analysis is pretty computationally heavy. Depending on what the grid-search parameters are set to, it can take a long time to test all of the potential models for each of the species.
4. Each of the unique parameters attempted are saved to a “model log”, along with the associated performance metrics. If the same parameters are applied again (to the same species), the code will skip over those parameters and refer to the model log. To force model training for all input parameters, the model log can be removed from woodpeckerNC/data/model\_data/model\_training\_logs.

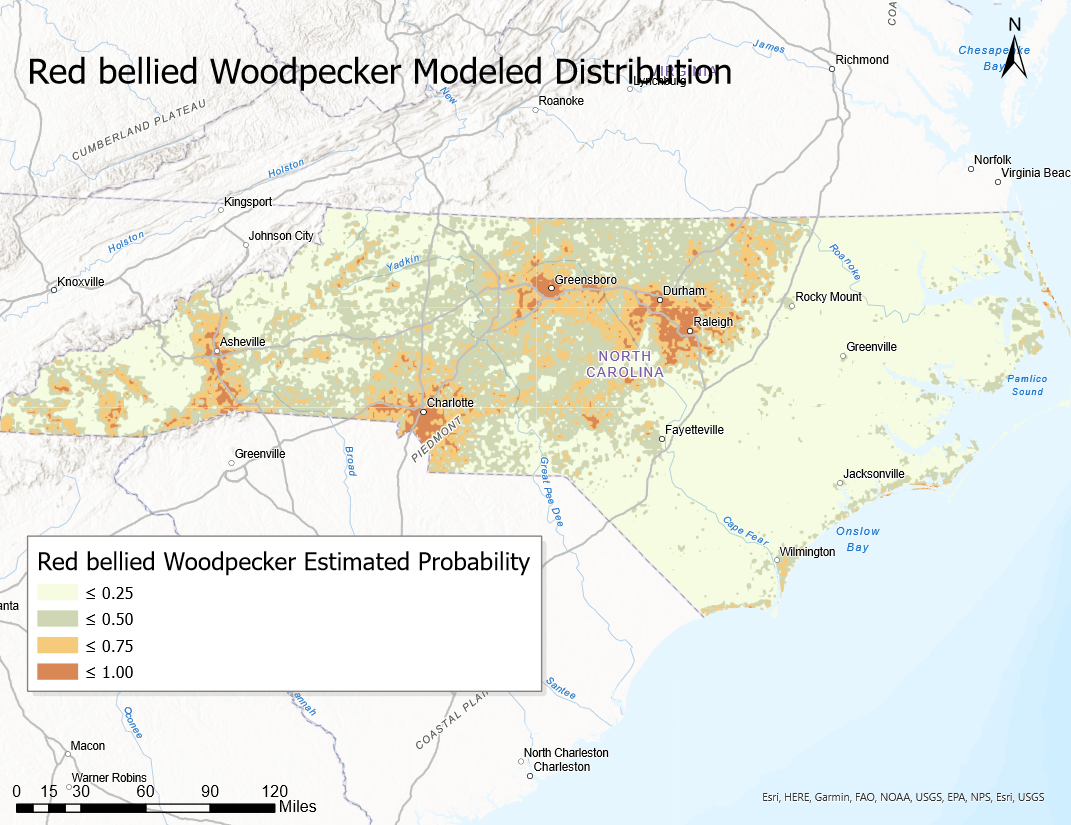
To run, first clone/download the project into your desired location and navigate to the woodpeckerNC directory. Activate a python environment with arcpy, numpy, and pandas installed (you will need ArcGIS Pro installed on your machine to use arcpy). Then run:

python path/to/woodpeckerNC/woodpeckers\_nc.py path/to/woodpeckerNC/woodpeckerNC.aprx

There is an additional script woodpeckers\_nc\_tool.py that is linked to a script tool in woodpeckerNC.atbx. This is essentially the same as running the command provided above. However, it allows the user to customize the parameters that should be tested when training the models using the grid-search algorithm (*for more details, see the Graphical User Interface section*).

## Mapping

Upon the completion of model training, the final rasters output by the arcpy’s PresenceOnlyPrediction are added to the map. In addition, the color scheme and labels are updated to help users to understand the output. Finally, each map layer (i.e., woodpecker species) is shown on the Layout and layout elements are updated accordingly. The layout is output to a pdf (by default the pdfs are saved to a folder created within the data directory, but the GUI has an option where this location can be changed) for each respective species.

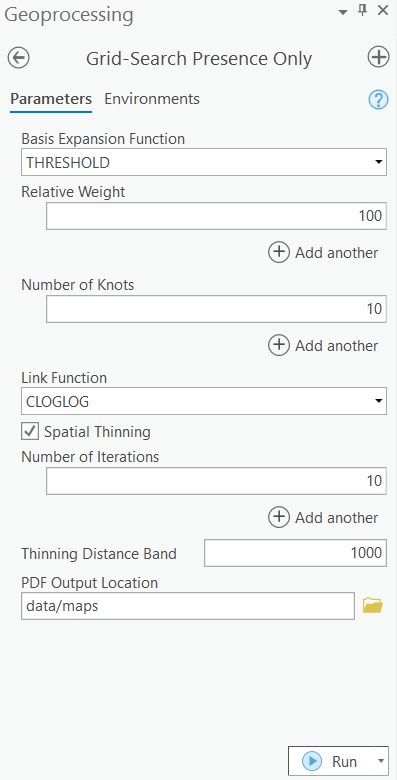
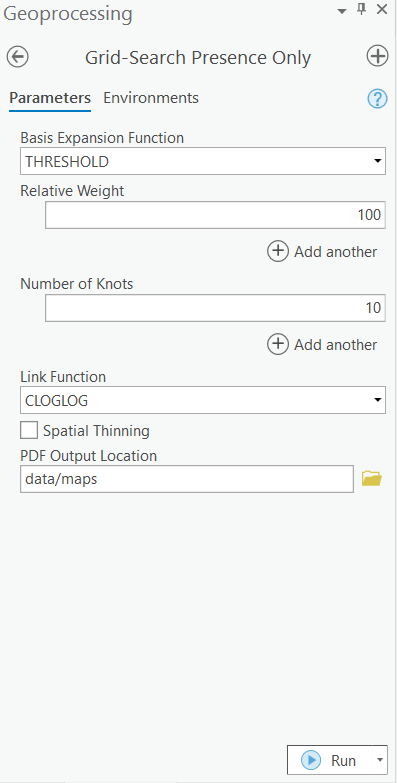


*Figure 2: PDF Output example for the Red-Bellied Woodpecker*

## Graphical User Interface

ArcGIS Pro has a Script Tool feature that enables users to easily link Python/R scripts to a graphical user interface (GUI), where input/output values can be manually manipulated instead of updated within the script or via the command line. For this project, a tool was created that runs the full data downloading, processing, modeling, and mapping all together. The parameters that can be manipulated are primarily hyperparameter values for the MaxEnt model, with the option to select multiple for several of these parameters and run a grid-search tuning algorithm. In addition, the output path for the PDF reports can be specified.

Note that the number of parameters visible to a user varies depending on the Spatial Thinning parameter selection, as some parameters are only relevant when it is set to True (*see Figure 3 below*).

*Figure 3: (left) Grid-Search Presence Only GUI with Spatial Thinning enabled; (right) Grid-Search Presence Only GUI with Spatial Thinning disabled*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Parameter** | **Data Type** | **Type** | **Direction** | **Default Value** | **Filter** |
| Basis Expansion Function | String | Required | Input | THRESHOLD | Value List |
| Relative Weight | [Long] | Required | Input | 100 | Range |
| Number of Knots | [Long] | Required | Input | 10 | Range |
| Link Function | String | Required | Input | CLOGLOG | Value List |
| Spatial Thinning | Boolean | Required | Input | True |  |
| Number of Iterations | [Long] | Required | Input | 10 | Range |
| Thinning Distance Band | Long | Required | Input | 1000 | Range |
| PDF Output Location | Folder | Optional | Input | data/maps |  |

*Table 1: Grid-Search Presence Only parameter summary*

## Project Data

The data used in this project are from four primary sources:

* 1. Project FeederWatch8 – Bird observation data, compiled by The Cornell Lab of Ornithology and Birds Canada. There are two file formats for the data: (1) *.xlsx* and (2) *.csv*. The Excel file contains the “data dictionary”, and is primarily needed so that the common names of the woodpecker species can be joined with the *.csv* data (in which they are instead identified by a unique ID). The *.csv* data contains the relevant data needed for this project including the species, date observed, and the location observed. The data can be obtained directly from the Project FeederWatch website via the dataset requests page: <https://feederwatch.org/explore/raw-dataset-requests/>.
  2. National Land Cover Database9 – using data from the Multi-Resolution Land Characteristics (MRLC) Consortium10, North Carolina State University created the NLCD. The data used in this project is clipped/re-projected land cover data for the state of North Carolina. <https://gisdata.lib.ncsu.edu/fedgov/mrlc/nlcd2019/NC_NLCD2019only.zip>
  3. North Carolina Digital Elevation Model (DEM)11 – this data was acquired directly through the NCSU GIS library at: <https://gisdata.lib.ncsu.edu/DEM/nc250.zip>. The data is a raster image of North Carolina elevation at a 1:250,000 scale.
  4. Prism Weather Data12 – this weather raster data is available for free to the public at a 4km grid resolution for the full United states for several different weather-related variables (used in this project are minimum temperature, maximum temperature, and average precipitation) aggregated at a yearly level. Also available for free is 30-year normalized monthly data at an 800m grid resolution. Both of these data types were used in this project. Data documentation can be found at <https://www.prism.oregonstate.edu/documents/PRISM_downloads_web_service.pdf>.

# Acknowledgements

## Different methods of handling file de-compression

Decompressing files using Python was discussed within the GIS540 coursework (*see Dr.* *Laura Tateosian’s Python for ArcGIS, 399-400*). However, the method taught in the course material involves first downloading the archived files, and then decompressing them. Although the method demonstrated in the course is used within the project, an alternative solution is also implemented to improve performance and declutter the workflow. The solution was proposed by Shyamal Vaderia in a blog post, where he explains the process of decompressing a file from a url directly into the specified location without the need to download the compressed file first. His blog post on the subject can be found at <https://svaderia.github.io/articles/downloading-and-unzipping-a-zipfile/>. Below is a code snippet of this method being applied within the land\_cover\_data.py script:

from io import BytesIO

from urllib.request import urlopen

from zipfile import ZipFile

…

# Credit to this method of unzipping a zip file goes to Shyamal Vaderia

# (see blog post at https://svaderia.github.io/articles/downloading-and-unzipping-a-zipfile/)

with urlopen(zipurl) as zipresp:

with ZipFile(BytesIO(zipresp.read())) as zfile:

zfile.extractall(raster\_path)

# End Credit

## Reading Data from a Public Google Drive URL

Data cannot be downloaded directly from a traditional Google Drive url that opens up the data within a web browser. However, some adjustments to the url can enable a direct download. The following snippet is an implementation of a solution to this problem provided by Bhishan Poudel in Stackoverflow (*Refer to the full post at* [*https://stackoverflow.com/questions/56611698/pandas-how-to-read-csv-file-from-google-drive-public*](https://stackoverflow.com/questions/56611698/pandas-how-to-read-csv-file-from-google-drive-public)):

# Credit goes to the following StackOverflow answer for re-formatting the url:

# https://stackoverflow.com/questions/56611698/pandas-how-to-read-csv-file-from-google-drive-public

# First, set up the url for the data dictionary (Google Drive).

url = 'https://drive.google.com/file/d/1kHmx2XhA2MJtEyTNMpwqTQEnoa9M7Il2/view?usp=sharing'

url = 'https://drive.google.com/uc?id=' + url.split('/')[-2]

# Read the Excel Sheet with the Species Codes

species = pd.read\_excel(url, sheet\_name='Species Codes', header=1)

# End credit

This code snippet can be found in the get\_bird\_data.py script as a solution to downloading the Feederwatch data dictionary.

# References

1. ESRI. (n.d.). Presence Only Prediction (Spatial Statistics). ArcGIS Pro. Retrieved from <https://pro.arcgis.com/en/pro-app/latest/tool-reference/spatial-statistics/presence-only-prediction.htm>
2. Elith, J., Phillips, S. J., Hastie, T., Dudík, M., Chee, Y. E., & Yates, C. J. (2011). A statistical explanation of MaxEnt for ecologists. Diversity and Distributions, 17(1), 43-57. <https://doi.org/10.1111/j.1472-4642.2010.00725.x>
3. Phillips, S. J., Anderson, R. P., & Schapire, R. E. (2006). Maximum entropy modeling of species geographic distributions. Ecological Modelling, 190(3-4), 231-259. <https://doi.org/10.1016/j.ecolmodel.2005.03.026>
4. Merow, C., Smith, M. J., & Silander, J. A. Jr. (2013). A practical guide to MaxEnt for modeling species’ distributions: what it does, and why inputs and settings matter. Ecography, 36(10), 1058-1069. <https://doi.org/10.1111/j.1600-0587.2013.07872.x>
5. Yackulic, C. B., Chandler, R., Zipkin, E. F., Royle, J. A., Nichols, J. D., Campbell Grant, E. H., & Veran, S. (2013). Presence-only modelling using MAXENT: when can we trust the inferences? Methods in Ecology and Evolution, 4(3), 236-243. <https://doi.org/10.1111/2041-210x.12004>
6. Wisz, M. S., Hijmans, R. J., Li, J., Peterson, A. T., Graham, C. H., & Guisan, A. (2008). Effects of sample size on the performance of species distribution models. Diversity and Distributions, 14(5), 763-773. <https://doi.org/10.1111/j.1472-4642.2008.00482.x>
7. Sullivan, B. L., Wood, C. L., Iliff, M. J., Bonney, R. E., Fink, D., & Kelling, S. (2009). eBird: A citizen-based bird observation network in the biological sciences. Biological Conservation, 142(10), 2282-2292. <https://doi.org/10.1016/j.biocon.2009.05.006>
8. FeederWatch. (n.d.). Project Overview. Retrieved from <https://feederwatch.org/about/project-overview/>
9. North Carolina State University Libraries. (n.d.). National Land Cover Database (NLCD). Retrieved from <https://www.lib.ncsu.edu/gis/nlcd>
10. Multi-Resolution Land Characteristics Consortium (MRLC). (n.d.). MRLC Home. Retrieved from <https://www.mrlc.gov/>
11. North Carolina State University Libraries. (n.d.). Digital Elevation Models (DEM). Retrieved from <https://www.lib.ncsu.edu/gis/dem>
12. PRISM Climate Group. (n.d.). PRISM Climate Data. Oregon State University. Retrieved from <https://prism.oregonstate.edu/>

# Appendix I: Project Pseudocode

## Classes

### birds.py

class Species:

FUNC \_\_init\_\_(self,

dataframe):

SET species code attribute as unique species codes in dataframe

SET species name attribute as unique species names in dataframe

SET family attribute as unique families in dataframe

ENDFUNC

class Bird(Species):

FUNC \_\_init\_\_(self,

dataframe,

bird name,

prefix):

Inherit from Species (super)

SET code attribute

SET name attribute

SET family attribute

SET name parts

SET formatted name attribute from name parts

SET feature class name attribute from formatted name attribute

ENDFUNC

## Functions

### get\_nc\_boundary.py

FUNC getNCBoundary(data path,

workspace,

coordinate system,

feature class name):

IF NOT valid workspace or NOT valid data path THEN

Raise error

ENDIF

SET environment workspace as workspace

IF NC Boundary Shape File NOT in data path THEN

GET NC Boundary Shape File

ENDIF

IF feature class name NOT workspace THEN

SET NC Boundary Shape File Projection

Dissolve Projected NC Boundary Shape File

SAVE NC Boundary Shape File to workspace as feature class name

END IF

RETURN feature class name

ENDFUNC

### get\_bird\_data.py

FUNC getSpeciesCodes(data path)

IF NOT valid data path THEN

Raise error

ENDIF

SET out file name

IF out file name in data path THEN

GET species dataframe from out file

ELSE

SET url

GET species dataframe from url

Rename species dataframe fields

SAVE species to out file

ENDIF

RETURN species dataframe

ENDFUNC

FUNC cleanFeederWatchData(data,

birds,

sub-national codes):

SET all field names

SET field names to be kept

Preprocess field names

FILTER data

valid is equal to 1, plus code NOT equal to 1, species code in bird species code

END FILTER

IF sub\_national\_code is not None THEN

FILTER data

sub\_national\_code is in sub-national codes

END FILTER

ENDIF

Join data with birds

Format dates

Sort by date and species names

RETURN DATA

END FUNC

FUNC getFeedWatcherData(out file,

timeframes,

birds,

sub-national codes

output directory,

file suffix,

save,

minimum year,

maximum year):

SET final out file as output directory, out file

IF final out file exists THEN

GET single dataframe from final out file

ELSE

SET dataframe list as empty list

FOR timeframe in timeframes

SET new out file name from timeframe and file suffix

IF new out file does not exist THEN

SET url using timeframe

GET data from url

CALL cleanFeederWatchData(data, birds, sub-national codes)

IF save THEN

SAVE data to new out file

ENDIF

ELSE

GET data from new out file

ENDIF

Append data to dataframe list

ENDFOR

Concatenate dataframe list into single dataframe

FILTER single dataframe

Year >= minimum year and Year <= maximum year

END FILTER

SAVE single dataframe to final out file

ENDIF

RETURN single dataframe

ENDFUNC

### process\_bird\_data.py

FUNC batchBirdProcessing(feederwatch file,

base feature class name,

existing feature class list,

coordinate system,

data path,

workspace,

feederwatch dataframe,

species dataframe,

NC State boundary,

Prefix):

SET feederwatch file as joined data path, feederwatch file

IF NOT valid workspace or NOT valid data path or NOT valid feederwatch file THEN

Raise error

ENDIF

SET environment workspace as workspace

IF base feature class name not in existing feature class list THEN

SAVE feederwatch file as base feature class name in workspace

SET base feature class as base feature class projected to coordinate system

Select intersection of base feature class with NC State boundary

ENDIF

FOR species name in distinct names in feederwatch dataframe

SET brd as CALL class Bird(species dataframe, name, prefix)

IF brd feature class name attribute NOT in existing feature classes THEN

Select by species name from base feature class

SAVE selection to workspace as brd feature class name

ENDIF

ENDFOR

ENDFUNC

### get\_land\_cover\_data.py

FUNC getLandCoverData(data path,

workspace,

coordinate system,

NC State boundary,

output feature class name):

IF NOT valid workspace or NOT valid data path THEN

Raise error

ENDIF

SET environment workspace as workspace

SET raster data path in data path

SET land cover url

IF raster data path does not exist THEN

GET raster data from land cover url

Decompress data

ENDIF

IF output feature class name not in workspace raster list THEN

Resample cell size of raster data

Reproject resampled raster data to coordinate system

Mask raster data to NC State boundary extent

SAVE raster data to workspace as output feature class name

ENDIF

RETURN output feature class name

ENDFUNC

### get\_dem\_data.py

FUNC getDEMData(data path,

workspace,

coordinate system,

NC State boundary,

Output feature class name):

IF NOT valid workspace or NOT valid data path THEN

Raise error

ENDIF

SET environment workspace as workspace

SET elevation data path from data path

IF elevation data path does not exist THEN

Create elevation data path

ENDIF

IF elevation data not in elevation data path THEN

SET elevation data url

GET compressed elevation data from url

Decompress elevation data and save to elevation data path

ENDIF

IF output feature class name not in workspace raster list THEN

Reproject elevation data to coordinate system

Mask elevation data to NC State boundary extent

SAVE elevation data to workspace as output feature class name

ENDIF

RETURN output feature class name

ENDFUNC

### get\_weather\_data.py

FUNC getRastersFromDir(variable name,

time periods,

output path):

SET output list as empty list

FOR period in time periods

SET file path from output path, var, and period

FOR each file in file path

IF file is a raster THEN

Append file name to output list

ENDIF

ENDFOR

ENDFOR

RETURN output list

ENDFUNC

FUNC getWeatherData(data path,

workspace,

NC State boundary,

coordinate system,

average precipitation name,

min temperature name,

max temperature name):

IF NOT valid workspace or NOT valid data path THEN

raise error

ENDIF

SET environment workspace as workspace

SET variables as list (precipitation, min temperature, max temperature)

SET years as list (2017-2019)

SET variable year pairs as list of unique variable year pairs

SET months as list of months (1-12)

SET variable month pairs as list of unique variable month pairs

SET output path from data path

FOR variable and year in variable year pairs

SET download output filename from variable, year

IF download output filename does NOT exist

SET url from variable, year

GET variable year data from url

ENDIF

ENDFOR

FOR variable and month in variable month pairs

SET download output filename from variable, month

IF download output filename does NOT exist

SET url from variable, month

GET variable month data from url

ENDIF

ENDFOR

FOR variable in variables

SET raster name from variable value

IF raster name not in workspace rasters THEN

SET aggregated rasters as empty dictionary

SET rasters from CALL getRastersFromDir(variables, years, output path)

SET raster out name

SET aggregate function from variable value

Update aggregate rasters with variable, year, raster out

Aggregate rasters of type variable, year

SAVE aggregated rasters to workspace as raster out

SET rasters from CALL getRastersFromDir(variables, months, output path)

SET raster out name

Update aggregate rasters with variable, month, raster out

Aggregate rasters of type variable, month

SAVE aggregated rasters to workspace as raster out

Project aggregated rasters to coordinate system

Resample year rasters to size of month rasters

Combine variable, year and variable, month rasters using weighted average

Mask combined rasters with NC State boundary

SAVE masked combined rasters to raster name in workspace

ENDIF

ENDFOR

SET output list as (average precipitation, min temperature, max temperature)

RETURN output list

ENDFUNC

### presence\_only.py

FUNC getPrecision(True Positive,

False Positive):

    IF True Positive + False Positive is 0 THEN

RETURN negative infinity

ELSE

        RETURN True Positive / (True Positive + False Positive)

ENDIF

ENDFUNC

FUNC getF1(True Positive,

False Positive,

Recall):

SET denominator 1 as (True Positive + False Positive))

IF denominator 1 is 0 THEN

RETURN negative infinity

ENDIF

SET denominator 2 as ((True Positive / denominator 1) + recall)

IF denominator 2 is 0 THEN

RETURN negative infinity

ELSE

RETURN 2 \* ((True Positive / denominator 1) \* recall) / denominator 2

ENDIF

ENDFUNC

FUNC scoreFromSensitivityTable(sensitivity table):

SET value dictionary as empty dictionary

SET cursor from sensitivity table

FOR row in cursor

GET values from row

SAVE values to value dictionary

ENDFOR

SAVE value dictionary to dataframe

    RETURN dataframe

ENDFUNC

FUNC logModel(parameters,

Log file):

IF log file is does NOT exist THEN

SAVE empty list as log file

ENDIF

GET content from log file

Append parameters to content

SAVE content to log file

ENDFUNC

FUNC checkModelParams(file dictionary,

parameters):

    FOR file in file dictionary

FOR parameter in parameters

IF file parameter is equal to parameter THEN

RETURN True

ENDIF

ENDFOR

ENDFOR

RETURN False

ENDFUNC

FUNC checkModelLogs(log path,

parameters,

species):

    SET files as list of files in log path like species

    FOR file in files

        SET file path from log path, file

        GET file dictionary from file path

            IF CALL checkModelParams(file dictionary, parameters) THEN

                RETURN False

ENDIF

ENDFOR

    RETURN True

ENDFUNC

FUNC getAllCombos(parameter grid):

    IF "NO\_THINNING" not in spatial thinning value in parameter grid THEN

        SET all combinations as cross joined parameter grid

    ELSEIF "THINNING" not in spatial thinning value in parameter grid THEN

        SET all combinations as cross joined parameter grid

Exclude spatial thinning parameters from all combinations

    ELSE

        SET no thinning combinations as cross joined parameter grid

SET thinning combinations as cross joined parameter grid

Exclude spatial thinning parameters from thinning combinations

SET all combinations as combined no thinning and thinning combinations

ENDIF

RETURN all combinations

ENDFUNC

FUNC runMaxEnt(static parameters,

              parameters,

              output dictionary,

              output boolean):

IF output boolean is False THEN

        FOR key in outputs dictionary keys

            IF k is NOT equal to output sensitivity table key THEN

                SET output value in output dictionary at key to None

ENDIF

ENDFOR

ENDIF

    SET result as CALL presence only prediction using all three dictionary inputs

RETURN result

ENDFUNC

FUNC batchMaxEnt(species dataframe,

                workspace,

                Data path,

                Explanatory rasters,

                NC State boundary,

parameter grid):

IF NOT valid workspace or NOT valid data path THEN

Raise error

ENDIF

    SET environment workspace as workspace

    SET model data path

IF model data path does NOT exist THEN

Create model data path

ENDIF

SET log path

IF log path does NOT exist THEN

Create log path

ENDIF

FOR species name in distinct species names from species dataframe

        SET bird as CALL Bird(species dataframe, species name)

        SET species from formatted name attribute of bird

SET trained features name

        IF trained features name not in workspace feature classes THEN

            SET static parameters

            SET all combinations as CALL getAllCombos(parameter grid)

            SET files as list of files in model data path

SET species model file name

            IF species model file name in files THEN

                GET species model

SET best combination from species model

SET best F1 score from species model

ELSE

SET best combination to None

Set best F1 score to 0.0

ENDIF

            FOR combination in all combinations

                SET parameters as dictionary of parameter grid key/combinations pairs

                IF CALL checkModelLogs(log path, parameters, species)THEN

SET output dictionary

                    CALL runMaxEnt(static parameters,

parameters,

output dictionary,

output = False)

SET trained sensitivity table

                    SET score as CALL scoreFromSensitivityTable(trained sensitivity table)

                    DELETE trained sensitivity table

SET F1 score, cutoff

Update parameters with F1 score and cutoff values

SET log file name

                    CALL logModel(parameters, log file name)

                    IF F1 score > best F1 score THEN

                        SET best F1 score as F1 score

                        SET best combination to combination

                        SAVE species model to a pickle file

ENDIF

ENDIF

ENDFOR

            GET species model for species

            CALL runMaxEnt(species model saved static parameters,

species model saved parameters,

output dictionary,

output = True)

ENDIF

ENDFOR

ENDFUNC

### presence\_only\_mapping.py

FUNC hexToRGB(hex color):

COMPUTE RGB from hex color

RETURN RGB

ENDFUNC

FUNC createMapAndExport(project path,

workspace,

bird rasters,

output folder):

SET environment workspace as workspace

SET project from project path

FOR raster in bird rasters

GET species name from bird rasters

Create raster layer from raster

Update raster layer color symbology

GET map from project maps

Add raster layer to map

GET project layout

Update title, map frame, legend for layout

SET pdf path

SAVE layout to pdf path

ENDFOR

SAVE project

ENDFUNC

FUNC outputMaxEntMaps (species dataframe,

project path,

workspace,

data path,

output folder):

IF NOT valid workspace or NOT valid data path or NOT valid project path THEN

raise error

ENDIF

SET environment workspace as workspace

SET output folder

IF output folder does NOT exist THEN

Create output folder

ENDIF

GET trained rasters from workspace

SET bird raster/species name key/value pairs

CALL createMapAndExport(project path, workspace, bird rasters, output folder)

ENDFUNC

## Workflow

### woodpeckers\_nc.py

Import libraries and modules

GET project path from argument 1

SET project from project path

GET default database from project

SET data path, suffix, prefix, base feature class name, feederwatch file name

IF default database does NOT exist THEN

Create default database

ENDIF

SET environment workspace as default database

GET existing feature classes from workspace

SET coordinate system

GET NC State boundary from CALL getNCBoundary(data path, workspace, coordinate system)

SET data timeframes, sub-national code, save, minimum year, maximum year

GET species from getSpeciesCodes(data path)

GET woodpeckers from species

GET feederwatch data from CALL getFeederWatch(feederwatch file name,

data timeframes, woodpeckers,

sub-national code, data path,

suffix,

save,

minimum year,

maximum year)

CALL batchBirdProcessing(feederwatch file name,

base feature class name,

existing feature classes,

coordinate system,

data path,

workspace,

feederwatch data,

woodpeckers,

prefix,

NC State boundary)

GET land cover data from CALL getLandCoverData(data path,

workspace,

coordinate system,

NC State boundary)

GET DEM data from CALL getDEMData(data path,

workspace,

coordinate system,

NC State boundary)

GET average precipitation, minimum temperature, max temperature

from CALL getWeatherData(data path, workspace, NC State boundary, coordinate system)

GET NC woodpeckers from woodpeckers and sub-national code

SET explanatory raster names

CALL batchMaxEnt(NC woodpeckers,

workspace,

data path,

explanatory raster names,

NC State boundary)

CALL outputMaxEntMaps(NC woodpeckers,

project path,

workspace,

data path,

output folder)

### woodpeckers\_nc\_tool.py

Import libraries and modules

GET project from current project

GET project path from project

GET default database from project

SET data path, prefix, suffix, base feature class name, feederwatch file

GET parameter grid from arguments 1-7 (dictionary values)

GET pdf output location from argument 8

SET environment workspace as default database

GET existing feature classes from workspace

SET coordinate system

GET NC State boundary from CALL getNCBoundary(data path, workspace, coordinate system)

SET data timeframes, sub-national code, save, minimum year, maximum year

GET species from getSpeciesCodes(data path)

GET woodpeckers from species

GET feederwatch data from CALL getFeederWatch(feederwatch file name,

data timeframes, woodpeckers,

sub-national code, data path,

suffix,

save,

minimum year,

maximum year)

CALL batchBirdProcessing(feederwatch file name,

base feature class name,

existing feature classes,

coordinate system,

data path,

workspace,

feederwatch data,

woodpeckers,

prefix,

NC State boundary)

GET land cover data from CALL getLandCoverData(data path,

workspace,

coordinate system,

NC State boundary)

GET DEM data from CALL getDEMData(data path,

workspace,

coordinate system,

NC State boundary)

GET average precipitation, minimum temperature, max temperature

from CALL getWeatherData(data path, workspace, NC State boundary, coordinate system)

GET NC woodpeckers from woodpeckers and sub-national code

SET explanatory raster names

CALL batchMaxEnt(NC woodpeckers,

workspace,

data path,

explanatory raster names,

NC State boundary,

parameter grid)

CALL outputMaxEntMaps(NC woodpeckers,

project path,

workspace,

data path,

pdf output location)