Visual Analytics of the Impacts of Climate Change on Migratory Bird Habitats
Technical Document

Jacob Vogt, Mihika Krishna, Catherine Kang, Hangyul Yun

Professor Dongyu Liu

University of California, Davis

June 2024

Contents

Introduction

Our Senior Capstone project is comprised of two components:

1. A species distribution model (SDM) capable of predicting how climate change will affect future bird habitats up to year 2100

2. A web-app that visualizes SDM output and displays other relevant information such as bird migration patterns and climate trends.

This technical document will overview how each of these components work, as well as the required data and file structure for them to operate correctly.

1 Species Distribution Model

1.1 Datasets

The species distribution model (SDM) uses two datasets, one for climate data and one for species distribution data.

Climate Data

NEX-GDDP-CMIP6 datasets are granular by year and offered for four (future) Shared Socioeconomic Pathways (SSP). These dastasets are offered in NETCDF (.nc) format. Descriptions of each SSP can be found at this link.

Climate scenarios used were from the NEX-GDDP-CMIP6 dataset, prepared by the Climate Analytics Group and NASA Ames Research Center using the NASA Earth Exchange and distributed by the NASA Center for Climate Simulation (NCCS).

Thrasher, B., Wang, W., Michaelis, A. et al. NASA Global Daily Downscaled Projections, CMIP6. Sci Data 9, 262 (2022). https://doi.org/10.1038/s41597-022-01393-4

Thrasher, B., Wang, W., Michaelis, A. Nemani, R. (2021). NEX-GDDP-CMIP6. NASA Center for Climate Simulation. https://doi.org/10.7917/OFSG3345

Species Distribution Data

The occurrence datasets for our bird species come from the Global Biodiversity Information Facility (GBIF). The citations for each dataset is below.

- *Numenius americanus Bechstein, 1812 in GBIF Secretariat (2023). GBIF Backbone Taxonomy. Checklist dataset https://doi.org/10.15468/39omei accessed via GBIF.org on 2024-06-09.*

- *Setophaga striata (J.R. Forster, 1772) in GBIF Secretariat (2023). GBIF Backbone Taxonomy. Checklist dataset* *https://doi.org/10.15468/39omei* *accessed via GBIF.org on 2024-06-09.*

- *Anser albifrons (Scopoli, 1769) in GBIF Secretariat (2023). GBIF Backbone Taxonomy. Checklist dataset* *https://doi.org/10.15468/39omei* *accessed via GBIF.org on 2024-06-09.*

- *Haliaeetus leucocephalus (Linnaeus, 1766) in GBIF Secretariat (2023). GBIF Backbone Taxonomy. Checklist dataset* *https://doi.org/10.15468/39omei* *accessed via GBIF.org on 2024-06-09.*

- *Numenius phaeopus subsp. phaeopus in GBIF Secretariat (2023). GBIF Backbone Taxonomy. Checklist dataset* *https://doi.org/10.15468/39omei* *accessed via GBIF.org on 2024-06-09.*

GBIF datasets are downloaded in .csv format.

1.2 R Preprocessing

Pre-processing is necessary to transform raw datasets into a form factor that is usable by our SDM. This pre-processing step uses R due to its specialty in working with biological and spatial data.

There are two different R scripts, one for preprocessing training data and another for preprocessing prediction data. These scripts are named `preprocess_train.R` and `preprocess_predict.R` respectively.

Packages

- *dismo*: Methods for species distribution modeling, that is, predicting the environmental similarity of any site to that of the locations of known occurrences of a species.

- *sf*: Support for simple features, a standardized way to encode spatial vector data. Binds to 'GDAL' for reading and writing data, to 'GEOS' for geometrical operations, and to 'PROJ' for projection conversions and datum transformations. Uses by default the 's2' package for spherical geometry operations on ellipsoidal (long/lat) coordinates.

- *raster*: Reading, writing, manipulating, analyzing and modeling of spatial data.

- *ncdf4*: Provides a high-level R interface to data files written using Unidata's netCDF library (version 4 or earlier), which are binary data files that are portable across platforms and include metadata information in addition to the data sets.

Prerequisites

The R scripts were developed using the latest version of R as of June 9, 2024 (v4.4.0). Different versions of R including v4.4.0 can be found at https://cran.r-project.org/bin/windows/base/old/. Versions older than v4.3.2 are not guaranteed to work.

RStudio is recommended to run the R scripts. Running the scripts from an independent R terminal could work in theory, but is not tested. RStudio can be installed from https://posit.co/download/rstudio-desktop/

The above packages should either be installed with RStudio's built-in package installer or by running the command `install.packages(c("sf", "raster", "dismo", "ncdf4"))` in an R terminal.

Both of these scripts require datasets that can be downloaded from the Box folder at https://ucdavis.box.com/s/l5iky1y6z526r6ewifvtr6c4ccp5jsjz. The `data` folder from the above link should be placed in the root directory of the repository.

Running the Scripts

If the above prerequisites are met, then the R scripts should run without issue. To do this in RStudio, open the desired script and click the ”Source” button.

If a custom species is desired, the species CSV must be downloaded from GBIF and stored under `/data/GBIF/[species name].csv`. Furthermore, the `species_list` parameter in `training-preprocessing.R` must be altered to reflect the new species.

Script Descriptions

See the above section for instructions on obtaining the appropriate inputs.

`training-preprocessing.R`

```
Input:
    Species datasets (.csv)
    Historical climate datasets (.nc)
Parameters:
    species_list: an array of strings describing the desired species
    e: extent variable describing longitude/latitude boundaries
Output:
    Presence/absence files (.geojson)
    Historical climate raster files (.tif)
```

Presence/absence files are created by using 1) occurrences from the species CSV file as presences and 2) an equal number of climate points from the climate

NetCDF file as pseudo-absences. Pseudo-absences are used due to lack of real absence data. GeoJSON files are stored under `/model/input/geojson`.

Climate raster files are converted from NetCDF to TIFF format for compatibility with python libraries. TIFF files are stored under `/model/input/historical`.

All outputs are cropped to the extent variable `e`. Default `e` is set to both Americas.

`prediction-preprocessing.R`

```
Input:
    Future climate datasets (.nc)
Parameters:
    e: extent variable that describes longitude/latitude boundaries
Output:
    Future climate raster files (.tif)
```

Climate raster files are converted from NetCDF to TIFF format for compatibility with python libraries. Unlike the climate rasters for model training (of which there are only two), there exists a pair of future rasters for each year for each SSP model, making a total of 680 raster files. TIFF files are stored under `/model/input/future`, organized by SSP model and year. Raster files are cropped to the extent variable `e`. Default `e` is set to both Americas.

1.3 Species Distribution Modeling in Python

The SDM was prototyped in Jupyter Notebook (see `birdmig_sdm.ipynb`) and implemented in python script (see `model_train.py` and `model_predict.py`). Python scripts were used over Jupyter Notebook for implementation due to ease of use with loops and terminal.

Requirements

Python scripts were developed using Python 3.10.12, which can be found at this link. While you may use newer versions of Python, they are not guaranteed to work. Required libraries include:

```
tqdm
pyimpute
rasterio
matplotlib
geopandas
scikit-learn
numpy
pandas
```

These libraries can be installed individually or by running the command `pip install -r model/requirements.txt`

Ensure that preprocessed data from R exists under `/model/input/`. There should be three folders: `geojson, historical,` and `future`, each containing the appropriate data. `geojson` and `historical` are required for model training, while `future` and `/data/models` is required for model predicting. Pickle files can be obtained by downloading the `/data/models` folder at the Box link or by running `model/model_train.py`.

1.3.1 SDM Training

The model is trained by associating presence/absence data with zonal averages for temperature and precipitation. In other words, the model learns what zonal environmental factors typically correlate to a presence or absence of a species. When given a file containing possible future climate readings, the model can then predict how the distribution of a species may react to that climate.

Training models for each bird species goes as follows:

1. Data is loaded into training vectors using the `load_data()` function
2. The random forest classifier is chosen for model training
3. Classifier's accuracy is measured by averaging K-fold evaluation (k=5)
4. Model is fit with training vectors and saved to a pickle file

Random forest was chosen for the model because it showed the highest and most consistent accuracy when compared to other classifiers (extra trees, SVM, xgboost, lightgbm) See `birdmig_sdm.ipynb`.

A description of the `load_data()` function is as follows:

```
load_data():
    Parameters:
        species: string representing species name
        n: optional int representing sample size (default=None)
    Return:
        train_xs: 2D array of floats where each inner array represents
                  zonal averages for temperature and precipitation
        train_y:  array whose values represent presence or absence &
                  are indexed to correlate to each array in train_xs

    load_data() loads appropriate files from /model/input/ into training
    vectors for SDM training. It takes an optional n parameter that
    randomly samples the presence/absence data in order to reduce the
    size of the training vector.

    load_data() primarily wraps the load_training_vectors() function
    from pyimpute, which is responsible for finding the zonal averages
```

```
to begin with. load_data() is also responsible for postprocessing
these vectors, removing NaN values to avoid training errors.
```

`model_train.py` is set up in such a way that it trains and saves models for each bird species in a loop, tracking the process using `tqdm`. If a custom species is desired, the `species_list` variable must be modified to reflect it (ensure that the CSV for the desired bird species has already undergone R preprocessing). These models are saved to `/data/models` via the `pickle` library.

In order to run the script, simply run the command `python3 model/model_train.py` in the root directory of this project.

1.3.2 SDM Predicting

SDM prediction takes place in `model_predict.py`. Its prototyping can again be found in `birdmig_sdm.ipynb`.

Species distribution prediction uses the following process:

1. Future climate rasters are loaded from `/model/input/future` and processed into an appropriate form factor
2. Pickle files for each species is loaded from `/model/output/models`
3. `impute()` from `pyimpute` is used to create distribution predictions in the form of TIFF images for each climate raster.
4. TIFF images are converted to PNGs for use in the web application and stored in `/public/png-images`

There exists a future climate raster for each year (2015-2100) for each SSP (ssp126, ssp245, ssp370, ssp585). These rasters are used to predict distribution for five species, making a total of 1,720 images.

`model_predict.py` is comprised of a triple-nested for loop (species `->` ssp `->` year) and two functions, `make_predictions()` and `create_pngs()`. The former function is responsible for steps 1-3 listed above, while the latter function is responsible for step 4. TIFF images are converted to PNG through the use of `matplotlib`, and are saved in high resolution.

In order to run the script, simply run the command `python3 model/model_predict.py` in the root directory of this project.

2 Web Application

2.1 APIs

2.1.1 React

React.js was used to created the front end, and the main component is found under `/src/App.js`. In App.js, there are calls to the back end to gather data for the various components located in the `/src/components` directory. These components include:

- `BirdInfo.js`: returns the summary of the desired bird
- `ClimateChart.js`: returns temperature and precipitation graphs
- `HeatMap.js`: returns heatmap graph (see Leaflet section)
- `PolylineMap.js`: return bird trajectory graph (see Leaflet section)
- `SDMchart.js`: returns png of SDM output
- `PredictionControl.js`: contains code to display the year slider and SSP buttons that change the Climate and SDM Charts

In addition to these components, the sidebar and header are created in `App.js` which allows users to view different birds. All the styling is written up in `/src/App.css`.

2.1.2 FastAPI

The middleware and back end of the website was developed using python and FastAPI, in the file `base.py` located under `/back end/app`. The FastAPI back end primarily serves the purpose of quickly retrieving data and sending it to the front end as requested, allowing the front end user-interface to remain lightweight and easy to load quickly. FastAPI was used to build the back end due to its ease in creating end points for a front end application to call, natively supporting features such as delayed requests and an intuitive way of passing arguments to the FastAPI back end.

The back end is defined predominantly by functions that handle various requests the front end makes in order to retrieve data to display. These functions and their functions are discussed in greater detail in the following section for RestFUL API.

2.1.3 RestFUL

RestFUL API was used to define how the end points were set up, partially due to built-in compatibility with FastAPI, but also due to its simple and intuitive interface. While RestFUL API primarily defines 4 primary methods for GET, PUT, POST, and DELETE, the final application was operational with a stateless back end, meaning only GET and PUT requests were used. In the file base.py, GET and PUT requests were used to define the following end points:

- `get_temperature_data()`: a GET endpoint that in turn performs a POST request to grid2.rcc-acis.org to obtain temperature data for the state of California throughout a year, which is specified per the user's request. Currently, this endpoint only utilizes the SSP8.5 scenario as other endpoints have not yet been developed by RCC-ACIS.
- `get_precipitation_data()`: a GET endpoint that also performs a POST request to grid2.rcc-acis.org to obtain precipitation data for the state of California throughout a year, specified per the user's request. Like the temperature endpoint, it also currently only utilizes the SSP8.5 scenario due to the limited development of other climate scenarios by RCC-ACIS.
- `get_predictions()`: a PUT end point that retrieves the pre-computed predictions the Species Distribution Model makes about a specified bird, for a specified year based on specified emission data. While a GET request would have sufficed for its current utility, a PUT request was preferred as this could be expanded to implement a machine learning model to generate predictions dynamically should such a feature be planned in the future.
- `get_trajectory_data()`: a GET end point that retrieves a csv file containing the migration trajectory of a specific bird, identified by its unique bird ID, that can be used to generate individual trajectory maps of an individual bird. The trajectory information is discussed later in the subsection for Leaflet.
- `get_bird_ids()`: a GET end point that returns all unique bird IDs per a given species. This is used to by the front end component PolylineMap.js to search for a specified bird ID.
- `get_heatmap_data()`: a GET end point that retrieves heatmap data of a bird species, to display the aggregated paths of all birds catalogued for that species on the front end with HeatMap.js, as a heat map. This is discussed in further detail in the Leaflet subsection.

2.2 Trajectory Map Preprocessing

To project trajectory maps on the front end, tracking data from online data repository, Movebank, was saved in CSV files. These files are preprocessed using `process_trajectory_data.py` located in the Box folder linked above. This script takes in an input CSV and the name of the output CSV to be created. The code processes the input CSV to only keep the columns utilized on the front end into the output CSV. The columns utilized on the front end are renamed to create standardized CSV files for the front end. The script performs these steps:

- Defines a dictionary new_columns for renaming columns.
- Defines a list keep_columns to specify which columns to retain.
- Reads the input CSV file into a DataFrame.
- Renames the columns using the new_columns dictionary.
- Filters the DataFrame to keep only the specified columns.
- Sorts the DataFrame by the 'TIMESTAMP' column.
- Drops rows with missing values in 'LATITUDE' and 'LONGITUDE' columns.
- Writes the processed DataFrame to the output CSV file.
- Prints the output file name, the processed DataFrame, and the range of 'TIMESTAMP' values.

To run the script, use the following command in the terminal:

```
python process_bird_data.py <input_file> <output_file>
```

Replace <input_file> with the path to your input CSV file and <output_file> with the path where you want to save the processed CSV file.

Example:

```
python process_bird_data.py bird_data.csv processed_bird_data.csv
```

2.2.1 Leaflet

Leaflet was utilized to delineate bird migration patterns in two distinct manners within the "Trajectory" component.

The initial map, "Individual Path," is constructed in /src/components/PolylineMap.js. The process involves fetching data from CSV files located in the data directory within /public/trajectory_data. Initially, all tagged Bird IDs are retrieved from the CSV for the specified bird. Subsequently, the trajectories for the first Bird ID are fetched from the back end and depicted on OpenStreetMap utilizing Leaflet. Additionally, arrows are computed between two points in the calculateBearings function and integrated onto the map.

The second map, "Aggregated Path," is generated in /src/components/HeatMap.js. This function retrieves all latitude and longitude coordinates from the CSV of the desired bird in the /public/trajectory_data directory. Employing Leaflet and the Leaflet plugin, L.heatLayer, a heat layer is generated to illustrate the combined trajectory path of all tagged birds atop the OpenStreetMap layer.

2.3 Usage

The website is designed to serve as an interactive tool for ornithologists, researchers, and bird watching enthusiasts. In order to start the application, the following actions must take place. **Ensure that your terminal is running commands in the root directory of this project.**

2.3.1 Requirements

Backend The backend was developed primarily with Python 3.12, but also functions on Python 3.11. It relies primarily on FastAPI for functionality, while unit testing is done using the `pytest` python module. Ensure that the correct Python version (Python 3.12 or Python 3.11) is installed, and ensure the following Python modules are installed:

```
fastapi
pydantic
requests
pandas
numpy
shapely
rasterio
pillow
pytest
```

These modules can be installed by running the command `pip install -r backend/requirements.txt`.

Frontend The front end of the application was implemented with React.js. For React.js to work, first Node.js must be installed. Information on how to do so can be found here. Once Node.js is installed, download React.js and Leaflet. Alternatively, running `npm install` in terminal will install the dependencies required to run the front end.

2.3.2 Process/Steps

1. Ensure that all the libraries required to run the application are installed. Refer back to the previous section to see the required libraries.

2. Prepare two command terminals: one dedicated to running and monitoring the back end, and one to running the front end. The environment could either be a Linux terminal or a Windows terminal.

3. Start the backend:

    (a) Run the command `fastapi dev backend/app/base.py`.

    (b) To access documentation of the back end, use a web browser and navigate to `localhost:8000/docs`

    (c) The back end should update automatically with saved modifications to the file. To terminate the back end, terminate the program with a SIGINT using `control + C`, if using either a Linux or Windows command terminal.

4. Once the back end is initialized, the front end of the application may be started.

   (a) Run the command `npm start`.

   (b) After a few minutes, the front end should open automatically in a new web browser window. If this window is either closed or does not open, use a web browser to navigate to `http://localhost:3000` to observe the front end User Interface.

5. Do not close either terminal unless their respective application is fully terminated. If a new terminal is started and another instance of either the front end or the back end is already running, a restart of the device may be required.

2.4 Design

The website is developed using a React front end and a FastAPI back end, leveraging modern web technologies and frameworks to ensure a responsive and intuitive user experience. Key design features include:

- **Modular Design:** The application is structured into several independent components like BirdInfo, SDMChart, and PolylineMap. Each component encapsulates a particular feature of the web app, facilitating seperation of concerns, readability, and scalability.

- **Data-Driven Interactions:** By integrating with external APIs and processing data through Python-based FastAPI, the application provides real-time data interactions and updates.

- **User-Centric Interfaces:** Emphasis on usability and accessibility, with interactive charts, maps, and dynamic controls that cater to both novice users and experienced researchers.

The application's architecture is designed to handle high volumes of data efficiently, utilizing caching and asynchronous data fetching to optimize performance and user experience.

2.5 FAQ

The necessary React.js components are installed, but the front end application is still causing an error.

- Ensure that all necessary React.js libraries have been installed in the project directory using the `npm install` command.

- If the front end still does not start, the React cache may need to be cleared. In a Windows or Linux terminal, navigate to the root directory of the project using the `cd` command, and run the command `npm cache clean --force` to clear the React cache, and then execute `npm install`.

The terminals that house the front end and/or back end have been closed without terminating the program, and now the corresponding end point of the application will not start.

- Restart the device, and ensure that the back end and front end applications have fully terminated before closing the terminals. This can be forced by forcing a SIGINT on the program with `control + C` on either the Windows or Linux terminal.

Running FastAPI on `base.py` fails due to missing packages.

- Ensure your terminal is in the root directory.
- Ensure that all required python packages are installed. This can be done by running `pip install -r backend/requirements.txt`