Automated iterative forecasting for the Portal Project

Introduction

Forecasting the future state of ecological systems is important for management, conservation, and evaluation of our fundamental understanding of ecology (Clark et al., 2001; Tallis & Kareiva, 2006; Díaz et al., 2015; Dietze, 2017). Since Clark et al. [-clark2001] called for a more central role of forecasting in ecology, an increasing number of ecological forecasts are being published. However, most of these forecasts are made once, published, and never assessed or updated. Without assessment, we have limited information on how much confidence to place in our predictions; without regular updates, forecasts lack the most up-to-date information as conditions change (Dietze et al., 2016). This lack of both regular assessment and active updating has limited the progress of ecological forecasting and hindered our ability to make useful and reliable predictions. For ecological forecasting to mature as a field, we need to change how we produce and interact with forecasts, creating a more dynamic interplay between model development, prediction generation, and incorporation of new data and information (Dietze et al., 2016).

With the goal of making ecological forecasting more dynamic and responsive, Dietze et al [-dietze2018] recently called for an increase in iterative near-term forecasting. Iterative near-term forecasting means making forecasts for the near future and making these forecasts repeatedly through a cycle of forecast evaluation, integration of updated data, and generation of new forecasts. This approach to forecasting has a number of advantages. Because forecasts are made 'near-term'—daily to annual forecasting instead of multi-decadal—predictions can be assessed more quickly and frequently, leading to more rapid model improvements (Dietze et al., 2016; Tredennick et al., 2016). Because the forecasts are made repeatedly through time, new data can be integrated with each new forecast cycle. This iterative approach to forecasting allows any changes in the state of the system that have occurred since the previous forecast to be incorporated and accounted for (Dietze et al., 2016). Iterative near-term forecasting has the potential to promote rapid improvement in the state of ecological forecasting by quickly identifying how models are failing, facilitating rapid testing of improved models, and incorporating updated data so models run with the most up-to-date information on the system available. While use of iterative near-term forecasting is often contextualized as a management tool, this approach to model testing can also be used to improve our basic understanding of ecological systems. For example, alternative mechanistic models can be competed to see ~~which~~ model provides the best forecasts for near-term dynamics, thus providing insights into the relative importance of different processes driving dynamics of ecological systems (Dietze et al., 2016). Whether deployed for basic or applied uses, iterative near-term forecasting incorporates a more dynamic interplay between models, predictions, and data that is clearly needed to improve ecological forecasting and our understanding of ecological systems more broadly.

Because iterative near-term forecasting requires a dynamic interplay of models, predictions, and data, Dietze et al [-dietze2018] highlight approaches to data management, model construction and evaluation, and cyberinfrastructure that are necessary to effectively implement this type of forecasting (Box 1). Data to be used for iterative near-term forecasting needs to be widely accessible, which requires data to be released quickly under open licenses (Dietze et al., 2016; Vargas et al., 2017) and structured so that it can be used easily by a variety of researchers and in multiple modeling approaches (Borer, Seabloom, Jones, & Schildhauer, 2009; Strasser, Cook, Michener, Budden, & Koskela, 2011). Models need to be able to deal with uncertainty,

~~in both the predictors and the predictions~~, to properly convey uncertainty in the resulting forecasts. Multiple models should be ~~compared~~ to assess which models are performing best and to ~~allow for combining~~ models to form ensemble predictions. Ensuring that data and models are regularly updated and new forecasts are made requires cyberinfrastructure to automate data processing, model fitting, prediction, model evaluation, forecast visualization, and archiving. In combination, these approaches should allow forecasts to be easily rerun and evaluated as new data bec~~omes~~ available (Box 1; Dietze et al., 2016).

While iterative near-term forecasting is an important next step in the evolution of ecological forecasting, the requirements outlined by Dietze et al (Box 1) are not trivial to implement; few of their recommendations are in widespread use in ecology today. We examined what it would entail to operationalize Dietze et al's recommendations by constructing our own iterative near-term forecasting pipeline for an on-going long-term (~40 year) ecological study that collects high-frequency data on desert rodent abundances. We constructed our forecasting pipeline with the goal of being able to forecast rodent abundances and evaluate our predictions on a monthly basis. In this paper, we ~~discuss~~ our approach for creating this iterative near-term forecasting pipeline, the challenges we encountered, the tools we used, and the lessons we learned that ~~may help~~ others ~~to~~ create their own iterative forecasting systems.

System Background

Iterative forecasting requires data that ~~is~~ collected repeatedly, and it benefits most from data that ~~is~~ collected frequently, as this provides more opportunities for updating model results and assessing (and potentially improving) model performance (Box 1; Dietze et al., 2016). The Portal Project is a long-term ecological study situated in the Chihuahuan Desert (2 km north and 6.5 km east of Portal, Arizona, US). Researchers have been continuously collecting data at the site since 1977, including data on the abundance of rodent and plant species (monthly and twice yearly, respectively) and climactic factors such as air temperature and precipitation (daily). The site consists of 24 50m x 50m experimental plots. Each plot contains 49 permanently marked trapping stations laid out in a 7 x 7 grid, and all plots are trapped with Sherman live traps for one night each month. For all rodents caught during a trapping session, information on species identity, size, and reproductive condition is collected, and new individuals are given identification tags. We use the data from the control plots at ~~this site~~, where rodent populations are not experimentally manipulated. Th~~is~~ data on rodent populations ~~is~~ high-frequency, ~~uses~~ consistent trapping methodology, and ~~has~~ an extended time-series (469 monthly samples and counting), making this study an ideal case for near-term iterative forecasting.

Implementing an automated iterative forecasting system

Implementation of iterative forecasting requires the regular rebuilding of models with new raw data as ~~it~~ becomes available and the presentation of those forecasts in usable forms; in our case, this occurs monthly. ~~Doing this~~ in an efficient and maintainable way relies on developing an automated pipeline to handle the six stages of converting raw data into new forecasts: data collection, data sharing, data manipulation, modeling and forecasting, archiving, and ~~presentation of~~ the forecasts (Figure 1). To implement the pipeline outlined in Figure 1, we used a "continuous analysis" framework (*sensu* Beaulieu-Jones & Greene, 2017) that automatically processes the most up-to-date data, refits the models, makes new forecasts, archives the forecasts, and updates a website with analysis of current and previous forecasts. In this section we describe our approach to streamlining and automating the multiple components of the forecasting pipeline and the tools and infrastructure we employed to execute each stage of the pipeline.

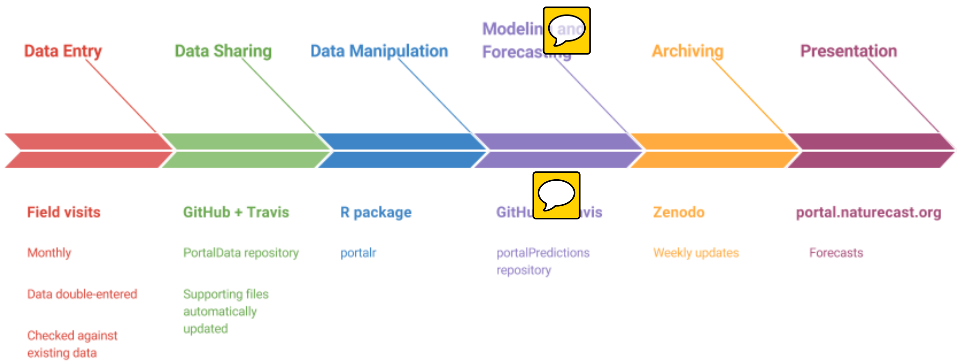

Figure 1: Figure 1. Stages of the forecasting pipeline. To go from raw data to forecast presentation involves a number of stages, each of which required unique tasks, tools and infrastructure. While each stage has unique tasks associated with it, the stages of our pipeline are also interdependent. Our pipeline is primarily a linear structure where outputs from one stage form the inputs for the subsequent stage

Continuous Analysis Framework

A core component of iterative near-term forecasting is the regular rerunning of the forecasting pipeline. This can be conducted manually—with a human making sure all code is run and all tables and files are updated—or automatically by having the computer conduct those tasks. We chose to have the computer run our pipeline and employed "continuous analysis" (*sensu* Beaulieu-Jones & Greene, 2017) to drive the automation of both the full pipeline and a number of its individual components. Continuous analysis uses a set of tools originally designed for software development called "continuous integration" (CI). CI combines computing environments for running code with monitoring systems to identify changes in data or code. Essentially, CI is a computer helper whose job is to watch the pipeline and, when it sees a change in the code or data, ~~it runs~~ all the tasks needed to ensure that the forecasting pipeline runs from beginning to end. This is useful for iterative near-term forecasting because it does not rely on humans to remember to create forecasts when new models or data are added. These tools are common in the area of software development where they are used to automate software testing and integrate work by multiple software developers working on the same software. However, these tools can be used for any computational task that needs to be regularly repeated or run after changes to code or data (Beaulieu-Jones & Greene, 2017). Because of the widespread use of CI in software development, several CI services already exist. Our forecasting pipeline currently runs on a publicly available continuous integration service (Travis CI; https://travis-ci.org/) that is free for open source projects (up to a limited amount of computing time); alternative services that can be used to run code on local or cloud-based computational infrastructure are available, such as Drone (http://try.drone.io/) (Beaulieu-Jones & Greene, 2017). As detailed below, we use CI to quality check data, test code using "unit tests" (Wilson et al., 2014), build models, make forecasts, and publicly present and archive the results (Figure 2).

In addition to automatically running software pipelines, the other key component of "continuous analysis" is making sure that the pipelines will continue to run even as software dependencies change (Beaulieu-Jones & Greene, 2017). Many of us have experienced the frustrations that can occur when software updates (e.g., changes in R package versions) create errors in previously functional code. We experienced this issue when a package one of our models relies on, `tscount` [liboschik2015], was temporarily removed from CRAN (the R package repository) and, therefore, could not be installed in the usual way. This broke our forecasting pipeline because we could no longer run models that used that package. To minimize issues with changes in software dependencies, we follow Beaulieu and Green's (2017) recommendation to use software containers. Software containers are standalone packages that contain copies of everything you need to run some piece of software. Once created, a software container is basically a time capsule, it contains all the software dependencies in the exact state used to develop and run the software. If those dependencies change (or disappear) in the

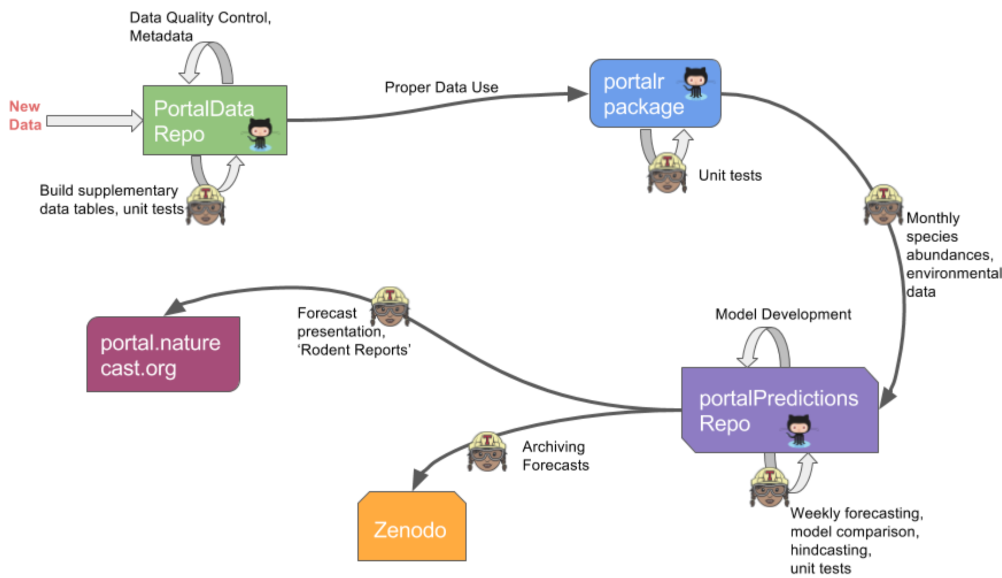

Figure 2: Figure 2. Continuous integration system. Each box denotes the core infrastructure used for each stage of the forecasting pipeline denoted in Figure 1. Continuous integration, denoted here with the Travis icon (a girl wearing safety glasses and hardhat), plugs into our pipeline at every stage. Travis triggers the code involved in major events that involve integration across stages of the pipeline, such as taking the output from the forecasting stage (purple box) to create an updated presentation (rose box). Travis also runs tasks within a stage, such as conducting tests to make sure code changes have not introduced errors (icons on arrows originating and ending on the same box)

wider world, they still exist, unchanged, in your container. We use an existing platform, Docker, to store an exact image of the complete software environment for running the forecasts. Docker also allows a specified set of packages to be used consistently across different computer and server environments. Using containers allows us to update to new package versions after testing and making any necessary changes to the data processing and analysis code. We use a container created by the Rocker project which is a Docker image with many important R packages (i.e. tidyverse) pre-installed (Boettiger & Eddelbuettel, 2017). We ~~use~~ add our code and dependencies to this existing Rocker image to create a software container for our forecasting pipeline. In combination, the automated running of the pipeline (continuous integration) and the guarantee it will not stop working unexpectedly due to software dependencies (via a software container) allow~~s~~ continuous analysis to serve as the glue that connects all stages of the forecasting pipeline.

Data Collection, Entry, and Processing

Iterative forecasting benefits from frequently updated ~~data on the state~~ of the system so that state changes can be quickly incorporated into new forecasts (Dietze et al., 2016). Frequent data collection and rapid ~~entry and~~ processing ~~of data~~ are both important for providing ~~the most up-to-date data for forecasting~~. Since our data ~~is~~ collected monthly, ensuring that the models have access to the newest data requires a data latency period of less than 1 month from collection to availability for modeling. To accomplish this, we automated components of the data processing and quality assurance/quality control (QA/QC) process to reduce the time needed to add new data to the database (Figures 1 and 2).

New data i~~s~~ double-entered into Excel using the “data validation” feature. The two versions are then compared in an R script to control for errors in data entry. Quality control (QC) checks written in R using the `testthat` R package (Wickham, 2011) are run on the data to test for validity and consistency both within the new data (e.g. sexual characteristics of an animal match M/F designation) and between the new and archived data (e.g. species and sex are consistent for recaptures of the same animal based on tag number). The local use of the QC scripts to flag problematic data greatly reduces the time spent error-checking and ensures that the quality of data is consistent. The data ~~is~~ then submitted to a GitHub-based data repository. GitHub is a software development tool for managing computer code development, but we have also found it useful for data management. Changes to data can be tracked through version control, and additions and changes to the data can be monitored through pull requests (notifications that someone has a modified version of the data to contribute). QA/QC checks are automatically rerun on the submitted data using continuous integration to ensure that these checks have been run and that no avoidable errors reach the official version of the dataset.

We also automated the updating of supplementary data tables, including information on weather and trapping history, that were previously updated manually. As soon as new field data ~~is~~ merged into the repository, continuous integration updates all supplementary files. Weather data ~~is~~ automatically fetched from our cellular-connected weather station, cleaned, and appended to the weather data table. Supplementary data tables related to trapping history are updated based on the data added to the main data tables. Using CI for this ensures that all supplementary data tables are always up-to-date with the core data.

Data Sharing

The Portal Project has a long history of making its data publicly available, which means that anyone can use it for forecasting or other projects. Historically the publication of the data was conducted through data papers, the most common approach in ecology; however, this approach caused years of data latency. Recently, the project has switched to posting data directly to a public GitHub repository with a CC0 license (Figure 1).

This immediate posting reduces that data latency to less than one month and, therefore, makes meaningful iterative near-term forecasting possible for not only our group but other interested parties as well.

Data Manipulation

Once data ~~is~~ available, ~~it~~ need~~s~~ to be processed into a form appropriate for modeling (Figure 1). In many ecological ~~datasets~~, this requires not only simple data manipulation but also a good understanding of the ~~dataset~~ to ~~allow data to be aggregated appropriately. These d~~ata manipulation steps are often conducted using custom one-off code to convert the data into the desired form (Morris & White, 2013), ~~but~~ this approach has several limitations. First, each researcher must develop their own data manipulation code, which is inefficient and can result in ~~different decisions by~~ different researchers ~~about the details of data cleaning and aggregation~~. Subtle differences in data processing decisions have le~~a~~d to confusion when reproducing results ~~for~~ the Portal data in the past. Second, this kind of code is rarely robust to changes in data structure and location. Based on our experience developing and maintaining the Data Retriever (Morris & White, 2013; Senyondo et al., 2017), these kinds of changes are common. Finally, this kind of code is generally poorly tested, which can lead to errors based on mistakes in data manipulation. To avoid these issues for the Portal Project data, the Portal team has been developing an R package (portalR; http://github.com/weecology/portalr) for acquiring the data and handling common data cleaning and aggregation tasks. As a result, our modeling and forecasting code only needs to install this package and run the data manipulation and summary functions to get the appropriate data (Figure 2). The package undergoes thorough automated unit tests to ensure that data manipulations are achieving the desired results. Having data manipulation code maintained in a separate package that focuses on consistently providing properly summarized forms of the most recent data has made maintaining the forecasting code itself much more straightforward.

Modeling and Forecasting

Ideally, iterative near-term forecasting involves regularly refitting a variety of different models (Figure 1). New models should be easy to incorporate to allow for iterative improvements to the general modeling structure and approach. We use CI to refit the models and make new forecasts each time the modeling code changes and when new data becomes available (Figure 2). We use a plugin infrastructure to allow new models to be easily added to the system. Each model is a script that takes in standardized inputs and returns standardized outputs (Figure 3). All models are run by the main forecasting code at each update, and the standardized outputs are combined to store the results of the different models' forecasts. A weighted ensemble model is then added with weights based on how well individual models fit the training data. This plugin infrastructure allows flexibility in all aspects of the modeling process, making it easier to explore new modeling approaches, and allows new models that fit the data well to immediately improve the ensemble forecast.

To allow flexibility in what the models predict, we designed a forecast output format that allows us to store relatively generic forecasts. Each forecast output file contains the date being forecast, the collection date of the data used for fitting the models, the date the forecast was made, the state variable being forecast (e.g., rodent biomass, the abundance of a species), and the forecast value and associated uncertainty of that forecast (Figure 3). This allows a variety of different forecasts to be stored in a common format and may serve as a useful starting point for developing a standard for storing ecological forecasts more generally.

Forecasts are currently evaluated using root mean square error (RMSE) to evaluate point forecasts and coverage to evaluate uncertainty. We plan to add additional metrics in the future. In addition to evaluating the actual forecasts, we also use hindcasting (forecasting on already collected data) to gain additional insight into the methods that work best for forecasting this system. For example a model is fit using rodent

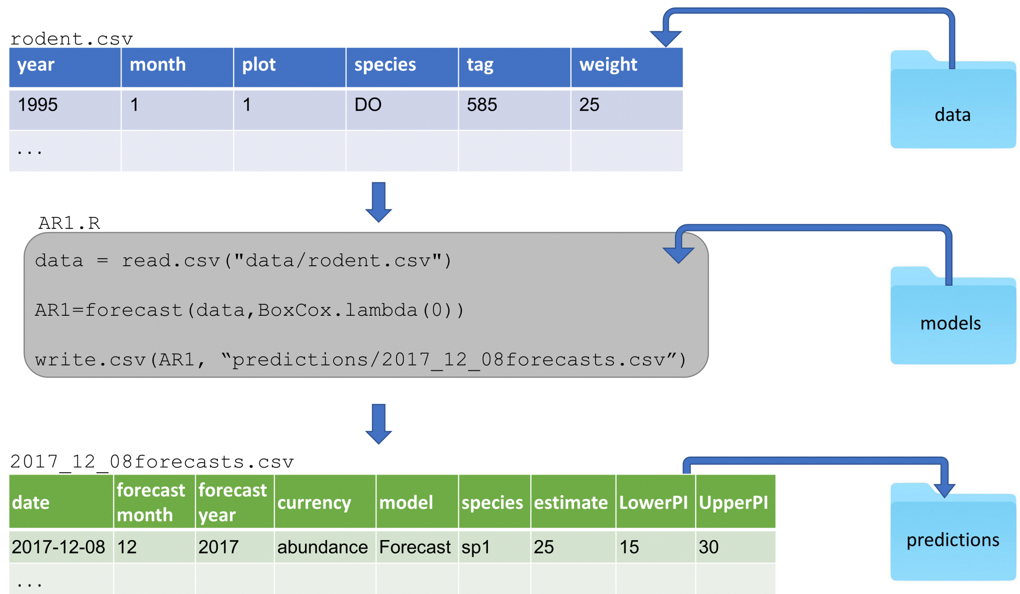

Figure 3: Figure 3. Demonstration of plugin infrastructure where each model script (represented here as AR1.R) uses data provided by the core forecasting code (represented here as rodent.csv) and returns its forecasts in a predefined structure that is consistent across models (represented here as 2017_12_08forecasts.csv) to allow cross-model evaluations.

observations up to June 2005, then used to make a forecast 12 months out to May 2006. The observations of that 12 month period can immediately be used to evaluate the model. Hindcasting can be conducted using all months from the beginning of the study, thus allowing model comparison of large numbers of hindcasts and giving insight into which models make the best forecasts. It can also be used to quickly evaluate new models instead of waiting for an adequate amount of data to accumulate.

Archiving

Publicly archiving forecasts before new data is collected allows the field to assess, compare, and build on forecasts made by different groups (McGill, 2012; Tredennick et al., 2016; Dietze, 2017; J. Harris David, Taylor, & White, 2018)(Figure 1). This archiving serves as a form of pre-registration for model predictions, helping facilitate unbiased interpretation of model performance. We explored three major repositories for archiving our forecasts: FigShare, Zenodo, and Open Science Framework. While all three repositories allowed for easy manual submissions (i.e., a human uploading files after each forecast), automating this process was substantially more difficult. Various combinations of repositories, APIs and associated R packages had issues with: 1) integrating authorization with continuous integration; 2) automatically making archived files public; 3) adding new files to an existing location; or 4) automatically permanently archiving the files. Our eventual solution was to leverage the GitHub-Zenodo integration (https://guides.github.com/activities/citable-code/) and automatically push forecasts to the GitHub repository from the CI server. The GitHub-Zenodo integration is designed to automatically create versioned archives of GitHub repositories. There is an existing one-time process for linking our forecasting repository on GitHub with Zenodo. Once this link is created, each time a new forecast is created, our pipeline adds the new forecasts to the GitHub repository and uses the GitHub API to create a new "release" of our repository. This triggers the Zenodo-GitHub integration, which automatically archives the resulting forecasts and the code that generated them under a top-level DOI that refers to all archived forecasts (https://doi.org/10.5281/zenodo.833438). Through this process, we automatically archive every forecast made with a documented history of the archive. While this approach is functional because everything in the repository is archived when a new forecast is made, these archives are complicated, making it more complicated than necessary to find and access the forecasting results.

Presentation

In addition to archiving the results of each forecast, we present them on a website that displays monthly rodent forecasts, model evaluation metrics, monthly reports, and information about the study site (Figure 4; http://portal.naturecast.org). The website includes a graphical presentation of the most recent month's forecasts (including uncertainty) and compares the latest data to the previous forecasts. Information on the

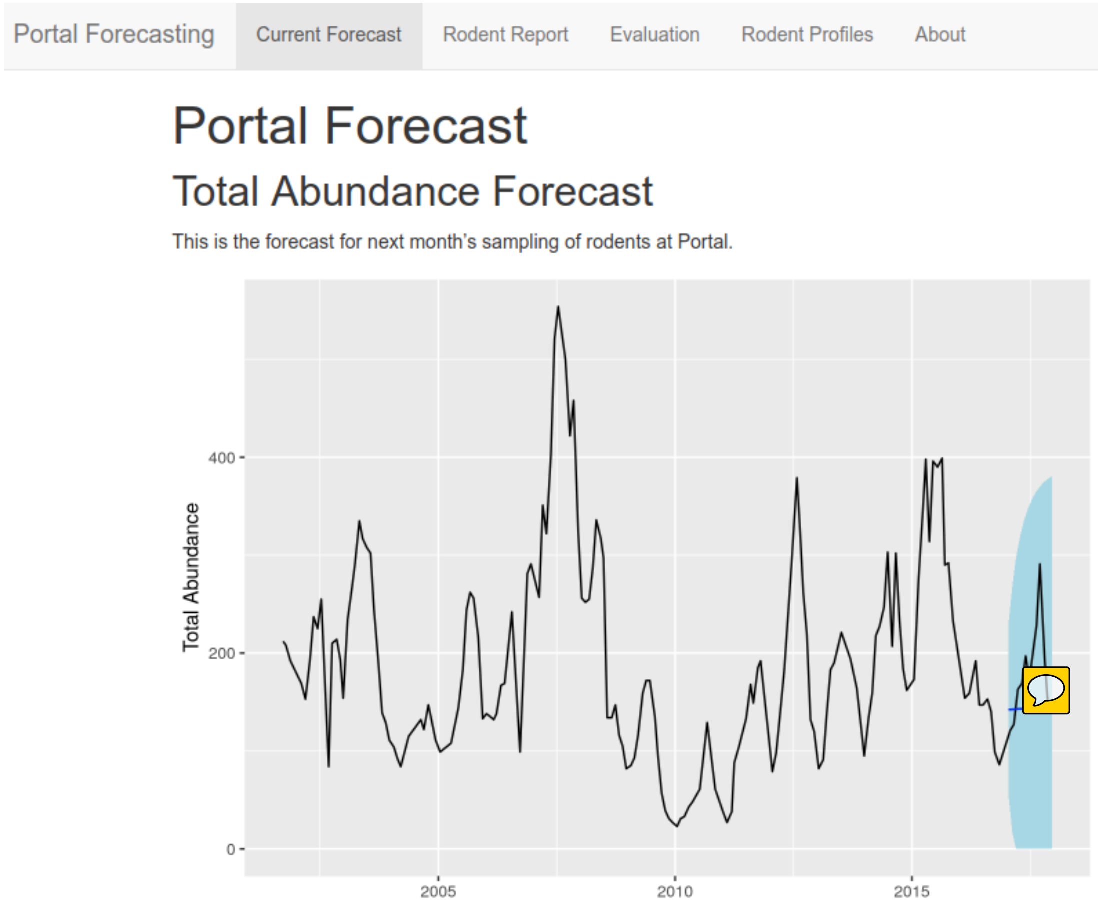

Figure 4: Figure 4. Screen capture of the homepage of the Portal Forecasting website; http://portal.naturecast.org. This site contains information on the most current forecasts, evaluation of forecast performance, and general information about the species being forecast.

the species and the field site targeted to a general audience are also included. The site is built using Rmarkdown (Allaire et al., 2017), which naturally integrates into the pipeline, and is automatically updated after each forecast. The `knitr` R package (Xie, 2015) compiles the code into HTML, which is then published using Github Pages (https://pages.github.com/). The files for the website are stored in a subdirectory of the forecasting repository. As a result, the website is also archived automatically as part of the forecast archiving.

Discussion

Following the recommendations of Dietze et al [-dietze2018], we developed an automated iterative forecasting system (Figures 1 and 2) to support repeated forecasting of an ecological system. Our forecasting system automatically acquires and processes the newest data, refits the models, makes new forecasts, publicly archives those forecasts, and presents both the current forecast and information on how previous forecasts performed. Every week our forecasting system generates a new set of forecasts with no human intervention, except for the entry of new field data. This ensures that forecasts based on the most recent data are always available and allows us to rapidly assess the performance of multiple forecasting models for a number of different states of the system, including the abundances of individual species and community-level variables such as total abundance. To create this iterative near-term forecasting system, we used R to process data and conduct analyses~~,~~ and we leveraged already existing services (i.e. GitHub, Travis, Docker) for more complicated cyberinfrastructure tasks. Thus, our approach to developing iterative near-term forecasting infrastructure provides an example for how short-term ecological forecasting systems can be initially developed.

We designed this forecasting system with the goal of making it relatively easy to build, maintain, and extend. We used existing technology for both running the pipeline and building individual components, which allowed the system ~~to be built~~ relatively cheaply in terms of both time and money. This included the use of tools like Docker for reproducibility, the Travis CI continuous integration system for automatically running the pipeline, Rmarkdown and `knitr` for generating the website, and the already existing integration between Github and Zenodo to archive the forecasts. By using this "continuous analysis" approach (Beaulieu-Jones & Greene, 2017), where analyses are automatically rerun when changes are made to data, models, or associated code, we have reduced the time required by scientists to run and maintain the forecasting pipeline.

To make the system extensible so that new models could be easily incorporated, we use a plugin-based infrastructure so that adding a new model to the system ~~is~~ as easy as adding a single file to the 'models' folder in our repository (Figure 3). This should substantially lower the barriers to other scientists contributing models to this forecasting effort. We also automatically archive the resulting forecasts publicly so that~~, as new data is collected,~~ the performance of these forecasts can be assessed by both us and other researchers. This serves as a form of "pre-registration" by providing a quantitative record of the forecast before the data being predicted were collected.

While building this system was facilitated by the use of existing technological solutions, there were still a number of challenges in making existing tools work for automated iterative forecasting. Continuous integration is designed primarily for running automated tests on software, not for running a coordinated forecasting pipeline. As a result, extra effort was sometimes necessary to figure out how to get these systems to work properly in non-standard situations, like running code that was not part of a software package. In addition, hosted continuous integration solutions, like Travis, provide only limited computational resources. As the number and complexity of the models we fit has grown, we have had to continually invest effort in reducing our total compute time so we can stay within these limits. Finally, we found no satisfactory existing solution for archiving our results. All approaches we tried had limitations when it came to automatically generating publicly versioned archives of forecasts on a repeated basis. Overall, we found existing technology to be sufficient to the task, but it required greater expertise and a greater investment of time than is ideal. Tool development to reduce the ~~effort~~ required for scientists to set up their own short-term forecasting systems would clearly be useful, but our efforts show that it is ~~still currently~~ possible for scientists ~~using existing tools~~ to develop initial iterative systems as a method for both advancing scientific understanding and developing proof of concept forecasting systems. By developing these systems so that they ~~already~~ produce automated iterative forecasts, it will be easier to convert the~~se systems~~ into fully operationalized forecasting systems that are relied on for decision making (Dietze et al., 2016, other paper by operationalization co-authors).

Because of the breadth of expertise needed to set up our forecasting pipeline, our effort required a team with diverse skills and perspectives, ranging from software development to field site expertise. It is rare to find such breadth within a single research group, and our system was developed as a collaboration between the lab collecting the data and a computational ecology lab. When teams have a breadth of expertise, communication can be challenging. We found a shared base of knowledge related to both the field research and fundamental computational skills was important for the success of the group. Everyone on the team ~~had~~ received training in fundamental data management and computing skills through a combination of university courses, Software and Data Carpentry workshops (Teal et al., 2015), and informal lab training efforts. In addition, everyone was broadly familiar with the study site and methods of data collection, and most team members had participated in field work at the site on multiple occasions. This provided a shared set of knowledge and vocabulary that actively facilitated interdisciplinary interactions. Given the current state of ~~existing~~ tools for forecasting, forecasting teams will need people with significant experience in working with continuous integration and APIs, which means interdisciplinary teams will generally be required for creating these pipelines until tool development improves.

We developed this infrastructure for automatically making iterative forecasts with the goals of making accurate forecasts for this well-studied system, learning what methods work well for ecological forecasting more generally, and improving our understanding of the processes driving ecological dynamics. The most obvious application of automated iterative ecological forecasting is for speeding up development of forecasting models by providing the most recent data available to models and by quickly iterating to improve the models used for forecasting. By learning what works best for forecasting in this and other ecological systems, we will better understand what the best approaches are for ecological forecasting more generally. By designing the pipeline so that it can forecast many different aspects of the ecological community, we also hope to learn about what aspects of ecology are more forecastable. Finally, automated forecasting infrastructures like thi~~s~~ one also provide a core foundation for faster scientific inquiry more broadly because new models can quickly be applied to data and compared to existing models. The forecasting infrastructure does the time-consuming work of data processing, data integration, and model assessment, allowing new research to focus on the models being developed and the inferences about the system that can be drawn from them (Dietze et al., 2016). We plan to use this pipeline to drive future research into understanding the processes that govern the dynamics of individual populations and the community as a whole. By regularly running different models for population and community dynamics, a near-term iterative pipeline such as ours should also make it possible to rapidly detect changes in how ~~the~~ system is operating, which should allow the rapid identification of ecological transitions or even possibly allow them to be prevented **(???** example). By building an automated iterative near-term forecasting infrastructure we can improve our ability to forecast natural systems, our understanding of the biology driving ecological dynamics, and detect or even predict changes in system state that are important for conservation and management.

Acknowledgements

This research was supported by the National Science Foundation through grant 1622425 to S.K.M. Ernest and by the Gordon and Betty Moore Foundation's Data-Driven Discovery Initiative through grant GBMF4563 to E.P. White. We thank all of the graduate students, postdocs, and volunteers who have collected the Portal Project over the last 40 years and the developers of all of the software and tools that made this

project possible.

Box 1. Key practices for automated iterative near-term ecological forecasting

A list of some of the key practices developed by Dietze et al [-dietze2018] for facilitating iterative near-term ecological forecasting and discussion of why these practices are important.

Data

1. Frequent data collection

Frequent data collection allows models to be regularly updated and forecasts to be frequently evaluated (Dietze et al., 2016). Depending on the system being studied, this frequency could range from sub-daily to annual, but typically the more frequently the data ~~is~~ collected the better.

2. Rapid data release under open licenses

Data should be released as quickly as possible (low latency) under open licenses so that forecasts can be made frequently and data can be accessed by a community of forecasters (Dietze et al., 2016; Vargas et al., 2017).

3. Best practices in data structure

To reduce the time and effort needed to incorporate thi~~s~~ data into models, best practices in data structure need to be employed for managing and storing collected data to ensure it is easy to integrate into other systems (interoperability) (Borer et al., 2009; Strasser et al., 2011; White et al., 2013).

Models

4. Focus on uncertainty

Understanding the uncertainty of forecasts is crucial to interpreting the forecasts and understanding their utility. Models used for forecasting should be probabilistic to properly quantify uncertainty and to convey how this uncertainty increases through time. Evaluation of forecast models should include assessment of how accurately they quantify uncertainty as well as point estimates. This can be done using “proper and local” scores (Hooten & Hobbs, 2015).

5. Compare forecasts to simple baselines

Understanding how much information is present in a forecast requires comparing its accuracy to simple baseline~~s~~ to see if the model~~s~~ yield improvement~~s~~ over the naive expectation that the system is static (J. Harris David et al., 2018).

6. Compare and combine multiple modeling approaches

To quickly learn about the best approaches to forecasting different aspects of ecology, multiple modeling approaches should be compared for forecasting tasks (J. Harris David et al., 2018). Different modeling approaches should also be combined into ensemble models, which are known to outperform single models for many forecasting and prediction tasks (Weigel, Liniger, & Appenzeller, 2008).

Cyberinfrastructure

In addition to improvements in data and models, iterative near-term forecasting requires improved infrastructure and approaches to support continuous model development and iterative forecasting (Dietze et al., 2016).

7. Best practices in software development

Best practices should be followed in the development of scientific software and modeling to make it easier to maintain, integrate into pipelines, and build on by other researchers. Key best practices include using open licenses, good documentation, version control, and cross-platform support (Wilson et al., 2014; Hampton et al., 2015).

8. Support easy inclusion of new models

To facilitate the comparison and ensembling of different modeling approaches, code for fitting models and making forecasts should be easily extensible, allowing models developed by different groups to be easily integrated into a single framework (Dietze et al., 2016).

9. Automated end-to-end reproducibility

Iteratively making forecasts requires that acquiring the newest data, refitting the models, and making new forecasts is simple. Ideally, this should be done automatically without requiring human intervention. Therefore, the process of making forecasts should emphasize end-to-end reproducibility, including data, models, and evaluation (Stodden & Miguez, 2014), to allow the forecasts to be easily rerun as new data becomes available (Dietze et al., 2016). Ideally, the entire forecasting pipeline will be rerun automatically as new data becomes available.

10. Publicly archive forecasts

Forecasts should be openly archived to demonstrate that the forecasts were made without knowledge of the outcomes (i.e., as a form of pre-registration *sensu*) and to allow the community to assess and compare the performance of different forecasting approaches both now and in the future (McGill, 2012; Dietze et al., 2016; Tredennick et al., 2016; J. Harris David et al., 2018). Ideally, the forecasts and evaluation of their performance should be automatically posted publicly in a manner that is understandable by both interested scientists and other stakeholders.

References

Allaire, J., Cheng, J., Xie, Y., McPherson, J., Chang, W., Allen, J., … Arslan, R. (2017). *Rmarkdown: Dynamic documents for r*. Retrieved from https://CRAN.R-project.org/package=rmarkdown

Beaulieu-Jones, B. K., & Greene, C. S. (2017). Reproducibility of computational workflows is automated using continuous analysis. *Nature Biotechnology*, *35*(4), 342–346.

Boettiger, C., & Eddelbuettel, D. (2017). An introduction to rocker: Docker containers for r. *arXiv Preprint arXiv:1710.03675*.

Borer, E. T., Seabloom, E. W., Jones, M. B., & Schildhauer, M. (2009). Some simple guidelines for effective data management. *The Bulletin of the Ecological Society of America*, *90*(2), 205–214.

Clark, J. S., Carpenter, S. R., Barber, M., Collins, S., Dobson, A., Foley, J. A., … others. (2001). Ecological forecasts: An emerging imperative. *Science*, *293*(5530), 657–660.

Dietze, M. C. (2017). *Ecological forecasting*. Princeton University Press.

Dietze, M. C., Fox, A., Betancourt, J. L., Hooten, M. B., Jarnevich, C. S., Keitt, T. H., … others. (2016). Iterative ecological forecasting: Needs, opportunities, and challenges. *Proceedings of the National Academy of Sciences*.

Díaz, S., Demissew, S., Carabias, J., Joly, C., Lonsdale, M., Ash, N., … others. (2015). The ipbes conceptual framework—connecting nature and people. *Current Opinion in Environmental Sustainability*, *14*, 1–16.

Hampton, S. E., Anderson, S. S., Bagby, S. C., Gries, C., Han, X., Hart, E. M., … others. (2015). The tao of open science for ecology. *Ecosphere*, *6*(7), 1–13.

Harris, J., David, Taylor, S. D., & White, E. P. (2018). Forecasting biodiversity in breeding birds using best practices. *PeerJ*.

Hooten, M. B., & Hobbs, N. (2015). A guide to bayesian model selection for ecologists. *Ecological Monographs*, *85*(1), 3–28.

McGill, B. J. (2012). Ecologists need to do a better job of prediction – part ii – partly cloudy and a 20% chance of extinction (or the 6 p's of good prediction). Retrieved from https://dynamicecology.wordpress.com/2013/01/09/ecologists-need-to-do-a-better-job-of-prediction-part-ii-mechanism-vs-pattern/

Morris, B. D., & White, E. P. (2013). The ecodata retriever: Improving access to existing ecological data. *PloS One*, *8*(6), e65848.

Senyondo, H., Morris, B. D., Goel, A., Zhang, A., Narasimha, A., Negi, S., … White, E. P. (2017). Retriever: Data retrieval tool. *The Journal of Open Source Software*, *2*(19), 451. doi:10.21105/joss.00451

Stodden, V., & Miguez, S. (2014). Best practices for computational science: Software infrastructure and environments for reproducible and extensible research. *Journal of Open Research Software*, *2*(1).

Strasser, C., Cook, R., Michener, W., Budden, A., & Koskela, R. (2011). Promoting data stewardship through best practices. In *Proceedings of the environmental information management conference 2011 (eim 2011)*. Oak Ridge National Laboratory (ORNL).

Tallis, H. M., & Kareiva, P. (2006). Shaping global environmental decisions using socio-ecological models. *Trends in Ecology & Evolution*, *21*(10), 562–568.

Teal, T. K., Cranston, K. A., Lapp, H., White, E., Wilson, G., Ram, K., & Pawlik, A. (2015). Data carpentry: Workshops to increase data literacy for researchers. *International Journal of Digital Curation*, *10*(1), 135–143.

Tredennick, A. T., Hooten, M. B., Aldridge, C. L., Homer, C. G., Kleinhesselink, A. R., & Adler, P. B. (2016). Forecasting climate change impacts on plant populations over large spatial extents. *Ecosphere*, *7*(10).

Vargas, R., Alcaraz-Segura, D., Birdsey, R., Brunsell, N. A., Cruz-Gaistardo, C. O., Jong, B. de, … others. (2017). Enhancing interoperability to facilitate implementation of redd+: Case study of mexico. *Carbon Management*, *8*(1), 57–65.

Weigel, A. P., Liniger, M., & Appenzeller, C. (2008). Can multi-model combination really enhance the prediction skill of probabilistic ensemble forecasts? *Quarterly Journal of the Royal Meteorological Society*, *134*(630), 241–260.

White, E. P., Baldridge, E., Brym, Z. T., Locey, K. J., McGlinn, D. J., & Supp, S. R. (2013). Nine simple ways to make it easier to (re) use your data. *Ideas in Ecology and Evolution*, *6*(2).

Wickham, H. (2011). Testthat: Get started with testing. *The R Journal*, *3*, 5–10. Retrieved from http://journal.r-project.org/archive/2011-1/RJournal_2011-1_Wickham.pdf

Wilson, G., Aruliah, D. A., Brown, C. T., Hong, N. P. C., Davis, M., Guy, R. T., … others. (2014). Best practices for scientific computing. *PLoS Biology*, *12*(1), e1001745.

Xie, Y. (2015). *Dynamic documents with r and knitr* (Vol. 29). CRC Press.