Meyers Lab Report #2

Advay Vyas

September 11, 2025

Contents

1 Flu-Metrocast

1.1 Dashboard

The dashboard has a scroller for the forecasts and the ability to select multiple forecasts. This forecast evaluation slider is quite neat, with the horizon at the end and how long the interval is. The evaluation tab also has a space to look at the various values and metrics for each forecast (as well as a heatmap).

2 Gradient boosting model

2.1 Data preprocessing methods

Removed COVID data to offset influenza and COVID mix-ups likely, added forecast targets. Used bags to "bootstrap" the data along with the gradient boosting. Used feature importance, how is that calculated? How is "horizon" the most important feature, isn't that we are predicting? Kept important features, scaled back to original, renamed, set up quantile. Made predictions using model.

3 Alternative methods for prediction

I just searched up some alternative methods and found the following:

- Hierarchical forecast reconciliation
- Spatio-temporal GAMs (mixed-effects)
- Bayesian hierarchical spatio-temporal models
- Gaussian process regression
- Deep learning

4 R package creation

I'm taking notes on "R Packages (2e)" by Hadley Wickham and Jennifer Bryan.

4.1 The Whole Game

Devtools package is useful for package development. Devtools automatically creates a `.gitignore` and likely has an implementation for Git usage outside of beginning in a Git repo (actually recommended to not do that). The command to use Git (in any R project) is `use_git()`. We save function definitions in the `function\_name.R` file in the folder. The `check()` function is the "gold standard" of checking R packages completely work.

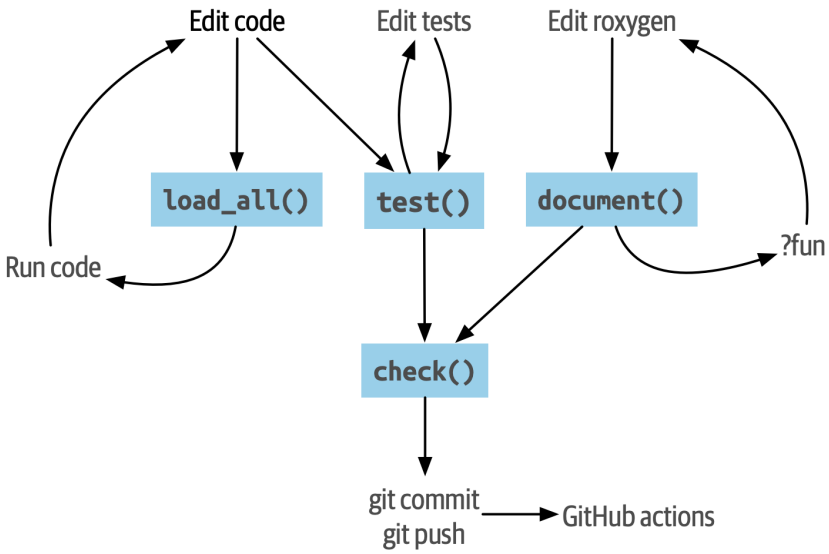

Figure 1: A picture of the package workflow from the book

Editing the DESCRIPTION file is important, we can do stuff like declaring an author, titling the package, version number, etc. Also important is declaring a license (MIT license will probably do just fine). Additionally, we need to add documentation to our functions and code so that users know how to use them in the future. We use *Code* → *Insert roxygen skeleton* to create our documentation boilerplate and then fill it out from there, then using `document()` to update the documentation. Help file can now be accessed like any other native function in R, with ?function_name. We can now `install()` the package into our library.

To write unit tests for our code, we use the `use_testthat()` funciton that starts the unit testing creation process. You can open a test file with `use_test()`. To use another package in your code (even functions that are often already in the default like `stats::median()` or `utils::head()`), you must use the `use_package()` command.

To use GitHub for R package development, it's recommended to start early with `use_github()`. Also, `use_readme_rmd()` initializes the file for easy editing and `build_readme()` renders the file. Lastly, the book recommends us to commit often, `check()` often, and `install()` often.

A list of useful functions:

- `create_package()`
- `use_git()`
- `use_mit_license()`
- `use_testthat()`
- `use_github()`
- `use_readme_rmd()`
- `use_r()`
- `use_test()`
- `use_package()`
- `load_all()`
- `document()`
- `test()`
- `check()`