BIO322 - Coding tips for miniproject

November 2022

Pluto notebooks are great to visualize data and show results but sometimes other setups are preferable. If you want you can code with julia scripts (`.jl`), which can be easier to handle between two people on git and less heavy to load than Pluto notebooks. Feel free to choose the setup you prefer the most. Here are some tips:

- To activate the MLCourse environment in a script:
  `Pkg.activate(joinpath(Pkg.devdir(), "MLCourse"))`;
- To import your own functions: `include("./myfunctions.jl")`;
- Suggested Julia programming workflow from the developers here;
- VisualStudio Code is a great editor for any programming language, you can work on your Julia scripts with VSCode. Tutorial to configure it with your local julia installation here.

1 Other useful Julia links

- Core Julia cheatsheet here;
- `MLJ` cheatsheet here;
- Differences between Julia and other languages here;
- Analogies and differences between `DataFrame.jl` and corresponding packages in other languages (`Pandas` [Python] , `data.table` [R]) here;

2 Installing new packages

If for your project you need to install packages that are not present in the MLCourse environment (you should not need them, but nobody stops you from exploring more) you need to create a new environment, as modifying the current MLCourse is going to create conflicts with future git updates. Here's one way to do it:

1. Copy the `Manifest.toml` file of the MLCourse environment in your project folder (in order to not reinstall all the MLCourse packages). You can find the manifest in
   `"computer_username/.julia/dev/MLCourse/Manifest.toml"`;
2. Activate your new environment: go in the project directory, open Julia and run `using Pkg; Pkg.activate(".")`. Your environment will have the same name as the folder it is in;
3. Compile the `Manifest.toml` packages with `Pkg.instantiate()`;
4. Install any package that you need with
   `Pkg.add("newpackage"); Pkg.instantiate()`;

5. For more info on how to deal with packages, watch 'Further Tips and Tricks for the Project' on the course Moodle page.

Remember to activate your environment before running your project scripts, be sure you are in your project folder, open a REPL and just do `using Pkg; Pkg.activate(".")`, or add these lines at the beginning of your script or notebook. If you use a different environment than MLCourse, be sure to include the `Manifest.toml` and `Project.toml` files in the git repo, otherwise we will have reproducibility issues and you will lose points.

3 Reproducibility

Reproducibility of simulation results is very important, as highlighted by a recent example where software engineers called for the retraction of a scientific paper that influenced politics[1]. In our project, reproducibility can be achieved by

1. communicating the versions of all dependencies that are used as specified in the Manifest.toml file of a project. If you use the MLCourse environment we will assume that your results are based on the current Manifest.toml of MLCourse. Please make sure that your local copy in folder .julia/dev/MLCourse is identical to the Manifest.toml on github. Otherwise you may include your own Manifest.toml in your git repository (as specified in the previous section).

2. setting the seeds of random number generators. If all code and dependencies are fully written in julia it is sufficient to call 'Random.seed!(SOME_NUMBER)' just before any command that could depend on some random number generator, like cross-validation dependent model tuning or initialization dependent neural networks. In Pluto notebooks, random seeds are not maintained across different cells; 'Random.seed!(SOME_NUMBER)' has to be called within each cell that relies on some random number. Alternatively, one can specify the random number generator, e.g. 'rng = Xoshiro(123)' and pass it to 'rand' etc. like 'rand(rng, 2)'. If some dependencies are written in another language there is usually a 'seed' argument that can be passed to functions that depend on random number generators. Among the packages used in MLCourse only 'MLJXGBoostInterface', 'MLJLIBSVMInterface' and 'GLMNet' depend on code written in other languages. In particular for machines defined by 'MLJXGBoostInterface' the seed matters.

[1] In this example the engineers were also not convinced that the code implements correctly the equations that were used in the publication. We take for granted here that the MLJ ecosystem does proper testing to assure that code and equations are consistent