Lab Handbook

Machine and Human Intelligence group, University of Helsinki

2024-11-12

Table of contents

Introduction

Lab handbook of the Machine and Human Intelligence group at the University of Helsinki.

Part I

Python tools

1 How to publish Python packages

We decsribe here the steps and tips for publishing Python packages on PyPI and `conda-forge`.

1.1 Publishing on PyPI

Publishing a package on PyPI is generally simpler than publish on `conda-forge`, and it's easiest to use the PyPI package as the source for a `conda-forge` recipe anyway, so it's best to start here. More details can be found here.

1. Make sure your package dependencies are all correctly specified, e.g. by creating a new Conda environment and running `pip install -e .` from an up-to-date source and ensuring that all tests pass. Dependencies should be specified with a lower bound, rather than pinned to a specific version, for maximum compatibility.
    - There are a few different ways to specify project dependencies, but the most modern is with a `pyproject.toml` file in the root of the project directory. This is what we have used for PyVBMC and GPyReg, this guide may need to be adapted if you are using a different method. From what I understand, it is also good practice to leave a "stub" in setup.py for backwards compatibility, as in here.
2. Make sure `pyproject.toml` contains the appropriate lines. These are taken from PyVBMC, so they may not all be required for your project, but this is a starting point. You may also already have a `pyproject.toml` with other information in it, such as configurations for the Black formatter. That's fine, those sections can remain.

```
# pyproject.toml
#
[project]
name = "PyVBMC"  # Naming a project is required
dynamic = ["version"]  # use git tags for version, via setuptools_scm
# If you don't want to use setuptools_scm, you can specify the version
# manually:
# version = "v1.0.0"
description = "Variational Bayesian Monte Carlo in Python."
readme = "README.md"
```

```
license = { file = "LICENSE" }
dependencies = [
    "cma >= 3.1.0",
    "corner >= 2.2.1",
    # ...
]
requires-python = ">=3.9"
#
[tool.setuptools]
include-package-data = true
# We want to include some files which are not in the source directory,
# such as the Jupyter Notebooks in /examples.
# See https://setuptools.pypa.io/en/latest/userguide/datafiles.html
# for more info.
packages = ["pyvbmc", "pyvbmc.examples"]
package-dir = {"pyvbmc.examples" = "examples"}
#
[tool.setuptools.package-data]
# Specify the extensions to include as data:
"pyvbmc.examples" = ["*.ipynb"]
#  Including files which *are* in the source directory but are not
# .py files can also be accomplished with a MANIFEST.in file. See
# e.g. https://github.com/acerbilab/pyvbmc/blob/main/MANIFEST.in
#
[project.optional-dependencies]
dev = [
    "myst_nb >= 0.13.1",
    "numpydoc >= 1.2.1",
    # ...
]  # These dependencies are for developers. They will only be installed
# with `pip install pyvbmc[dev]` (or `pip install .[dev]`, locally).
# You can specify other sets of optional dependencies similarly.
#
[build-system]
# Required for using setuptools.scm, which automatically assigns the
# version number based on Git tags.
requires = [
    "setuptools >= 45",
    "setuptools_scm[toml] >= 6.2",
]
build-backend = "setuptools.build_meta"
```

- `setuptools_scm` is a tool which extracts versioning info from Git tags, so that you don't have to manually specify package versions.

3. Tag the current commit as a release version with

```
git tag vX.Y.Z
```

See here for some details about semantic versioning.
4. Install Python build tools. It's probably a good idea to create a new environment for the whole build and packaging process.

```
conda create -n build-env  # optional, but recommended
conda activate build-env # if you created build-env, otherwise make sure you have activated y
pip install setuptools-scm
pip install build
pip install twine
```

5. Build your package:

```
python -m build
```

(from the project directory)
6. This should create a directory `dist/` with `.whl` and `.tar.gz` files matching your package name and tagged version. Inspect these files and make sure that they contain the contents you expect. By default this should be the source directory corresponding to your project's name, e.g. `/pyvbmc`. If you are missing files that should be there, then see the link above (https://setuptools.pypa.io/en/latest/userguide/datafiles.html) regarding `package-data` and `MANIFEST.in`.
7. You can also run `twine check dist/*` to ensure that the package name and description will render properly on PyPI.
8. If everything looks correct, test out the build by creating a new environment and running `pip install dist/*.whl` to install the packaged version and test it out. For example, you could run the tests with `pytest --pyargs your-package` (do this from somewhere *outside* the project directory, to ensure that `pytest` is finding the version you just installed, and not the local tests). You could also open a Python REPL and just ensure that your package imports. You may need to open a new shell before the installation can be found on your path.

```
conda create -n test-package-release
conda activate test-package-release
pip install dist/*.whl
cd ~ # Test the build package outside the working directory.
```

```
pytest --pyargs your-package
# Ensure later to come back to the project working directory
```

9. If everything checks out and you are ready to upload, first head to https://test.pypi.org/ and register an account if you don't already have one. Then you can test uploading your package by running `twine upload --repository testpypi dist/*` from the project directory. Your package should then be visible under your account on the test repository, and you can make sure that the description and other info are correct. You can also attempt to install it with

```
python3 -m pip install --index-url https://test.pypi.org/simple/ --extra-index-url https://py
```

The `--extra-index-url`tells `pip` to also look at the regular PyPI repository, which is important if your package has dependencies which are not on the test repo (this is likely the case).
10. If everything looks good on the test repository, you can run `twine upload dist/*` to upload to the official repository (you will need an account there as well, separate from your test account). Be aware that while a version of a package can be deleted from PyPI (so that it is no longer available), that same version number can never be re-uploaded. So it pays to double-check.
11. Once uploaded, you should be able to run `pip install your-package`!

Other notes

1. You can add another user as an owner or maintainer of a project through your account at https://pypi.org/
2. Optionally, you may also want to list the package version on GitHub. You can do this by pushing the version tag(s) to the origin with `git push --tags`, then selecting the tag from https://github.com/account-name/repo-name/tags and clicking "Create release". You can describe the release, and optionally upload the built `.whl` and `.tar.gz` files.

1.2 Uploading to `conda-forge`

Uploading to `conda-forge` is slightly more involved. Detailed instructions can be found here, but here is a summary of steps:

1. Fork the `conda-forge` staged-recipes repository from GitHub, and clone the fork locally. Checkout a new branch, e.g. `pyvbmc-recipe`, then `cd` into the `staged-recipes/recipes/` directory.

2. Install `grayskull` with `pip install grayskull` (or `conda install -c conda-forge grayskull`). Grayskull is a utility which will help generate the appropriate metadata for your package.
3. If your package is already on PyPI, you can run `grayskull pypi --strict-conda-forge your-package` to generate a `conda-forge` recipe for your package. It should create the file `staged-recipes/recipes/your-package/meta.yaml`.
4. Everything should be filled in automatically (though it's good to double-check), with the exception of

```
|about:
|  home: https://acerbilab.github.io/pyvbmc/
|

|extra:
|  recipe-maintainers:
|    - AddYourGitHubIdHere
```

(ignore the vertical bars, Colab won't let me include leading spaces without them). You can also add other people as recipe maintainers, with their permission. It just means they will receive automated PRs and other updates regarding the project from `conda-forge`, and possibly answer any questions that pop up. 5. Optionally, you can add documentation and development URLs to the `about:` section, e.g.

```
|about:
|  home: https://acerbilab.github.io/pyvbmc/
|  description: |
|    PyVBMC is a Python implementation of the Variational Bayesian Monte Carlo (VBMC) algori
|  doc_url: https://acerbilab.github.io/pyvbmc/
|  dev_url: https://github.com/acerbilab/pyvbmc/
|  ...
```

6. Optionally, you can include commands in the `test` section of the `meta.yaml` recipe, which will be automatically run when `conda-forge` updates the package. The `pyvbmc` test suite is quite expensive to run, so I elected to just include the default basic tests here, which just ensure that the package can be imported.
7. Once you think everything is correct (see a checklist here), you can commit the changes to your new branch, push them to GitHub, and then open a PR to merge your fork to the original `staged-recipes` repo. Fill out the template checklist which appears when you draft the PR.
8. Once you've opened the PR, the `conda-forge` automation will check your recipe and attempt to install the package. Correct any errors that occur, and comment on GitHub with `@conda-forge-admin, please restart ci` to re-run the automated checks.

9. Once the automated checks all pass, you can ping a member of the `conda-forge` team to review and approve the PR with `@conda-forge-admin, please ping conda-forge/help-python`.
10. After the PR is merged, it will take a few hours (and possibly up to 24) for the package to become available on the Conda servers. After that `conda install --channel=conda-forge your-package` should work!
11. A repo will be created at `conda-forge/your-package-feedstock`, and you and any other recipe maintainers will be added to it. This is where automated PRs regarding your package will be issued.

1.3 Updating the package version

Fortunately, this part is simple.

- **Updating the version on PyPI:**

  Re-run steps 3-5 from the section Publishing on PyPI above, and upload the new `.whl` and `.tar.gz` files to PyPI with `twine`.

- **Updating the version on `conda-forge`:**

  After the package becomes available on PyPI, `conda-forge` should automatically find the new version (this will take a few hours). After that, the `conda-forge` process will automatically issue a PR to the feedstock repo, which a recipe maintainer can approve in order to update the `conda-forge` version. After approval, the binary on the `conda-forge` cloud/repo will be automatically updated within 24 hours.

Part II

HPC resources

2 Introduction

Whenever doing research, High Performance Computing (HPC) access and usage is fundamental, you will rarely be running things only on a local machine. The main HPC resources we can use are CSC and UH's Turso.

2.1 CSC

The CSC is in charge of Finland's national computational resources. Many storage and computing programmes exist here, but the main ones for running ML experiments are Lumi, Mahti, and Puhti.

- Mahti and Puhti are considered the same project resource. When you get approved for a project, you can use either to run. The Mahti partition contains 24 GPU nodes, each equipped with four NVIDIA A100 GPUs. The Puhti partition includes 80 GPU nodes, each featuring four NVIDIA Volta V100 GPUs.
- Lumi is separate from Mahti and Puhti and has a separate project approval process. It contains 2,978 nodes, each equipped with four AMD Instinct MI250X GPUs, totaling 11,912 GPUs, considered one of Europe's largest supercomputers

2.2 HPC tutorials

This is a tutorial on HPC usage held in June 2021 by Aalto University, very useful. The videos are here.

HPC user guide for UH. They have support sessions organized every day if you have any questions or need any help with HPC usage.

Slack channel: Join and check out the **#hpc** channel on the lab Slack.

3 Useful commands

3.1 Squeue

Check the status of submitted slurm jobs.

```
squeue -u $USER --format="%.18i %.9P %.8j %.8u %.8T %.10M %.20S %.9l %.6D %R %.10m"
```

For convenience, you could set the default by writing the following lines in your `.bashrc` or `.bash_profile` and then simply call `squeue`.

```
alias squeue='squeue -u $USER'
export SQUEUE_FORMAT="%.18i %.9P %.8j %.8u %.8T %.10M %.20S %.9l %.6D %R %.10m"
```

See the squeue manual for more options.

Part III

Research topics

4 Resources for Gaussian processes and Bayesian optimization

Gaussian Processes

- **Gortler et al. (2019)**. A Visual Exploration of Gaussian Processes. Distill
- **Videos and tutorials** from the Gaussian Process Summer School: GPSS 2021 Program
  - Start with the first video, "Intro to GPs", and the sessions on "Kernel Design" and "Representation Learning with GPs" (other videos can be explored later as time permits).
- **Tutorials/workshops (Jupyter notebooks)**: GPSS 2021 Labs
- **The GP Bible** by Rasmussen and Williams (2006): Gaussian Processes for Machine Learning

Bayesian Optimization

- **Exploring Bayesian Optimization** (Agnihotri and Batra 2020). Distill
- **Frazier (2018)**. A Tutorial on Bayesian Optimization. arXiv
- **Video**: "Introduction to Bayesian Optimization" from the Gaussian Process Summer School, Day 3: GPSS 2021 Day 3
- **The Bayesian Optimization Book** by Garnett (2023): BayesOptBook

Slack Channel

- Join and check out the **#gaussian-processes** channel on the lab Slack.

5 Seminars

5.1 Ongoing

- Machine Learning Coffee seminars
- Aalto Seminar on Advances in Probabilistic Machine Learning [APML]
- Seminar on Gaussian Processes, Spatiotemporal Modeling, and Decision-making Systems

5.2 Interesting Channels

- Secondmind Labs Research seminars

Part IV

Other info

6 Contributing Guide

Here are the detailed steps to ensure a smooth workflow. **It's simply editting the `.qmd` file and making pull requests on GitHub**. So don't be scared .

6.1 Steps to Contribute

1. **Fork the Repository**
   Visit the handbook repository and click on "Fork" to create a copy of the repository under your GitHub account.

2. **Clone the Repository**
   Clone your fork to your local machine:

   ```
   git clone https://github.com/YOUR_USERNAME/handbook.git
   cd handbook
   ```

3. **Create a New Branch**
   Before making changes, create a new branch to keep your work organized:

   ```
   git checkout -b your-branch-name
   ```

4. **Add or Modify Content**

   - Write or edit a `(q)markdown` file as needed.
   - If you're adding a new chapter, create a new `.qmd` file, then include it under the `book: chapters:` field in _quarto.yml. For example:

   ```
   book:
     chapters:
       - index.qmd
       - part: Python tools
         - your-new-chapter.qmd
   ```

5. **Preview Locally**

   To see your changes before pushing, build and preview the book locally. You could use for example VSCode as the editor and Quarto has a VSCode plugin to help preview the book website locally, see instructions here.

Or make sure you have Quarto installed, then run in bash terminal:

```
quarto preview
```

This will open a live preview in your browser where you can verify your changes.

6. **Commit and Push Changes**
   After making sure your changes look good, commit and push them to your fork:

```
git add .
git commit -m "Brief description of your changes"
git push origin your-branch-name
```

7. **Make a Pull Request**
   Go to the original handbook repository and open a Pull Request from your branch. Provide a brief summary of your changes and mention any specific details reviewers should be aware of.

6.2 Additional Resources

For more advanced features and customization options, please refer to the Quarto documentation.

Bibliography

Agnihotri, Apoorv, and Nipun Batra. 2020. "Exploring Bayesian Optimization." *Distill*. https://doi.org/10.23915/distill.00026.

Garnett, Roman. 2023. *Bayesian Optimization*. Cambridge University Press.

Rasmussen, Carl Edward, and Christopher KI Williams. 2006. *Gaussian Processes for Machine Learning*. Vol. 2. 3. MIT press Cambridge, MA.