Lamalab Tool and Paper Notes

LamaLab

4/11/24

Table of contents

1 Tool and paper minutes

In our group seminars, we have a tradition of dedicating a few minutes to showcase tools/software/tricks/methods that we find useful. This repository is a collection of these tool minutes.

2 References

Part I

Tools

3 Hydra

3.1 Getting started

Hydra is an open-source Python framework that simplifies the development of research and other complex applications. The key feature is the ability to dynamically create a hierarchical configuration by composition and override it through config files and the command line. The name Hydra comes from its ability to run multiple similar jobs - much like a Hydra with multiple heads.

3.1.1 Key features:

- Hierarchical configuration composable from multiple sources
- Configuration can be specified or overridden from the command line
- Dynamic command line tab completion
- Run your application locally or launch it to run remotely
- Run multiple jobs with different arguments with a single command

3.1.2 Installation

```
pip install hydra-core --upgrade
```

3.1.3 Basic example

Config, e.g., in `conf/config.yaml`:

```
db:
driver: mysql
user: omry
pass: secret
```

4 IP Rotator

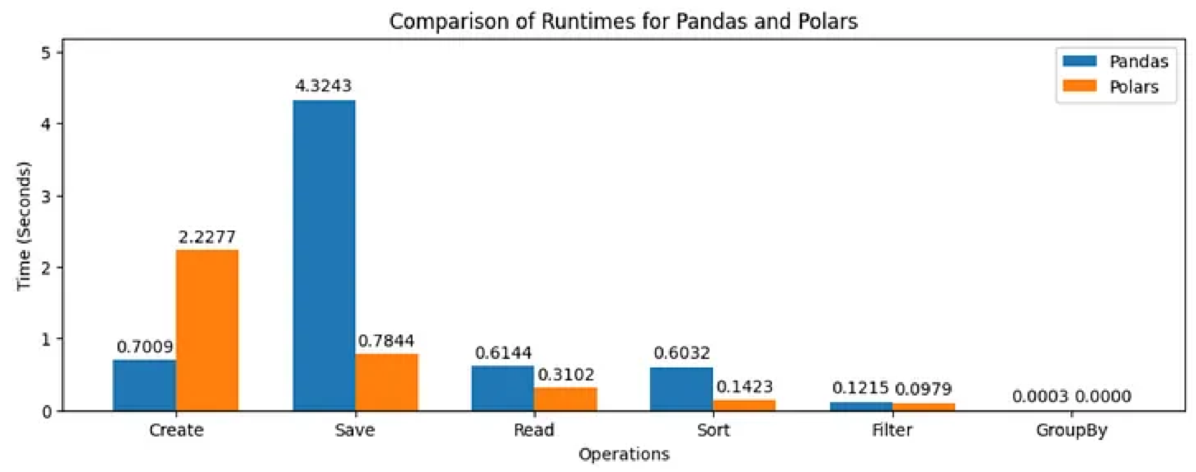
4.1 GitHub repository

iq-requests-rotator

4.1.1 Example usage:

```
import requests
from requests_ip_rotator import ApiGatewaywith ApiGateway("https://site.com") as g:
    session = requests.Session()
    session.mount("https://site.com", g)    response = session.get("https://site.com/index
    print(response.status_code)
```

5 Polars



5.1 An alternative to pandas

The advantages of polars can be directly seen in the image above. It is clear from the graph that Polars perform faster than Pandas for most operations. This is particularly true for the GroupBy operation, where Polars is nearly 20 times faster than Pandas. The Filter operation is also significantly faster in Polars, while Create operations are somewhat faster in Pandas. Overall, Polars seems to be a more performant library for data manipulation, particularly for large datasets.

5.2 Syntax example

```python
import polars as pl

q = (
    pl.scan_csv("docs/data/iris.csv")
    .filter(pl.col("sepal_length") > 5)
    .group_by("species")
    .agg(pl.all().sum())
)

df = q.collect()
```

6 Thunder Client

Thunder Client is a lightweight alternative to Postman that can be used directly from VS-Code.

You can use it to test your API endpoints.

For an example, see this video.

6.1 Installation

Install the Thunder client extension from the marketplace.

7 tmux

`tmux` is a terminal multiplexer. It lets you switch easily between several programs in one terminal, detach them (they keep running in the background) and reattach them to a different terminal. And do a lot more.

7.1 Installation

```
sudo apt install tmux
```

or on Mac

```
brew install tmux
```

7.2 Usage

Let's assume you are via ssh on a remote server and you want to run a long running process. You can use `tmux` to run the process in a session and then detach from it. You can then log out and log back in later to check on the process. Your process will still be running, even if your ssh session is closed.

7.2.1 On the remote server

```
tmux new -s myprocess
```

Then run your process. When you are done, detach from the session by pressing `Ctrl+b` and then `d`.

7.2.2 On the remote server later

```
tmux ls
```

This will list all the sessions. You can then reattach to the session you want by typing:

```
tmux attach -t myprocess
```

7.2.3 Panes

You can split your terminal into panes. This is useful if you want to run multiple processes in the same terminal. You can split the terminal vertically by pressing `Ctrl+b` and then `"` or horizontally by pressing `Ctrl+b` and then `%`.

To move panes around, you can use `Ctrl+b` and then `o` to cycle through the panes.

8 Robust statistics and Trimean

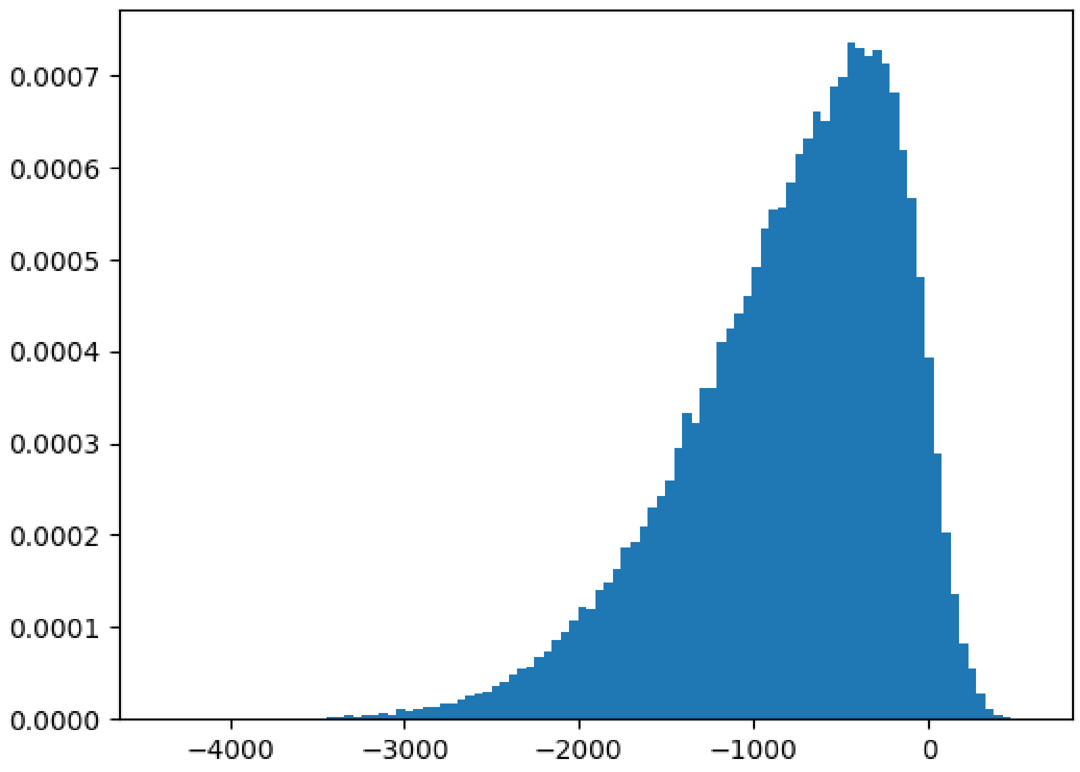
```
from scipy.stats import skewnorm
import numpy as np
import matplotlib.pyplot as plt
```

Let's generate some data that might be something we find in the real world.

```
skew_magnitude = -6
arr = skewnorm.rvs(skew_magnitude, loc=0, scale=1000, size=100000)
```

(The skew is a third-order moment.)

```
plt.hist(arr, bins=100, density=True)
plt.show()
```



Let’s get a very common measure of central tendency:

```
np.mean(arr)
```

```
-789.5809069979605
```

The mean overstates the central tendency because of the skew.

The mean is defined as

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$$

and treats all numbers equally. No matter how big or small.

One can “fix” this by looking at “robust” statistics that are often rank based. Rank based means that we sort the data and then base our statistics on the rank of the data. In this way, they are no longer sensitive to outliers.

```python
def interquartile_range(arr):
    q1 = np.percentile(arr, 25)
    q3 = np.percentile(arr, 75)
    return q3 - q1

print("Median", np.percentile(arr, 50))
print("Interquartile Range", interquartile_range(arr))
print("Mean", arr.mean())
print("Standard Deviation", arr.std())
```

```
Median -679.7024551978025
Interquartile Range 834.2816858677052
Mean -789.5809069979605
Standard Deviation 614.9363837309692
```

A very nice measure of centrality is the so-called trimean.

> "An advantage of the trimean as a measure of the center (of a distribution) is that it combines the median's emphasis on center values with the midhinge's attention to the extremes."
>
> — Herbert F. Weisberg, Central Tendency and Variability

It is defined as

$$\text{trimean} = \frac{Q_1 + 2Q_2 + Q_3}{4}$$

where Q_1 is the first quartile, Q_2 is the median, and Q_3 is the third quartile.

```python
def trimean(arr):
    q1 = np.percentile(arr, 25)
    q3 = np.percentile(arr, 75)
    median = np.percentile(arr, 50)
    return (q1 + 2*median + q3)/4

print("Trimean", trimean(arr))
```

```
Trimean -708.4430042323374
```

9 Easy fast `.apply` for pandas

`apply` in `pandas` is slow. This is the case because it does not take advantage of vectorization. That means, in general, if you have something for which there is a built-in `pandas` (or `numpy`) function, you should use that instead of `apply`, because those functions will be optimized and typically vectorized.

The `pandarallel` package allows you to parallelize `apply` on a `pandas` `DataFrame` or `Series` object. It does this by using `multiprocessing`. However, since it uses multiple processes, it will use more memory than a simple `apply`.

If your data just barley fits in memory, you should not use `pandarallel`. However, if it does fit in memory, and you have a lot of cores, then `pandarallel` can speed up your code significantly with just changing one line of code.

```python
from pandarallel import pandarallel

pandarallel.initialize(progress_bar=True)

# df.apply(func)
df.parallel_apply(func)
```

10 Easy fast `.apply` for pandas

`apply` in `pandas` is slow. This is the case because it does not take advantage of vectorization. That means, in general, if you have something for which there is a built-in `pandas` (or `numpy`) function, you should use that instead of `apply`, because those functions will be optimized and typically vectorized.

The `pandarallel` package allows you to parallelize `apply` on a `pandas` `DataFrame` or `Series` object. It does this by using `multiprocessing`. However, since it uses multiple processes, it will use more memory than a simple `apply`.

If your data just barley fits in memory, you should not use `pandarallel`. However, if it does fit in memory, and you have a lot of cores, then `pandarallel` can speed up your code significantly with just changing one line of code.

```
from pandarallel import pandarallel

pandarallel.initialize(progress_bar=True)

# df.apply(func)
df.parallel_apply(func)
```

11 BFG Repo-Cleaner

If you did not take with your `.gitignore` or just used `git add .` you might have by accident committed large files. This might lead to an error like

```
remote: error: See https://gh.io/lfs for more information.
remote: error: File reports/gemini-pro/.langchain.db is 123.01 MB; this exceeds GitHub's file
remote: error: GH001: Large files detected. You may want to try Git Large File Storage - http
To github.com:lamalab-org/chem-bench.git
 ! [remote rejected]     kjappelbaum/issue258 -> kjappelbaum/issue258 (pre-receive hook decl
error: failed to push some refs to 'github.com:lamalab-org/chem-bench.git'
```

To fix this, you need to remove the large files. A convenient tool for doing this is BFG.

Once you download the file you can run it using something like

```
java -jar ~/Downloads/bfg-1.14.0.jar --strip-blobs-bigger-than 100M --no-blob-protection
```

to remove large files.

Note that this here uses `--no-blob-protection` as BFG defaults to not touching the last commit.

After the BFG run, it will prompt you to run something like

```
git reflog expire --expire=now --all && git gc --prune=now --aggressive
```

12 showyourwork

showyourwork : https://github.com/showyourwork is a framework for building reproducible papers. The package works on a combination of Tex and Python code, where you can on the fly modify your plots.

The pre-requisites are: 1. define a conda environment with the packages are that necessary for plotting 2. use the `\script{}`, `\variable{}` and other commands to link your figures/tables to a Python script. 3. compile the paper