A Reproducible Research Pipeline

Using Git and Data Version Control (dvc)

Lukas Erhard

2025-02-20

University of Stuttgart, CSS Lab

The Big Problem

How can we work on the same research project collaboratively?

Working on research together has major advantages

- The research is faster
- The code has less bugs
- It keeps the research reproducible

What do we need to work together?

1. Same code
2. Same data
3. Same environment
4. Ensured order of execution

What do we want for compute-intensive projects?

1. Being able to run code on clusters
2. Not re-run all code every time

Problem 1

Having the same code

This is what git is for, so we can skip this problem...

Do we all know what a `.gitignore` file is?

Problem 2

Having the same data

aka “oh no, my data is too big for git, let’s use `data_final_corrected_final2_superfinal.csv`”

What is dvc?

- Tool (written in Python) that augments the functionality of git
- DVC stands for: **Data Version Control**
- provides the commmand: **dvc**

command: dvc add [path/to/file]

- adds any file or folder to a `.gitignore` file
- creates a `.dvc` file instead which is still tracked by git
  - contains the md5-hash of the original file
  - is used to keep dvc and git in sync
- moves the file to `.dvc/cache/` and links to it from its original position

command: dvc commit [path/to/file]

- if a file changes and you want to add the changes, run **dvc commit** to update it

command: dvc push

- pushes all dvc tracked data to a specified remote
- There are many backends available, **we have our own**

Demo Time: Introducing MinIO S3 Storage

Consequences of this approach

The workflow for using git + dvc changes to:

Sending data

1. dvc add
2. git add
3. git commit
4. dvc push
5. git push

Getting data

1. git pull
2. dvc pull

Switching branch

1. git checkout `BRANCH`
2. dvc checkout / pull

Problem 3

Same environment

We had a talk on this recently, a possible solution would be **nix**

Demo Time: A `flake.nix` file for us

Problem 4

Ensure the order of execution & not re-run all code every time

aka “Well, it worked on my machine OR oh, you need to run `prepare_data23.py` BEFORE `create_model.R`”

With **dvc** you can create a DAG of all steps of your research pipeline

- A pipeline is defined in stages which depend on one another

command: dvc repro [STAGE]

- Command is used to reproduce a research pipeline
- Re-runs only stages that have changed, used cached results for everything else
- Result caches are shared by using dvc

What does this mean?

We can run different parts of the pipeline on different computers (e.g., parts of it on a cluster)!

The pipeline is defined in a file called `dvc.yaml`

Pipeline is defined in stages

- Each stage can have **dependencies** and **outputs**. These determine the DAG.

The pipeline is defined in a file called `dvc.yaml`

Pipeline is defined in stages

- Each stage can have **dependencies** and **outputs**. These determine the DAG.
- A dependency can be any hashable object, e.g.:
  - outputs of previous stages (This is how we define the DAG)
  - folders with data in it
  - python / R files with code in it

The pipeline is defined in a file called `dvc.yaml`

Pipeline is defined in stages

- Each stage can have **dependencies** and **outputs**. These determine the DAG.
- A dependency can be any hashable object, e.g.:
  - outputs of previous stages (This is how we define the DAG)
  - folders with data in it
  - python / R files with code in it
- Each stage has a **single command** that is executed, e.g.:
  - python pipeline/some_python_script.py
  - Rscript pipeline/some_r_script.R

Demo Time 1: Show `dvc dag`

Demo Time 2: A look at the `dvc.yaml`