Develop ML Projects with DVC

H Swathi Shenoy

6 min read · Oct 20, 2023

Learn how to use DVC to version data, create models, and automate pipelines.

Image generated with the help of Stable Diffusion, taken from Iterative.

As a data scientist, you frequently work with large amounts of data. Data that has been processed, divided into train, validation, and test sets. Additionally, model files and configurations are also involved. Further, when multiple people are collaborating on the development, we must ensure that the correct data is being used for reproducibility. In this case, DVC is a lifesaver!

Before using any tool, we must understand what kind of problem it solves. I must admit that when I first started working, I wasn't a "disciplined" data scientist. My supervisor made a frightfully funny statement later that day: "I'm going to delete your code repo, and disappear. Let me know what you're going to do next". Since then, versioning the code is now a must-do procedure.

> *What is a DVC?*

DVC is a MLOps Swiss army knife. A version control system for machine learning projects that encourages structured development, collaboration, reproducibility and deployment.

Version Control Systems allows the developers to track the changes or developments in the project and gives them liberty to revert to previous state when there is flaw observed in the current state. Also, it gives some kind of happiness when there is an up to date backup of your work.

Unlike software projects, source code is not the only input to machine learning, but data. And data can get heavy. With Git versioning data and model wont be feasible as:

1. Data and Models are way too big to upload to git
2. Use external storage, but it loses the idea of versioning.

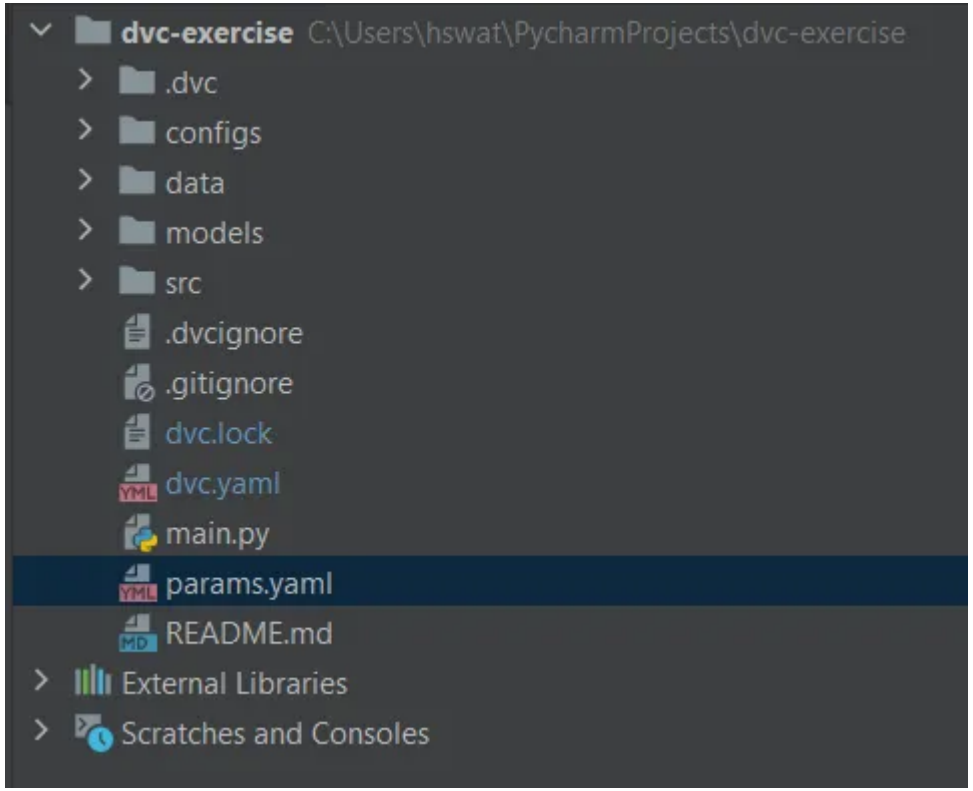
This is precisely what DVC Solves! It is built on top of Git and thus provides versioning capabilities for data, models, or images. Further, you even track experiments and reproduce them. Let's look in detail at how we do that.

> *Hands-On Tutorial for DVC*

At first, we need to install a few libraries before starting to use DVC. One of course is the *DVC* itself, and *DVC-s3* to configure the remote storage, here I have used AWS S3.

```
pip install dvc dvc-s3
```

For better understanding, I'm going to take -up an easy ML template that will involve FTI (Feature-Training-Inference) pipelines. The template will look like this. (Ignore the dvc folder/files for now)



ML Template

1. data: Store all kinds of data i.e. raw/processed data which you can store as subfolders within this folder.
2. configs: Store all the model configurations.
3. models: Store the artifacts/feature-related objects here.
4. src: Your codebase i.e. scripts goes here.

> *The TL;DR; (if you are in hurry)*

The main commands to start versioning your data are (*Considering you have already started versioning your code with git*):

1. First, configure your remote storage i.e. s3 storage(prior to this, initialize your DVC)

```
dvc init
dvc remote add -d dvc-remote s3://<bucket>/<key>
```

2. Now, add the folder/file i.e. the data you need to version with the following command

```
dvc add <folder/file-name>
```

3. Notice that dvc has automatically *created a file with a .dvc* extension. Now, DVC has started tracking your file/folder. It consists of the metadata

describing what datasets, ML artifacts, etc. to track in remote storage. This metadata can be put in Git in lieu of large files

4. Now, finally push that .dvc file prior to committing and push it to git.

```
dvc push
```

5. So, as you switch between your features as in branches/commits; changes get reflected in the metadata of .dvc file and you do run dvc pull.

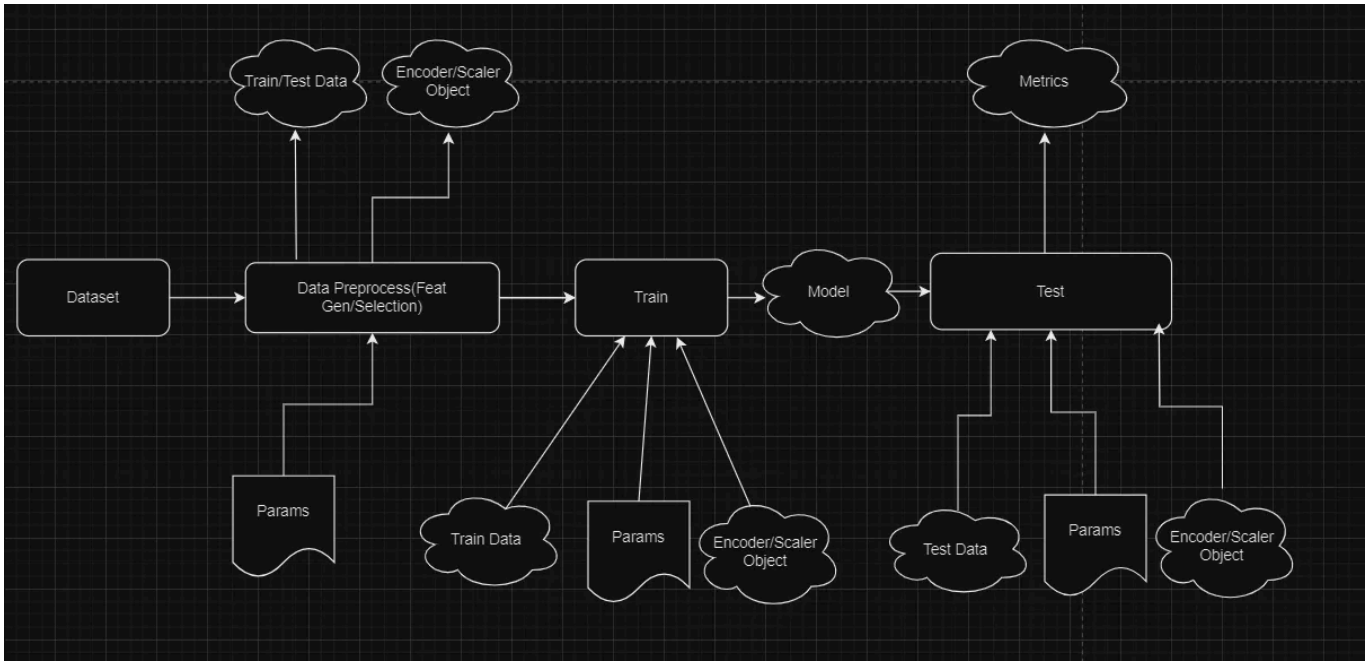
```
dvc pull <filename.dvc>
```

That's how DVC does the data versioning.

> *Data Pipelines.*

There are several pipelines involved when working on machine learning projects, including the training, inference, feature, and data preparation pipelines. It can be time-consuming to add inputs and outputs one by one for each pipeline because each are unique. What a way to automate that!

DVC pipelines can come in handy in this case. A *dvc.yaml* file must be defined, in which stages must be defined along with their corresponding inputs and outputs. and even monitor the graphs and metrics. How amazing is that!?



The ML development pipeline with inputs and outputs specified.

It is very clear from the above figure how the pipelines and its corresponding Inputs/Outputs from each stage work.

So to make sure you can get a sense of it, let me demonstrate the dvc.yaml file template.
Note: I built a complete machine learning pipeline using a simple IRIS data as my dataset.

```
stages:
  prepare:
    cmd: python src/load.py raw/IRIS.csv
    deps:
    - data/raw/IRIS.csv
    - src/load.py
    params:
    - load.split
    - load.random_state
    outs:
    - data/prepared
```

1. First, define the step “stages” under which each pipeline is defined.
2. Each stage is followed by a “cmd” is essentially a Python script with arguments, or inputs, for that particular script.

3. You specify the dependencies (Inputs) later on. The same arguments you passed in the last “cmd” command, basically. Keep in mind that the script itself is a dependent.
4. “Params” is for the configurations the scripts needs i.e. configs
5. Finally, “outs” refer to the results obtained from each stage’s execution.

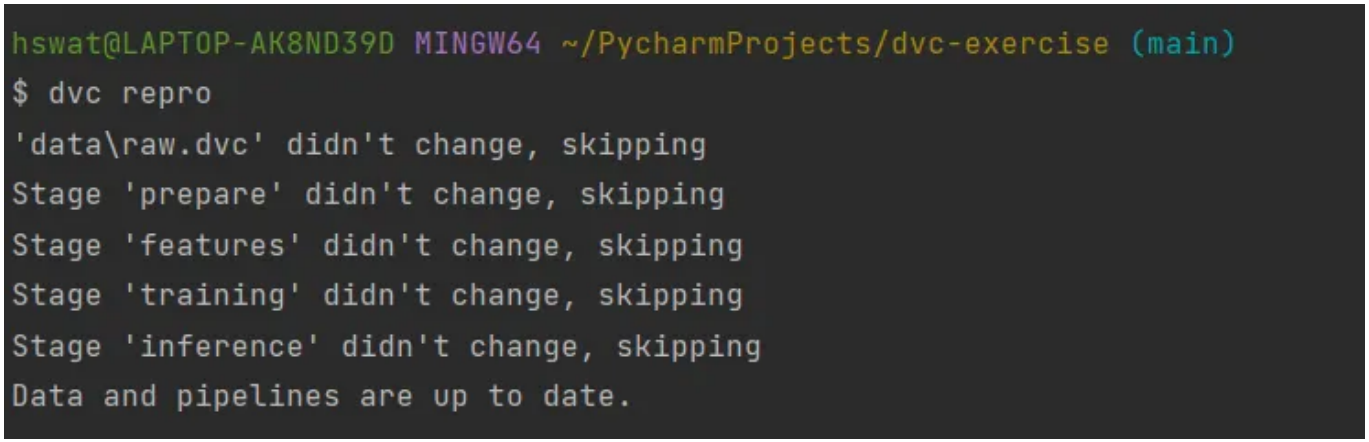
```
stages:
  prepare:
    cmd: python src/load.py raw/IRIS.csv
    deps:
    - data/raw/IRIS.csv
    - src/load.py
    params:
    - load.split
    - load.random_state
    outs:
    - data/prepared
  features:
    cmd: python src/features.py data/prepared processed encoder features
    deps:
    - data/prepared/train.csv
    - src/features.py
    params:
    - features.max_features
    outs:
    - data/processed
    - models/encoder
    - models/features
```

```
  training:
    cmd: python src/model.py data/processed artifacts features
    deps:
    - data/processed/train-norm.csv
    - data/processed/encoded-labels.csv
    - models/features/features.yaml
    - src/model.py
    outs:
    - models/artifacts
  inference:
    cmd: python src/inference.py data/prepared features artifacts encoder
    deps:
      - data/prepared/test.csv
      - models/features/features.yaml
      - models/artifacts/model.pkl
      - models/encoder/scaler_encode.pkl
      - models/encoder/label_encode.pkl
    metrics:
        - data/evaluate/metrics.json
```

Note the *metrics* stage at the final “inference” stage. To keep track of the metrics for every experiment, DVC offers `dvc live .`

The code snippet above contains all of the dvc.yaml needed for running an end-to-end FTI pipeline. To execute the yaml file, run following command.

```
dvc repro
```



Output for dvc repro

To track the experiments within each experiment , run

```
dvc metrics show
```

```
hswat@LAPTOP-AK8ND39D MINGW64 ~/PycharmProjects/dvc-exercise (main)
$ dvc metrics show
Path                        avg_prec.test    avg_recall.test
data\evaluate\metrics.json  1.0              1.0
```

Output for metrics tracker.

As the pipeline is run , *dvc.lock* file gets generated.

To record the state of your pipeline(s) and help track its outputs, DVC will maintain a `dvc.lock` file for each `dvc.yaml` . Their purposes include:

- Allow DVC to detect when stage definitions, or their dependencies have changed. Such conditions invalidate stages, requiring their reproduction.
- Tracking of intermediate and final outputs of a pipeline — similar to `.dvc` files.

You only need to add the `dvc.lock` file to your current commit once data - pipelines are updated. Consequently, the ongoing experiment is saved on git.

Final thoughts

Our projects can really go to the next level if we use DVC to manage Machine Learning pipelines and track experiments. Creating separate Python scripts

for every stage and clearly defining the parameters, inputs, and outputs that each script uses are essential for machine learning projects reproducible. Thus, use `dvc repro` to execute the pipeline however you see fit.

You can find the entire code base in here.

DVC provides pipeline management, experimentation, and version control in addition to collaboration and deployment features. I'll try to include more about it in the upcoming article!

Thank you for reading, and that's everything for now! 😁

References:

Developing ML Projects with DVC

The first in a series on how to build machine learning projects with DVC. Part 1: The Need for DVC.

www.ridgerun.ai

https://dvc.org/doc

Data Pipeline Dvc Data Versioning

Written by H Swathi Shenoy

24 Followers · 20 Following

Edit profile

I write to gain clarity. Senior MLE | Python Developer.

No responses yet

What are your thoughts?

Respond

More from H Swathi Shenoy

In Dev Genius by H Swathi Shenoy

AWS CodeBuild/CodePipeline 👉 Deploy Lambda Tutorial🔥

Automatic deployment of lambda code(based on docker image)via CodeBuild...

Jul 4, 2024 5

H Swathi Shenoy

Build Scikit-learn Prediction Pipeline with Custom Transforme...

Learn how to build Custom Transformers/Estimators with...

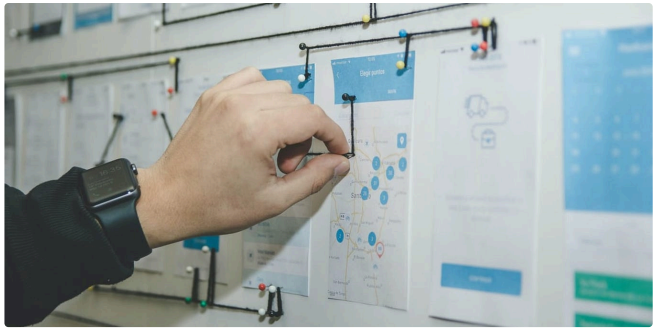
Jul 8, 2023 54

H Swathi Shenoy

Validate your Data with Pydantic

Learn how to use Pydantic's custom validators to validate your data

Feb 10, 2023 3



In Dev Genius by H Swathi Shenoy

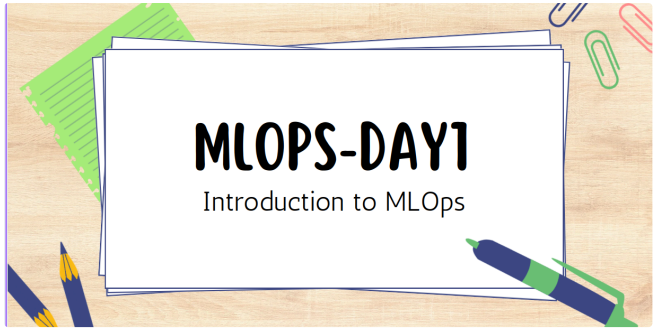
MLFlow Tutorial -ML Tracking and Serving

Detailed walkthrough of the process to log your ML experiments(using MLFlow...

Dec 4, 2023 9

See all from H Swathi Shenoy

Recommended from Medium



Siddharth Singh

MLOPS Day1: Introduction to MLOps

Bridging Machine Learning and Operations for Scalable, Reliable AI Systems

Nov 26, 2024 40 1

FinanceAndCode

AI Engineer Career Roadmap: A Comprehensive Guide

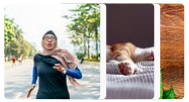
The field of Artificial Intelligence (AI) is rapidly evolving, offering myriad opportunities for...

Sep 22, 2024

Lists

Staff picks
812 stories · 1622 saves

Stories to Help You Level-Up at Work
19 stories · 938 saves



Self-Improvement 101
20 stories · 3299 saves

Productivity 101
20 stories · 2783 saves

Bonny Ophelie

MLOps: Integrating DevOps Practices into AI/ML Pipelines

As machine learning (ML) models become integral to business decision-making,...

Sep 2, 2024 11

Komal Agrawal

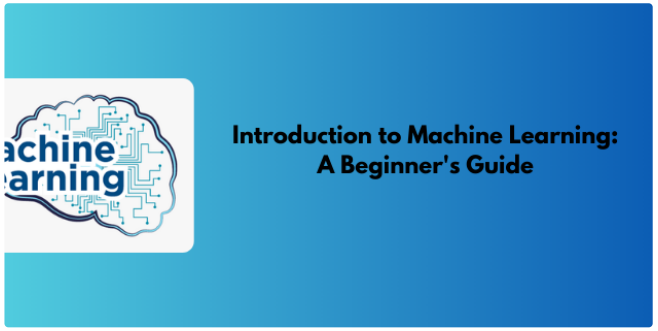
MLOps

MLOps, or Machine Learning Operations, is a set of practices that combines machine...

Oct 13, 2024 21

Christopher Adamson

Implement End-to-End MLOps with SageMaker Projects



Mohsin Rubel

Welcome to the first part of our Introduction to Machine Learning...

Implementing robust machine learning pipelines remains a challenge for many...

Sep 2, 2024 134

Welcome to the first part of our Introduction to Machine Learning Tutorial. This series is...

Nov 28, 2024 3

See more recommendations