DVC

11 January 2025 19:46

What is DVC (Data Version Control)?

DVC (Data Version Control) is an open-source version control system for managing machine learning (ML) projects. While Git is used for tracking code changes, DVC extends this functionality to handle large datasets, models, experiments, and other machine learning artifacts that don't fit well in Git due to their size or nature. It helps manage data pipelines, models, and ensures reproducibility in machine learning workflows by allowing teams to version control data files, track experiments, and collaborate on complex ML projects.

DVC works on top of Git, and it integrates seamlessly with Git repositories. It provides functionalities like:

1. **Data Management**: Tracks large datasets and other binary files that can't be tracked directly by Git.

2. **Pipeline Management**: Defines, runs, and tracks data processing pipelines to ensure reproducibility.

3. **Experiment Management**: Helps in organizing and comparing different machine learning experiments.

4. **Collaboration**: Allows teams to collaborate on data and models by sharing pointers to large files via cloud storage, without the need for direct file transfer.

Key Concepts in DVC:

1. **DVC Files**: DVC uses a set of files (such as .dvc files) to track data and models. These files act as pointers to the actual data stored in external storage systems like AWS S3, Google Drive, or a local file system.

2. **DVC Remote**: A remote storage location where DVC stores the large datasets and models.

3. **DVC Pipeline**: A series of steps to process data, such as training a model, testing it, or evaluating it. Each step is tracked and versioned by DVC.

4. **DVC Cache**: A local cache where DVC stores copies of files being tracked. This speeds up the process of retrieving files and sharing data.

5. **DVC Command**: Commands for versioning, tracking, and managing machine learning data and experiments.

Init, add, push, pull

13 January 2025 17:28

DVC Commands Explained in Depth

1. dvc init

- Initializes a DVC repository. This command sets up the necessary .dvc directory and configurations to start version controlling data.

```
dvc init
```

- **Description**: After running this command, a .dvc directory is created, and a .dvc/config file is initialized for setting up DVC configurations.
- **Use Case**: Typically used when starting a new project with DVC. It creates a version-controlled environment for managing datasets and experiments.

2. dvc add

- Tracks large files or directories with DVC. This command adds files to the DVC cache and creates a .dvc file to track them.

```
dvc add <file-or-dir>
```

- **Description**: This command makes DVC aware of large files or datasets (like CSVs, images, or model files) that should be version-controlled but are too large for Git.
- **Example**:

```
dvc add data/raw_data.csv
```

- This will track the file raw_data.csv and create a raw_data.csv.dvc file to store metadata about the file (such as hash values, file size, etc.).

3. dvc push

- Uploads data files (or model files) from the local DVC cache to the remote storage.

```
dvc push
```

- **Description**: This command uploads your tracked files to a DVC remote (like AWS S3, Google Drive, or a local server) so that others can access the data. DVC uses the .dvc files to know where to push the data.
- **Use Case**: When you want to share large datasets or models with your team or back them up in a cloud service.

4. dvc pull

- Downloads the latest version of the files from the DVC remote storage to your local system.

```
dvc pull
```

- **Description**: This retrieves the data from the remote storage based on the information stored in the .dvc file. You can think of it as a way to sync the data you need to your local machine.
- **Use Case**: When you or a teammate needs to download the latest version of datasets or models from remote storage.

Status, run, repro, checkout

13 January 2025 17:29

5. dvc status

- Shows the current status of tracked files and the remote storage.

```
dvc status
```

- **Description**: This command checks whether the local data files match the version stored in the DVC remote. It helps in identifying whether you need to run dvc pull or dvc push.
- **Use Case**: To see if your local repository is synchronized with the remote storage or if there are updates to pull or push.

6. dvc run

- Used to define a stage in the DVC pipeline (e.g., data processing, model training).

```
dvc run -n <stage-name> -d <input-file> -o <output-file> <command>
```

- **Description**: This command defines an executable pipeline step. For example, you can define a step to pre-process data or train a model. DVC tracks dependencies (-d) and outputs (-o) for each stage.
- **Example**:

```
dvc run -n preprocess -d data/raw_data.csv -o data/processed_data.csv "python preprocess.py"
```

- This command defines a pre-processing step, tracks the raw_data.csv as input, and the processed_data.csv as output. It also associates a command (python preprocess.py) to run.

7. dvc repro

- Reproduces the pipeline stages, executing commands defined by dvc run.

```
dvc repro
```

- **Description**: This command reruns the pipeline stages that have changes or need to be executed. It automatically checks for changes in the dependencies and regenerates outputs if necessary.
- **Use Case**: After modifying your code or data, dvc repro helps you recreate the results from the pipeline.

8. dvc checkout

- Updates the workspace to match the files defined by the current DVC state (either from a branch, tag, or commit).

```
dvc checkout
```

- **Description**: It updates your workspace with the right versions of files from the DVC cache based on the current branch/commit.
- **Use Case**: When you switch between branches or commits, dvc checkout ensures your workspace reflects the correct versions of data files.

Remote, diff, gc, lock

13 January 2025 17:32

9. dvc remote

- Manages DVC remote storage locations. You can configure, list, or remove remotes.

```
dvc remote add -d <remote-name> <remote-URL>
```

- **Description**: This command allows you to add or configure remote storage for DVC. You can link cloud storage (S3, GCP, etc.) or a local server to the DVC project.
- **Example**:

```
dvc remote add -d myremote s3://mybucket/myfolder
```

- This command adds an S3 bucket as a remote storage location.

10. dvc diff

- Compares two versions of the tracked files to see what has changed.

```
dvc diff <commit1> <commit2>
```

- **Description**: This command allows you to compare different versions of files tracked by DVC. It shows differences between output files, datasets, or models between two commits or branches.
- **Use Case**: Useful when you want to see how your model or data has changed between commits or experiment versions.

11. dvc gc (Garbage Collection)

- Removes unused data and cache to free up space.

```
dvc gc
```

- **Description**: DVC's garbage collection removes files that are no longer referenced in the cache or remote storage. It helps in cleaning up unused data and optimizing space.
- **Use Case**: After finishing an experiment or when you know that certain datasets are no longer needed, you can clean up old or unused files to optimize storage.

12. dvc lock

- Locks the state of files for reproducibility across team members.

```
dvc lock
```

- **Description**: Ensures that the exact data versions, pipeline dependencies, and outputs are fixed and reproducible. This is important when collaborating in teams or working with large experiments.

Conclusion

DVC is a powerful tool for managing machine learning data and experiments with Git-like version control but optimized for large datasets and binary files. It enables you to track datasets, models, and pipeline stages, ensuring reproducibility and collaboration. By using commands like `dvc add` , `dvc push` , `dvc run` and `dvc repro`, you can efficiently manage and share data, optimize workflows, and collaborate with your team. Understanding DVC's commands and workflow will help you maintain clean, reproducible, and scalable machine learning projects.