

DVC - Tutorial:

Setup:

Follow the instructions in the following link to install DVC 2.0

[How To Install | Data Version Control · DVC](#)

DVC Basics:

Assuming that DVC has been initialised along with Git, to start tracking a file or a directory the following command is run: `dvc add filename`. The tracked file information is then stored in a ".dvc" file which is a placeholder for the tracked file which then can be versioned just like a source code with Git:

- `$ git add filename.dvc filename/.gitignore`
- `$ git commit -m "Add filename data"`

Note: You will not need to version individual assets such as datasets, features etc. Rather a DVC pipeline (dvc.yaml file) has already been created for the purpose of this experiment and the variations made to the pipeline and its data are automatically tracked by DVC.

[Getting Started with DVC](#)

Structure of the experiment project

We define a DVC pipeline in the dvc.yaml file. The experiment files addressed in the config files are structured as shown below.

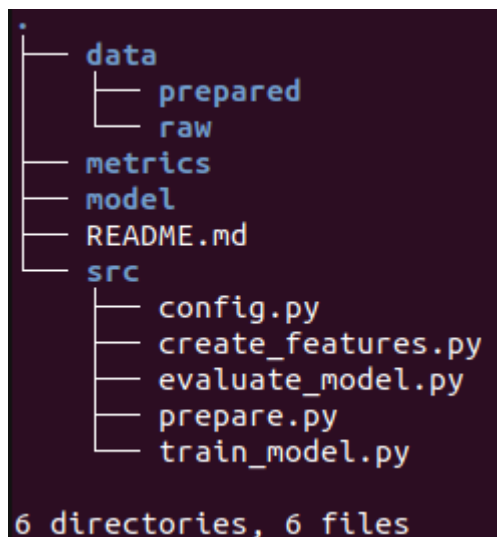


Figure A: Structure of the experiment directory.

Dataset

When the dataset gets generated in the *prepare.py* and extracting features in *create_features.py*:

- The dataset gets fetched and split in the *prepare.py* file. And the original and split datasets get saved to “data/raw” and “data/prepared” respectively. (Check lines 20 and 25-26).
- In the *create_features.py* file, we extract the features we want to use, upon doing that the newly generated train and test features get saved in the “data/feature” directory, (Figure A. also see lines 18 and 19).

Model

In the *train_model.py*, we train a model and that model gets saved as a pickle file to “model/directory”. (Figure A)

Features

In the *create_features.py* we extract the features with which we want to train the model with.

Parameters

The parameters you used during the experiment are all saved in *params.yaml* all under the stage in which you use them in.


Evaluation Metrics

In the *evaluate.py* the model performance is calculated, and evaluation metrics are generated. These generated metrics get saved as a JSON file and saved in the metrics/directory.

Experiment Management

Tracking Changes

We are going to use the “*\$dvc exp*” family of commands to track, list and compare changes made during our controlled experiment. The *dvc exp* commands let you automatically track a variation to an established data pipeline. You can create multiple isolated experiments this way, as well as review, compare, and restore them later, or roll back to the baseline. The basic workflow goes like this:

- Modify stage parameters or other dependencies (e.g. input data, source code) of committed stages.
- Use [dvc exp run](#) to execute the pipeline. The results are reflected in your workspace, and tracked automatically.
- Use [metrics](#) to identify the best experiment(s) (See section on retrieving details of experiment runs).
- Visualize, compare experiments with [dvc exp show](#) or [dvc exp diff](#). Repeat 
- Use [dvc exp apply](#) to roll back to the best one.
- Make the selected experiment persistent by committing its results to Git. This cleans the slate so you can repeat the process.

You can assign custom names to your experiment runs when using the “dvc exe run” by using the -n option. E.g., “\$dvc exp run -n run1” specifies the name of the experiment as “run1” instead of a default name based on the experiment's hash.

[DVC Experiment Command Reference](#)

Retrieving Details of Experiment Runs

Show experiment table:

DVC provides an option to obtain an overview info on completed runs by using the “\$dvc exp show”. This command prints a customizable table of experiments, their metrics and parameters.

```
$ dvc exp show
```

Experiment	avg_prec	roc_auc	train.n_est	train.min_split
workspace	0.56191	0.93345	50	2
master	0.55259	0.91536	50	2
├─ exp-bfe64	0.57833	0.95555	50	8
└─ exp-ad5b1	0.56191	0.93345	50	2

You can also filter the assets to return from the “exp show” command. E.g., you can specific parameters or metrics as shown below

The option “--include-params=featurize” limits the param columns to only include the featurize group

```
$ dvc exp show --include-params=featurize
```

Experiment	Created	auc	featurize.max_features
workspace	–	0.61314	1500
10-bigrams-experiment	Jun 20, 2020	0.61314	1500
├─ exp-e6c97	Oct 21, 2020	0.61314	1500
├─ exp-1dad0	Oct 09, 2020	0.57756	2000
└─ exp-1df77	Oct 09, 2020	0.51676	500

Similarly, you can use the “--include-metrics <list>” to display specific metrics.

List experiment run:

You can also list the names of completed experiments under the current branch of the DVC repository.

```
$ dvc exp list --all
10-bigrams-experiment:
    exp-e6c97
    exp-1dad0
    exp-1df77
```

Restore prior experiment:

You can also retrieve assets from any of the experiment runs by simply reverting back to the earlier run and inspect the state of the assets at that particular run using the “exp apply” command. Once you have the desired working directory, you can simply navigate to the location of a specific asset, as described in the section named “Structure of the project section” below.

```
$ dvc exp apply exp-e6c97
Changes for experiment 'exp-e6c97' have been applied...
```

Comparing Evaluation Metrics

DVC provides yet another CLI command to directly inspect and diff metrics between two runs using `$dvc metrics show` and `$dvc metrics diff <rev1> <rev2>`. See examples below.

```
$ dvc metrics show
eval.json:
    AUC: 0.66729
    error: 0.16982
    TP: 516
```

```
> dvc metrics diff run6 run2
```

Path	Metric	run6	run2	Change
metrics/metrics.json	mean_absolute_error	67.81391	63.41259	-4.40132
metrics/metrics.json	r_squared	0.06745	0.08112	0.01366
metrics/metrics.json	rmse	<u>78.34447</u>	75.7472	-2.59728

Comparing parameters between two revisions

Similar to metrics, you can also compare parameters of two revisions using the “\$dvc params diff”. The example below shows the comparison between the parameters prepare.split and train.normalize in run4 and run2.

```
> dvc params diff run4 run2
Path      Param      run4      run2
params.yaml prepare.split 0.3       0.2
params.yaml train.normalize True      False
```

Comparing Experiments between two revisions

“\$dvc exp diff” provides a quick way to compare both parameters and metrics between two experiments. The example below compares experiment exp-1dad0 and exp-1df77.

```
$ dvc exp diff exp-1dad0 exp-1df77
Path      Metric  exp-1dad0  exp-1df77  Change
scores.json auc      0.577559   0.51676    -0.060799

Path      Param      exp-1dad0  exp-1df77  Change
params.yaml featurize.max_features 2000       500        -1500
```

Links:

<https://dvc.org/doc/command-reference/exp>

<https://dvc.org/doc/command-reference/params>

<https://dvc.org/doc/command-reference/metrics>