Experiment management with `dvc exp run`

In previous lessons, we learned how to run our pipeline with `dvc repro`. This command checks for changes in our code, parameters, and data, and runs the pipeline downstream from the first change. There is, however, another command we can use to trigger a pipeline run: `dvc exp run`.

The `exp` in this command stands for *experiment*, and it provides a clue to the intended use of this command. As we discussed previously in lesson 3.2, we can consider a single run of our pipeline to be an experiment: we have changed something in our configuration and will now see how that change affects our results.

Numerous DVC features are specific to experiments:

- We can update parameters right from our CLI
- We can queue experiments and run them subsequently
- We can compare many experiments against each other
- We can persist experiments through individual Git branches
- We can share experiments

Changing parameters in experiments

Rather than manually changing a parameter in our `params.yaml` file, we can change a parameter for an experiment with the -S option:

```
dvc exp run -S prepare.split=0.25 -S featurize.max_features=2000
```

In this example, we run the experiment with a `split` value of `0.25` in the `prepare` stage and a `max_features` value of `2000` in the `featurize` phase, regardless of the values specified directly in our parameter file.

In the background, DVC changes `params.yaml` automatically. That means that if we are happy with the experiment results, we can commit it with updated parameters.

Queueing experiments

We can queue experiments if we want to run multiple experiments after each other. This can be helpful when we are tuning hyperparameters, for example. When combined with a script to generate different settings for hyperparameters, DVC can help us speed up grid searches.

```
dvc exp run --queue -S featurize.max_features=20
dvc exp run --queue -S featurize.max_features=1000
dvc exp run --queue -S featurize.max_features=2000

dvc exp run --run-all --jobs 2
```

In the example above, we specify three experiments with different settings for `max_features`. We then run all of these queued experiments with `--run-all`. We even parallelize them with `--jobs 2`. This speeds up the rate at which we can conduct multiple experiments.

Evaluating experiment metrics

Once we have run our experiments, we can take a step back and evaluate the results of those experiments. We can compare the metrics of the experiments in our workspace with `dvc exp show`, which displays a table containing our experiments, evaluation metrics, and optionally experiment parameters. For example:

```
dvc exp show --only-changed
```

```
 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
```

This high-level overview allows us to compare experiments at a glance and notice trends in our results.

Persisting experiments

Once we have discovered which experiments are successful, we can decide to persist those experiments in our workspace. In the example above, `exp-e6c97` yields the best `auc`. We can apply the changes to the parameters used in this experiments to our workspace, and create a Git commit to make sure they are included in our Git history:

```
dvc exp apply exp-e6c97
```

If we want to make a branch out of a certain experiment, DVC has also got us covered. We can use `dvc exp branch` to do so:

```
dvc exp branch exp-e6c97 maxf-1500
```

Here we take experiment `exp-e6c97` and create a branch called `maxf-1500`. We can then use this branch in every way we would usually use a branch: synchronize it with our remote repository, maybe merge it to `main`, or experiment further from here on out.