Group C -DVC Experiment Tasks

We will experiment with two different algorithms, RandomForestRegression and Linear Regression. A reproducible pipeline has been created and the stages are stored in the *"dvc.yaml"*. You will use the $dvc exp run" command to execute a new run whenever you make changes to the pipeline files.

Explore the dvc.yaml file where the experiment stages are defined. The first stage of the pipeline is preparing, and that will load the dataset and split them into train and test .csv files. The creating features stage will extract features from the dataset. The train_model will train the model and output a model. Finally, the evaluation stage will generate the evaluation metrics and output them as a JSON file.

DVC supports flexible ways of tracking and managing ML experiments. In this controlled experiment, we are only interested in how DVC helps with tracking and querying variations of an experiment pipeline. Hence, it is not required to make our experiment persistent to Git.

DVC experiment commands reference

Preparation

Initialize DVC

1. Go into the dvc-experiment-c folder.
2. Initialise Git
   ```
   $ git init
   ```
3. Initialise DVC: This will initialise DVC in your project and will add the DVC folder in the repo.
   ```
   $ dvc init
   ```
4. Committing the changes.
   ```
   $ git add .
   $ git commit -m "Add DVC config"
   ```

Experimenting with algorithm 1 (Linear Regression)

Carry out the following experimental runs using different sets of dataset features and learning parameters as described below. Execute the python script at each run to train and evaluate the model performance by running **`$dvc exp run`**.

Use the *-n <name>* option of the *$dvc exp run* option to add a readable name for each run. After each run, you can use the $dvc exp show or $dvc exp list to see completed runs.

1. **Run 1**: Open the terminal and navigate into the ***dvc-experiment-c*** folder. What you're going to do first is reproduce the dvc.yaml by running the following command:
   $ dvc exp run -n run1

2. **Run 2**: In the params.yaml file, change the Linear Regression normalize parameter to False, and run the experiment again.
   $ dvc exp run -n run2

3. **Run 3**: In the Boston dataset we have a total of 13 features:

```
CRIM: Per capita crime rate by town
ZN: Proportion of residential land zoned for lots over 25,000 sq. ft
INDUS: Proportion of non-retail business acres per town
CHAS: Charles River dummy variable (= 1 if tract bounds river; 0
otherwise)
NOX: Nitric oxide concentration (parts per 10 million)
RM: Average number of rooms per dwelling
AGE: Proportion of owner-occupied units built prior to 1940
DIS: Weighted distances to five Boston employment centers
RAD: Index of accessibility to radial highways
TAX: Full-value property tax rate per $10,000
PTRATIO: Pupil-teacher ratio by town
B: 1000(Bk − 0.63)², where Bk is the proportion of [people of African
American descent] by town
LSTAT: Percentage of lower status of the population
MEDV: Median value of owner-occupied homes in $1000s
```

   Change the "**RM**" to "**RAD**" in create_features.py in line 13.

   $ dvc exp run -n run3

4. **Run 4**: Modify the ratio of training to test data. Open the params.yaml file and change the split amount from 0.2 to 0.3.
   $ dvc exp run -n run4

5. **Run 5:** We will now use more features to train the model. Add `"s2"` in line 13 of create_features.py. Also, set the "normalize" parameter to True, that's in train_model.py
   $ dvc exp run -n run5

6. **Run 6:** We will use new features, add the following features to line 13 in *create_features.py:*
   `'DIS', 'B'`

   $ dvc exp run -n run6

7. **Run 7:** Revert back to the “run3” using the following command:
   $ dvc exp apply run3

   Now change the seed value that’s in “params.yaml” to 34556, and see if it affects the result. Run the experiment again
   $ dvc exp run -n run7

Experimenting with algorithm 2 (RandomForestRegression)

Now we try a different algorithm by

- Commenting line 16 and uncommenting line 17 in train_model.py
- Commenting the lr_parameters and uncommenting the rfr_parameters lines in the “params.yaml” file.

8. **Run 8:** Execute the experiment using the RFR algorithm.
   $ dvc exp run -n run8

9. **Run 9**: We are now going to use the following parameters: n_estimators, max_depth, min_samples_split, and ccp_alpha.

In train_model.py line 17, replace the **RandomForestRegressor()**, with the following code, and run the pipeline again.

```
# Copy this
model = RandomForestRegressor(
  n_estimators=params["n_estimators"], max_depth=params["max_depth"],
  min_samples_split=params["min_samples_split"],
ccp_alpha=params["ccp_alpha"],  max_features=params["max_features"],)
```

   $ dvc exp run -n run9

10. **Run 10:** In the params.yaml change the ccp_alpha and max_depth to 0.1 and 10 respectively.

$ dvc exp run -n run10

11. **Run 11:** Now we will add more features to train the model. Replace **`'LSTAT'`** with **`'AGE'`** in line 13, ***extract_features.py***, as shown below:

```
def extract_features(df):
    return df[['RAD', 'AGE', 'TAX', 'DIS', 'B']]
```

$ dvc exp run -n run11

12. **Run 12:** Let's try to improve the model. The default value of max_features is None. Change the max_features to **log2** in the params.yaml file.
$ dvc exp run -n run12

13. **Run 13:** Revert back to "run8" and include all features in the dataset as described in step 3 above.
$ dvc exp run -n run13

*Now return to the experiment's questionnaire