using_the_benchmark

June 13, 2023

[ ]:
```
from SCRBenchmark import BenchmarkSuite
```

1 Design of the SCRBenchmark Suite

The notebook generate_data.ipynb details how the `benchmark` class can be used to generate data for individual equation instances. Here, we detail how a full suite of benchmark sets can be generated and how we ensure repeatability and how we compare results.

Generating the full suite is as easy as follows:

```
from SCRBenchmark import BenchmarkSuite, FEYNMAN_SRSD_HARD, HARD_NOISE_LEVELS,HARD_SAMPLE_SIZES
BenchmarkSuite.create_hard_instances(target_folder='./Data'
                                     , Equations=FEYNMAN_SRSD_HARD
                                     , sample_sizes= HARD_SAMPLE_SIZES
                                     , noise_levels=HARD_NOISE_LEVELS
                                     , repetitions= 10 )
```

1.1 Static Validation Sets

Unlike the generation of training data, where we want data in a distribution similar to real-world occurrences, for the validation set we desire maximized coverage of the input domain to evaluate the extrapolation capabilities of the trained model. As guided behavior is one of the key benefits of shape-constrained regression and the introduction of prior knowledge in the form of shape constraints.

To facilitate comparison of multiple training runs and different algorithms we reuse one static validation dataset for each equation. This data is generated by uniform sampling of a large size of data from the full defined input domain. The target is calculated by evaluating the known base equation on the input data, and we do not introduce artificial noise on the validation data. See `generate_data.ipynb` for more detailed information on the sampling methodology.

The validation sets are shipped as fixed csv files in `../SCRBenchmark/Data/Test` with one file per equation. When we generate benchmark data, we sample training data from the log10 based distribution and simply append the contents of the responding validation set.

1.2 Seeded Training Data

We provide a fixed range of seeds to ensure repeatability in sampling the training sets. Thereby, we ensure a fair benchmarking of algorithms.

We sample new training data for each repetition. Therein, each repetition seeds `np.random.seed(xyz)` it's associated integer number.

```
SEEDS = [
 342229   ,1271677  ,571939   ,926645   ,2300754  ,747148   ,749940   ,1605390  ,1112752  ,156
,903911   ,1209292  ,2301474  ,772368   ,1565092  ,1676573  ,1623234  ,58404    ,1449071  ,147
]

#...
def create_hard_instances( target_folder = './data',
                           Equations = FEYNMAN_SRSD_HARD,
                           sample_sizes = HARD_SAMPLE_SIZES,
                           noise_levels = HARD_NOISE_LEVELS,
                           repetitions = None):
  #... iterate over equations, sample_sizes, selected noise_levels and repetitions
  BenchmarkSuite.create_individual_dataset(target_folder,
                                           benchmark,
                                           equation_folder,
                                           noise_level,
                                           sample_size,
                                           seed = SEEDS[repetition],
                                           sampling_patience = 40,
                                           )
  #...
```

The seed is fixed before sampling data and sampling the random noise.

```
def create_dataset(self, sample_size,  noise_level = 0, seed = None, patience = 10 ):
        assert (0<=noise_level and noise_level<=1), f'noise_level must be in [0,1]'

        # fixing seed before data and noise sampling if seed is provided
        if(not (seed is None)):
          np.random.seed(seed)

        xs = self.equation.create_dataset(sample_size,patience)

        if(noise_level>0):
          std_dev = np.std(xs[:,-1])
          xs[:,-1] = xs[:,-1] + np.random.normal(0,std_dev*np.sqrt(noise_level),len(xs))

        return (xs, self.read_test_dataframe().to_numpy())
```

As long as the user selected the same adjustable parameters `noise_level, sample_size` they are guaranteed to receive the same resulting dataset.

`../tests/check_benchmark_generation.py` asserts that a generating the dataset for a full benchmark suite is guaranteed to produce the same data.