The temporal overfitting problem with applications in wind power curve modeling

Datasets:

The paper uses two groups of wind turbine datasets, referred to as:

1. *case_study_1*: This dataset is available in the supplementary material of the Ding(2019) book, Data Science for Wind Energy. We use Dataset #6 (Inland and Offshore Wind Farm Dataset2) downloaded from the website https://aml.engr.tamu.edu/book-dswe/dswe-datasets/. The datasets are in `CSV` file format with eight columns. The first row of the dataset contains the headers for each column.
2. *case_study_2*: This dataset comprises of thirty inland wind turbine data for a period of roughly one and half years. This dataset is provided through our industry collaborator. The data has been anonymized by normalizing the wind power and removing other identifiers from the raw data. This dataset can be downloaded from https://github.com/TAMU-AML/Datasets/tree/master/TemporalOverfitting. These datasets are also in `CSV` file format and contain nine columns. The first row contains the headers for each column.

The meaning of the column headers are available at the following URL in the README file: https://github.com/TAMU-AML/Datasets/tree/master/TemporalOverfitting

Code:

The results generated in the paper uses two scripting languages: `R` and `MATLAB`, with some packages and toolboxes. The codes are computationally and memory intensive, thus it is advised to use some high performance computing cluster to reproduce the results. A quick output of the tables and the figures can be generated from the pre-computed results by executing the shell script `generateFiguresAndTables.sh` without running any other script.

Folder structure:

The `root` folder contains the following sub-folders: (P.S. if any of the given sub-folders is missing in the root folder available to you, please create those missing sub-folders before executing the code. The code doesn't create the sub-folders and will return an error.)

- `algorithms`: A folder that contains the core code files used to implement all the methods described in the paper.
- `case_study_1`: An empty folder. The four Case Study 1 datasets must be copied to this folder before reproducing the results.
- `case_study_2`: An empty folder. The thirty Case Study 2 datasets must be copied to this folder before reproducing the results.
- `intermediate_results`: A folder that stores all the computational results and logs necessary to generate the figures and tables in the paper.
- `results`: A folder that stores the final figures and tables.

Note: The code that are in the `root` folder are either wrappers for the core code files or generate the figures and tables using the results computed using the core files.

Core Functions:

There are eight methods used in the paper: `binning`, `kNN`, `AMK`, `tempGP` (proposed method), `regGP`, `TS-kNN`, `CVc-kNN`, and `PW-AMK`.

- Four of these methods are implemented in `R` (`binning`, `kNN`, `AMK`, `PW-AMK`) and the rest of the four methods (`tempGP`, `regGP`, `TS-kNN`, `CVc-kNN`) are in `MATLAB`.

- For kNN and AMK, we utilize an existing R package DSWE (Data Science for Wind Energy). PW-AMK also builds upon the AMK function of DSWE.
- tempGP, TS-kNN, and CVc-kNN are implemented using object-oriented programming in MATLAB, and the classes for these methods are in the folder: algorithms/classes/.
- regGP uses the builtin fitrgp function in MATLAB.

Softwares used: MATLAB version 2018b and R version 3.6.2

Libraries used:

- MATLAB toolboxes: Optimization Toolbox, Statistics and Machine Learning Toolbox, and Econometrics Toolbox.
- R libraries: DSWE package (version 1.5.1 or higher) available on CRAN at https://CRAN.R-project.org/package=DSWE.

Note: All the results are generated using Linux operating system on a high performance computing cluster.

Instructions

All the tables and figures can be reproduced using the code files except for Table 1 and 2 and Figure 3. These tables and figures were generated by hand and do not use any coding. The codes assume that all the scripts would be executed from the root folder, and the path inside the code files are set accordingly. The following workflow reproduces the figures and tables in the paper:

1. The code is wrapped using shell scripts. Most of the shell script assumes a multi-core parallelization on a single node (number of cores given in the table below). These scripts can be run on parallel on different nodes and does not interact with each other. The details of the number of cores and RAM required and expected runtime for each of the shell scripts are given below:

Shell Script File	Description	# Cores	RAM	Runtime
binning.sh	Computes binning results for all the turbines in Case Study 1 and 2	1	1GB	5 mins
knn_cs1.sh	Computes kNN results for all the turbines in Case Study 1	4	4GB	15 mins
knn_cs2.sh	Computes kNN results for all the turbines in Case Study 2	4	4GB	30 mins
amk_cs1.sh	Computes AMK results for all the turbines in Case Study 1	4	4GB	6.5 hours
amk_cs2.sh	Computes AMK results for all the turbines in Case Study 2	12	12GB	5 hours
tempGP_cs1_wt1.sh	Computes tempGP result for WT1 in Case Study 1	12 (1 GPU)	80GB	30 mins
tempGP_cs1_wt2.sh	Computes tempGP result for WT2 in Case Study 1	12 (1 GPU)	80GB	30 mins
tempGP_cs1_wt3.sh	Computes tempGP result for WT3 in Case Study 1	12 (1 GPU)	72GB	30 mins
tempGP_cs1_wt4.sh	Computes tempGP result for WT4 in Case Study 1	12 (1 GPU)	64GB	30 mins

Shell Script File	Description	# Cores	RAM	Runtime
`tempGP_cs2.sh`	Computes tempGP result for all the turbines in Case Study 2	12	32GB	2.5 hours
`ts-knn_cs1.sh`	Computes TS-kNN results for all the turbines in Case Study 1	4	4GB	1 hour
`ts-knn_cs2.sh`	Computes TS-kNN results for all the turbines in Case Study 2	4	4GB	2 hour
`cvc-knn_cs1.sh`	Computes CVc-kNN results for all the turbines in Case Study 1	4	4GB	10 mins
`cvc-knn_cs2.sh`	Computes CVc-kNN results for all the turbines in Case Study 2	4	4GB	15 mins
`pw-amk_cs1.sh`	Computes PW-AMK results for all the turbines in Case Study 1	4	4GB	4 hours
`pw-amk_cs2.sh`	Computes PW-AMK results for all the turbines in Case Study 2	12	12GB	2 hours
`tempGP_cs2_thin2.sh`	Computes tempGP result with `thinning number=2` for all the turbines in Case Study 2	12	32GB	20 hours
`tempGP_cs2_thin4.sh`	Computes tempGP result with `thinning number=4` for all the turbines in Case Study 2	12	32GB	9 hours
`tempGP_cs2_thin8.sh`	Computes tempGP result with `thinning number=8` for all the turbines in Case Study 2	12	32GB	5 hours
`tempGP_cs2_thin16.sh`	Computes tempGP result with `thinning number=16` for all the turbines in Case Study 2	12	32GB	2 hours
`tempGP_cs2_thin32.sh`	Computes tempGP result with `thinning number=32` for all the turbines in Case Study 2	12	32GB	1.5 hours
`tempGP_cs2_thin64.sh`	Computes tempGP result with `thinning number=64` for all the turbines in Case Study 2	12	32GB	1 hour
`regGP_cs2.sh`	Computes regGP result for all the turbines in Case Study 2	12 (1 GPU)	48GB	120 hours

One can also run all the shell scripts in the table sequentially by running the script `runAll.sh`. Please note that the computation time for running this script would be the sum of all the runtimes shown above.

2. The computed results and respective log files are stored in the sub-folder `intermediate_results/`. After all the aforementioned shell scripts are executed, one can generate the figures and tables in the paper using the script: `generateFiguresAndTables.sh` on any computer. For quick compilation of the pre-computed results, one can execute the same script `generateFiguresAndTables.sh` without going through Step #1. But once Step #1 is initiated, it must be completed before the figures and tables can be generated.

3. `Figure 6`, the last figure in the paper, requires the largest computing power. For each of the thirty turbines, it is recommended to use a 12 core CPU and 36 GB of RAM. Using these resources, the expected runtime for one turbine dataset is 8 to 10 hours. Hence, it is advisable to run all the thirty turbines on different nodes in parallel. The shell script `computeFigure6Results_multinode.sh` has 30 lines of script each corresponding to one turbine. These should be fed into a high performance computing cluster's batch scheduler which would assign different node to each line of the command. Ones the job execution is done, one can generate `Figure 6` using the shell script `generateFigure6.sh` on any computer.