0 Introduction

This is the official documentation of the code repository of the paper: Patrick Killeen, Ci Lin, Futong Li, Iluju Kiringa, and Tet Yeap "Nitrous Oxide Emission Prediction Using IoT Soil and Weather Sensor Data".

It includes N2O emission prediction via machine learning, and deep learning regression models using Python and TensorFlow.

1 Requirements

This is my experiment environment.

- 64-bit Linux (Ubuntu 20.04) desktop with a NVIDIA GeForce RTX 3060 graphics card
- Python 3.8.19
- TensorFlow version 2.12.0
- CUDA version 11.8
- cuDNN version 8.6
- sklearn version 1.3.2
- numpy version 1.24.3
- pandas version 2.0.3

2 Data Pre-processing

We perform some data pre-processing before feeding the datasets to the models. In this section we briefly explain some of the main logic used for data pre-processing, namely, data fusion via lagged features and feature selection via partial least squares regression (PLSR).

2.1 Data Fusion via Lagged Features

We fuse N2O datasets and sensor datasets by deriving lagged features. For every N2O data point after reducing (or leaving as is) the temporal resolution of the N2O dataset, a lagged feature will be derived for each window size. The lag operation applied is decided by the suffix of each feature's name. Features whose name ends with '_avg', '_max', '_min', and '_sum' have the average, maximum, minimum, and summation operations applied, respectively. Any other unknown suffix will just lead to average being used. This data fusion also includes temporal resolution rescaling, so we lower the temporal resolution of the final dataset via averaging.

The Python module involved with this logic is resampling.py, specifically the 'extractLaggedFeatures' function. An example can done by running 'runLaggedFeatureCreation.sh' such that an N2O dataset from 2021 is merged (into 30 min, and 180 min resolution datasets) with the raw sensor dataset of a Pessl sensor from 2021, where the arguments are as follows:

- --n2oDatasetInputPath (string): path to n2o emission dataset. For every sample in this dataset lagged features will be derived from the other input dataset

- -- sensorDatasetInputPath (string): path to sensor data to derive lagged features and fuse to the n2o dataset
- -- outputDatasetPath (string): path where to create output file of fused data
- -- temporalResolution (int): the target temporal resolution in minutes to save the dataset as (should be lower or equal than the n2o dataset)

So an example data fusion can be run as follows:

```
python resampling.py --n2oDatasetInputPath raw-data/licor-n2o-data/2021.csv --
sensorDatasetInputPath raw-data/pessl-soil-weather-data/2021/sensor-node-00209FC8.csv --
outputDatasetPath output/2021-n2o-00209FC8-fused-30min.csv --temporalResolution 30
```

The feature names will have the format ‘<feature name>_<aggregation operation>_Lag<window size>H’, where <feature name> is the name of the original feature used to derived a lagged version of itself, <aggregation operation> is the aggregation operation used to derive the feature, and <window size> is the size of the window used (how far in the past to consider samples in the aggregation operation) to aggregate samples.

2.2 Feature Selection

Once a dataset was pre-processed and contains no missing data, feature selection using PLSR is used. This logic is implemented in the plsr.py module, and can be run using the ‘runFeatureSelection’ function. We explain the API to ‘runFeatureSelection’´below:

- inPath (string): input path to the dataset to apply feature selection to
- outResPath (string): output path to the result file that will contain PLSR model performance and the selected features, where each feature will have a column in the file such that a 0 indicates the feature was not selected and 1 indicates the feature was selected
- outCoeffPath (string): output path to the file that will contain PLSR coefficients. This is useful for feature important analysis (larger coefficients in absolute value have more impact on N2O). A column is found for each feature and has the coefficient of that feature
- year (string): optional argument for logging purposes that will be stored in result file
- n2oChamberName (string): optional argument for logging purposes that will be stored in result file
- resolution (string): optional argument for logging purposes that will be stored in result file
- inDatasetName (string): optional argument for logging purposes that will be stored in result file
- trials (int): number of trials random search is conducted, that is, the number of different hyperparameter sets that are used for hyperparameter tuning when identifying the best hyperparameter choice to use for PLSR
- temporalCVFlag (bool): flag indicating whether block cross-validation is used or not. True means block cross-validation is used and False means random (standard) cross-validation is used.
- nFolds (int): number of folds used by cross-validation
- blockClusterSize (float): (only used when using block cross-validation) the size of the blocks in hours when clustering the dataset before randomly sampling

An example of this feature selection can be run by running ‘runFeatureSelection.sh’ or:

```
python plsr.py --year 2021 --chamber C4 --resolution 180
```

, where the result files will be found in ‘output/feature-selection/’

3 Experiment Description

The are 2 types of experiments, cross-validation experiments that use a single dataset and cross-validation to evaluate models (interpolation), and multi-dataset (multi-year) extrapolation experiments that train a model using one year’s data and test the model using another year’s data. The scripts take as input a CSV configuration file path and an output directory path. The configuration file controls what types of experiments are run (e.g., the learning model). The output directory is the directory where the output result files will be stored, such for each experiment, directories are created in the output directory and are named after the current time.

The main function is the ‘experimenter.py’ script taking 2 input arguments:

- --inFile: (string) the configuration file path to control the experiments
- --outDirectory: (string) the directory where output files will be stored

3.1 Input file formats

The files input to the logic in this project are as follows

- **configuration file**: The configuration file specifies a batch of experiments to run such that each row specifies the type of experiment to run along with various parameters to tune the experiment. Note that there is a memory management issue hidden somewhere in the code, and if the configuration file contains too many experiments (or too many iterations/folds/execution-trials), TensorFlow may run out of memory. A work around is to separated a configuration file into many single entry configuration files and run all the separated configuration files via a shell script with multiple calls to running ‘experimenter.py’ (the main file)
- **Input dataset files**: specified by the configuration file, an input dataset file is in CSV format, where there first column is called 'timestamp' that contains the timestamp of each reading. The dataset can have as many columns/features as desired. The last column is treated as the target variable. The dataset is expected to be ordered by timestamp, so earlier readings are at the start of the file and later readings are at the end. It is also expected that there are no missing readings in the dataset (pre-processing should therefore be first applied). Furthermore, the time difference between consecutive samples can be no smaller than the temporal resolution specified in the configuration file (we discuss this configuration file entry later in the documentation)
- **A selected feature file**: specified by the configuration file, this input CSV file specifies which features are selected to be part of the machine learning experiments. The feature columns in the input dataset file should all be present here. The timestamp and target variable column should not be in this file. Only a single row (excluding the header) should be found in this file, where a 0 in a cell means the feature is not included and a 1 means the feature is included. This design allows for a single dataset file to exist, and many different selected feature files can be used to try out different features without having to change the input dataset.

3.2 Cross-validation

We explain cross-validation experiments in this section.

The cross-validation logic performs nested K-J-fold cross-validation on a single input dataset: the dataset is split into K outer folds, where a fold contains outer-train/test data. Then each outer fold is split again using an inner cross-validation loop into train/validation data splits. Hyperparameters are tuned using the inner-train and validation data, then the model with the best hyperparameters is trained from scratch using the outer-train data and is evaluated against the test data. For each outer fold, J models with different hyperparameters will be evaluated against the test data.

3.2.1 Configuration File Entries

- 'input dataset path' (string): file path to the input dataset file
- 'selected sensors path' (string): file path to the selected feature/sensors file. This specifies which column will be part fed to the machine learning model
- 'year' (string): the year the dataset was from (only used for logging purposes in the result file)
- 'chamber' (string): the gas chamber id (only used for logging purposes in the result file)
- 'temporal resolution (min)' (int): the temporal resolution of the input dataset in minutes
- 'feature selection scheme' (string): the name of the feature selection scheme used to select the features (only used for logging purposes in the result file)
- 'algorithm' (string): machine learning model name. Available options are:
    - ZeroR: a baseline model that ignore input features and simply predicts the average of the target variable from the training data
    - RF: random forest regressor
    - DNN: deep neural network. A multilayer perceptron with multiple hidden layers
    - CNN: 1D convolutional neural networks
    - LSTM: long short-term memory
    - SVM: support vector machine/regressor
    - LR: linear regression
- 'input tensor number of time steps' (int): number of time steps the input samples will have for tensor-based model (e.g., CNN, and LSTM).
- 'apply min-max scaling' (boolean): flag indicating whether to apply min-max scaling (to rescale the data between 0 and 1) or not . Setting this flag to False is useful when the input dataset has already been normalized between 0
- 'seed' (int): the seed used fro random number generation to enable experiment reproducibility
- 'outer CV split type' (string): the type of cross-validation to use for the outer cross-validation. Available options are:
    - random CV: random cross-validation such that samples are randomly sampled when creating folds
    - blockCV: block cross-validation such that samples are first clustered into blocks of a size specified by the 'clustered outer-CV cluster size (hours)' entry, and then the folds are populated by treating the blocks as samples and applying random sampling over the blocks such that all samples from a block are then p folds are then placed into a fold

(note that the samples within the block keep their original order: future work is to apply random sampling the blocks as well). Care must be taken on how large this value is, because too large of a value may result in empty folds being created and crashing the program.

- 'inner CV split type' (string): the type of cross-validation to use for the inner cross-validation, which should be the same as the 'outer CV split type' entry. Available options are:
  - random CV: random cross-validation
  - blockCV: block cross-validation, where the cluster sizes are specified by the 'clustered inner-CV cluster size (hours)' entry.
- 'clustered outer-CV cluster size (hours)' (int): cluster size (in hours) of the blocks when using block cross-validation for the outer cross-validation. This entry is ignored when random cross-validation is used.
- 'clustered inner-CV cluster size (hours)' (int): cluster size (in hours) of the blocks when using block cross-validation for the inner cross-validation. This entry is ignored when random cross-validation is used.
- 'iterations' (int): number of iterations that cross-validation should be repeated. More iterations would mean a better understanding of the average model performance. Should be at least 1
- 'number of outer folds' (int): number of folds used for the outer cross-validation. Must be at least 2
- 'number of inner folds' (int): number of folds used for the inner cross-validation. Must be at least 2
- 'number of trials' (int): the number of sets of hyperparameters that are tested out when tuning hyperparameters for a given outer-inner fold pair. Must be at least 1.
- 'number of executions per trial' (int): the number of times a model is re-trained from scratch using a set of hyperparameters to reduce the effects of unlucky initial weights affecting hyperparameter selection. Must be at least 1.
- ‘output hyperparameter choice’ (Boolean): flag indicating whether the choices of hyperparameters are output for each fold.

3.2.2 Output

The output directory provided to the main function is the root directory of all output files. Each time a configuration file is used to run a batch of experiments, the batch of experiments will have their own output directory named using the yyyy-mm-dd_hh_mm_ss (<year>-<month>-<day>_<hour>_<minute>_<second>) date format naming convention. This subdirectory will have its own result files. A batch of experiments’ output subdirectory will have 3 main output elements, namely, 2 output files and a subdirectory for each experiment within the provided configuration file. These output elements are named:

- **experiments**: the subdirectory that will hold all the prediction output files for each experiment specified by the input configuration file (one for each row of the configuration file)
- **results**.csv: the CSV file that summarize every iteration and fold’s performance metrics for all of the experiments.

- **log**: the log file that output all information and error messages. The amount of stuff being logged can be controlled by changing the 'GLOBAL_LOG_LEVEL' variable in the 'myio' module. Available log levels are:
    - LOG_LEVEL_DEBUG: a lot of debug information is logged. Useful for debugging errors
    - LOG_LEVEL_INFO: (the default) only a small amount of useful information is logged
    - LOG_LEVEL_WARNING: only warning messages are logged
    - LOG_LEVEL_ERROR: only error messages are logged
- **configFile.csv**: a copy of the configuration file used to run the batch of experiments is made in the output directory.

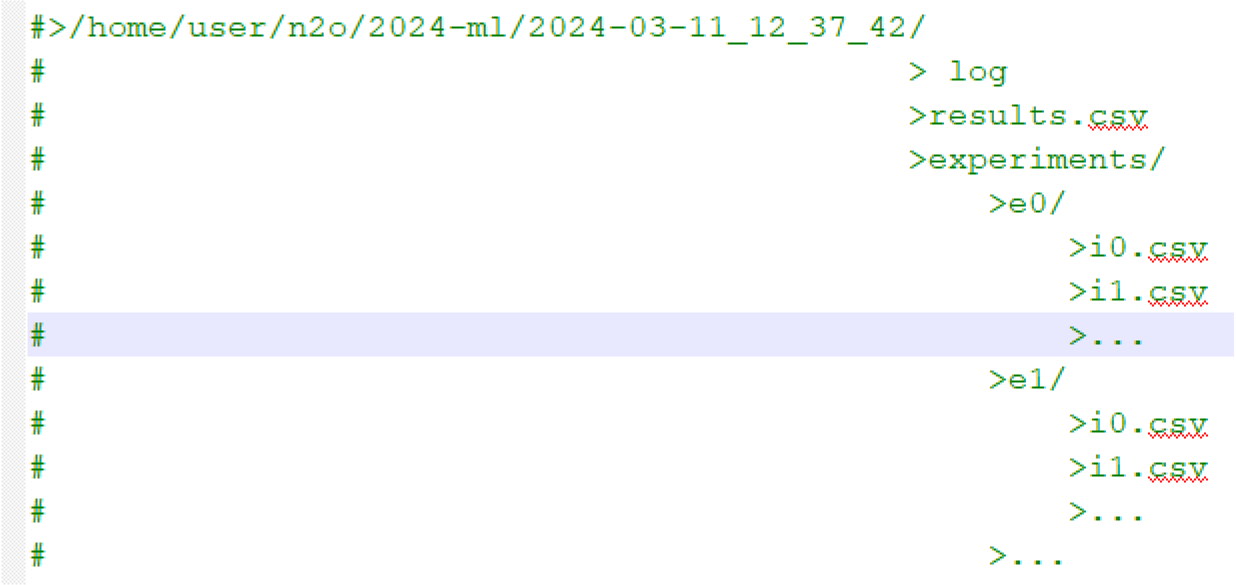
3.2.2.1 The 'experiments' subdirectory

This subdirectory will have subdirectories for each experiments provided in a configuration file. The naming convention of the subdirectories is 'e*X*', where *X* is the experiment id (the row ID, starting from 0, of a configuration file entry). For each of these 'eX' subdirectories, each iteration will produce an output file named 'iY.csv', where Y is the iteration id. The iY.csv prediction files will have a list of the actual N2O value, the timestamp, and the predictions made by each inner fold's model. This way an analyst can keep track of what prediction was made for what sample (useful for time-series plots of emission predictions vs. actual emissions)

The prediction 'iY.csv' files have the following columns:

- **sampleid:** the id of the sample in the input dataset
- **timestamp:** the timestamp of the sample
- **index of sample after CV shuffle:** the index of the sample after cross-validation (CV) was applied and the dataset was shuffled. This way you can sort the output file by this column and you will obtain how the dataset was shuffled
- **outer-fold:** the id of the fold the sample belonged to when it was used for testing in the outer-cross-validation loop
- **actual_N2O:** the actual N2O reading from the original dataset
- **predicted_inner_fold*i*:** the predicted N2O made by the model for inner fold *i*
- **predicted_inner_fold_average:** the average predictions over all inner folds for the sample

An example output file structure would be as follows, if /home/user/n2o/2024-ml was provided as the output directory to the main function and the experiment batch had at least 2 experiments (2 configuration row entries) and the experiments each had at least 2 iterations.



```
#>/home/user/n2o/2024-ml/2024-03-11_12_37_42/
#                                           > log
#                                           >results.csv
#                                           >experiments/
#                                               >e0/
#                                                   >i0.csv
#                                                   >i1.csv
#                                                   >...
#                                               >e1/
#                                                   >i0.csv
#                                                   >i1.csv
#                                                   >...
#                                               >...
```

Furthermore, there is an optional file called 'hyperparameter-choices.csv' (controlled by the 'output hyperparameter choice' configuration file entry) that will contain the hyperparameter choices of the best performing model resulting from hyperparameter tuning, for each iteration, and fold. The 3 first headers are 'iteration', 'outer fold', and 'inner fold', and the remaining headers are for each hyperparameter.

3.2.2.2 The 'results.csv' file

This file contains all the result information of all experiments of a single configuration file, and is useful for creating excel pivot table to analyze average performance over all folds and iterations for all experiments in a batch of experiments specified by a configuration file. This file has the following columns:

- **experiment id:** id of the experiment
- **processing device:** indicates whether the model was trained using a CPU or GPU (deep learning models are expected to be run on GPU, unless the execution environment was poorly configured)
- **algorithm:** the model name
- **year:** the year (provided by the configuration file entry)
- **chamber:** the chamber id (provided by the configuration file entry)
- **temporal resolution (min):** temporal resolution (in minutes) of the input dataset (provided by the configuration file entry)
- **input tensor number of time steps:** number of time steps in the input samples to tensor-based model (provided by the configuration file entry)
- **feature selection scheme:** the scheme used to select the features (provided by the configuration file entry)
- **number of features:** number of features selected (specified by the selected features input file)
- **number of instances:** number of samples in the input dataset
- **iteration:** the current iteration number
- **seed:** the random number generation seed used (provided by the configuration file entry)

- **outer fold: current** fold number in the outer cross-validation loop
- **inner fold: current** fold number in the inner cross-validation loop
- **MSE:** mean squared error performance of the model on the given outer-inner fold pair
- **RMSE:** root mean squared error performance of the model on the given outer-inner fold pair
- **R2:** coefficient of determination performance of the model on the given outer-inner fold pair
- **MAPE:** a version of mean absolute percentage error performance of the model on the given outer-inner fold pair such that any predicted or actual emission that is between -1 and 0 (exclusive) are snapped to -1 and any predicted or actual emission that is between 0 and 1 (inclusive) are snapped to 1 do deal with the common occurs of near-0 emissions making MAPE computations huge due to divisions almost by zero. Furthermore, note that this is in ratio format and would need to be multiplied by 100 to convert it into percentage error units.
- **execution time(s):** the time (in seconds) taken to train and evaluate the model on the given outer-inner fold pair

3.2.3 Running the experiment example

Assuming your working directory is setup as follows:

- experimenter.py (and all other python scripts in the same directory)
- input/configs/config.csv
- output/

Run the below line of code in the command line to run the batch of experiments specified by the ‘input/configs/config.csv’ configuration file, where the results will be output into the ‘output’ directory:

```
python experimenter.py --inFile input/configs/config.csv  --outDirectory output
```

3.3 Specifying a training and testing dataset

We explain holdout experiments (when --trainTestFileSpecific is True when provided to experimenter.py) in this section where a training file is provided and a testing file is provided. This type of experiment trains models on a given training dataset and tests the models on a given test set. Hyperparameter tuning is performed on the training dataset using cross-validation, where for each fold the dataset is split into inner-train and validation datasets. The hyperparameter tuning proceeds in a similar fashion as cross-validation (see Section 3.2).

3.3.1 Configuration File Entries

- 'input train dataset path' (string): file path to the training dataset file
- 'input test dataset path' (string): file path to the test dataset file
- 'selected sensors path' (string): file path to the selected feature/sensors file. This specifies which column will be part fed to the machine learning model.
- ' train dataset year' (string): the year the train dataset was from (only used for logging purposes in the result file)

- 'train dataset chamber' (string): the gas chamber id of the training dataset (only used for logging purposes in the result file)
- ' test dataset year' (string): the year the test dataset was from (only used for logging purposes in the result file)
- ‘test dataset chamber' (string): the gas chamber id of the testing dataset (only used for logging purposes in the result file)
- 'temporal resolution (min)' (int): the temporal resolution of the input dataset in minutes (rows can be missing from the dataset, but timestamps in the dataset should be spaced out by no less than the temporal resolution specified in this entry)
- 'feature selection scheme' (string): the name of the feature selection scheme used to select the features (only used for logging purposes in the result file)
- 'algorithm' (string): machine learning model name. Available options are:
    - ZeroR: a baseline model that ignore input features and simply predicts the average of the target variable from the training data
    - RF: random forest regressor
    - MLP: multilayer perceptron. A neural network with 1 input layer, 1 hidden layer, and 1 output layer.
    - DNN: deep neural network. A multilayer perceptron with multiple hidden layers
    - CNN: 1D convolutional neural networks
    - LSTM: long short-term memory
    - SVM: support vector machine/regressor
    - LR: linear regression
- 'input tensor number of time steps' (int): number of time steps the input samples will have for tensor-based model (e.g., CNN, and LSTM).
- 'apply min-max scaling' (boolean): flag indicating whether to apply min-max scaling (to rescale the data between 0 and 1) or not . Setting this flag to False is useful when the input dataset has already been normalized between 0
- 'seed' (int): the seed used for random number generation to enable experiment reproducibility
- ‘inner CV split type' (string): the type of cross-validation to use for hyperparameter tuning on the training dataset. Available options are:
    - random CV: random cross-validation such that samples are randomly sampled when creating folds
    - blockCV: block cross-validation such that samples are first clustered into blocks of a size specified by the 'clustered outer-CV cluster size (hours)' entry
- 'clustered inner-CV cluster size (hours)' (int): cluster size (in hours) of the blocks when using block cross-validation for the cross-validation used for hyperparameter tuning. This entry is ignored when random cross-validation is used.
- 'iterations' (int): number of iterations that cross-validation should be repeated. More iterations would mean a better understanding of the average model performance. Should be at least 1
- 'number of inner folds' (int): number of folds used for the cross-validation hyperparameter tuning process. Must be at least 2
- 'number of trials' (int): the number of sets of hyperparameters that are tested out when tuning hyperparameters for a given outer-inner fold pair. Must be at least 1.

- 'number of executions per trial' (int): the number of times a model is re-trained from scratch using a set of hyperparameters to reduce the effects of unlucky initial weights affecting hyperparameter selection. Must be at least 1.

3.3.2 Output

The same as the cross-validation experiments (see Section 3.2.2)

3.3.2.1 The 'experiments' subdirectory

The same as the cross-validation experiments (see Section 3.2.2.1)

3.3.2.2 The 'results.csv' file

This file is almost identical to the result file of cross-validation experiments (see Section 3.2.2.2): the differences are as follows:

- The 'year' column is replaced with 'train dataset year' and 'test dataset year', since we have 2 input datasets (provided by the configuration file entry)
- The 'chamber' column is replaced with 'train dataset chamber' and 'test dataset chamber', since we have 2 input datasets (provided by the configuration file entry)
- The 'outer fold' column was removed, since there is no outer cross-validation loop

3.3.3 Running the experiment example

Assuming your working directory is setup as follows:

- experimenter.py (and all other python scripts in the same directory)
- input/configs/config.csv
- output/

Run the below line of code in the command line to run the batch of experiments specified by the 'input/configs/config.csv' configuration file, where the results will be output into the 'output' directory:

```
python experimenter.py --inFile input/configs/config.csv  --outDirectory output --
trainTestFileSpecific True
```

4 Adding a new Machine Learning Model

This section explains what parts of the code need to be changed to create and implement a new model. The model.py module contains all the model logic and needs to be changed. Below is the list of changes that must be made to model.py:

- Add the name/acronym of the new model as a constant variable using the naming convention 'ALG_' as a prefix to the variable name (e.g., 'ALG_RANDOM_FOREST'). The value of this variable will be the acronym of the model that is specified in the 'algorithm' entry of the configuration file.
- Add an entry to the 'deepLearningModelMap' variable to indicate whether the new model is a deep learning model or not using the new algorithm acronym as the key

- Add an entry to the 'modelsWithTensorsMap' variable to indicate if the model has multi-dimensional input in time (tensor-based input samples), like CNN and LSTM, using the new algorithm acronym as the key
- In the constructor of the Model class (the __init__ function), add to the algorithm name if condition structure the new model's name, and call the appropriate function to build the model given the hyperparameters
- Create a new function that takes as input the set of hyperparameters and the function creates the model, initializes it, and returns the new model. The name of the function should following the naming convention 'buildModel_name', where 'Model_name' is the name of the model.
- Modify the 'generateHyperparameterSets' function to add the new model's name to the if structure to define the random search hyperparameter search range of each parameter. The logic is as follows to add a hyperparameter
    - Add an empty list to the 'paramDict' map using the hyperparameter name as the key (e.g., paramDict["units"]=[])
    - Use a for loop to populate the list. The list contains all possible values that random search can sample from. So appending 'i' in the loop 'for i in range(8,256,1):' would mean that i starts at 8, is incremeneted by 1 up until 256.
    - Repeat this for each hyperparameter