# mlstream

***Release 0.1.2***

**Feb 14, 2020**

# Contents:

Machine learning for streamflow prediction.

PyPI: https://pypi.org/project/mlstream GitHub: https://github.com/gauchm/mlstream

CHAPTER 1

# Installation

This project is available on PyPI: https://pypi.org/project/mlstream

```
pip install mlstream
```

If you want to use the `LumpedXGBoost` model from `mlstream.models.xgboost`, you need to separately install the optional dependency `xgboost`.

CHAPTER 2

# Usage

The idea of this package is simple: Just provide your data, select a model (or provide your own), and get the predictions.

## 2.1 File Structure

The `data_root` directory should have the following structure:

```
- gauge_info/
  - gauge_info.csv
- discharge/
  - *.nc
- forcings/
  - lumped/
    - *_<basinId>.{rvt,txt,csv}
```

`gauge_info.csv` is a csv-file with at least a column 'Gauge_ID' and possibly further static basin attributes. If the file further contains a column 'Cal_Val' indicating calibration ('C') and validation ('V') basins, the method `datautils.get_basin_list()` can be used to select calibration or validation basins.

Currently, only NetCDF files for discharge and csv, txt, or rvt files for daily lumped forcings are supported. The NetCDF discharge files have the variables `station_id(nstations)`, `time(time)`, and `Q(nstations, time)`. If multiple NetCDF-files exist, their contents are merged and duplicate stations ignored.

The csv or txt-forcing files have a header row and are expected to be comma-separated. The first column needs to contain the dates as `YYYY-MM-DD`. The rvt-files are simple comma-space-separated csv-files as generated by Raven. An example rvt file looks like this:

```
:MultiData
2000-01-01 00:00:00 1.0 6210
:Parameters,                    PRECIP,                 TEMP_DAILY_AVE,                 ␣
↪  TEMP_MIN,                        TEMP_MAX
:Units,                         mm/d,                           C,                      ␣
↪          C,                               C
```

(continues on next page)

(continued from previous page)

```
                        0.471473290000,                  -2.02099736340,
↪  -6.94564452600,                  3.092910019300,
...
:EndMultiData
```

There are four header lines and one footer line; and the second header line starts with the start date formatted as `YYYY-MM-DD`. The third header line contains the column names as `:Parameters, <col1>, <col2>, ..., <colN>`.

## 2.2 Training

For training, create an `Experiment` with the path to the data folder, specify a run directory to store results, start and end date, the training basins, forcing and static basin attributes, and input sequence length. Further, you can specify if the target to predict can take negative values (`allow_negative_target=True`). E.g., when you predict discharge, this should be `False`, but when you predict the error of another model's discharge prediction, it should be `True` (since the other model can be off in both directions). Then, set the model to use for training, and start training with `exp.train()`:

```
data_path = Path('./data')
run_dir = Path('./experiments')
exp = Experiment(data_path, is_train=True, run_dir=run_dir,
                 start_date='01012000', end_date='31122015',
                 basins=['00AAA123', '00AAB234', '00AAC567'],
                 seq_length=100, concat_static=True,
                 static_attributes=['area', 'regulation'],
                 forcing_attributes=['precip', 'tmax', 'tmin'],
                 allow_negative_target=False)
exp.set_model(model)
exp.train()
```

Any model class that implements the methods `train(self, ds: LumpedH5)` and `predict(self, ds: LumpedBasin)` can be used. Pre-implemented models can be found in `mlstream.models`.

## 2.3 Inference

To run inference after training, create a new `Experiment` with `is_train = False`, provide the data path, the path to the run directory from training, the test basins, and start and end date. There is no need to specify sequence length, forcing and static attributes, or `allow_negative_target` again; instead, these values are loaded from the configuration file in the run directory. Load and set the trained model (which was saved in the run directory during training), and run predictions with `exp.predict()`, which will return a DataFrame of predictions.

```
exp = Experiment(data_path, is_train=False, run_dir=run_dir,
                 basins=['01ABC123', '02DEF123'],
                 start_date='01012016', end_date='31122018')
model.load(run_dir / 'model.pkl')
exp.set_model(model)
results = exp.predict()
```

To obtain NSE scores for each test basin, run `exp.get_nses()`.

CHAPTER 3

# mlstream

## 3.1 mlstream package

### 3.1.1 Subpackages

#### mlstream.models package

#### Submodules

#### mlstream.models.base_models module

**class** mlstream.models.base_models.**LumpedModel**
Bases: object

Model that operates on lumped (daily, basin-averaged) inputs.

**load**(*model_file: pathlib.Path*) → None
Loads a trained and pickled model.

**Parameters** **model_file** (*Path*) – Path to the stored model.

**predict**(*ds: mlstream.datasets.LumpedBasin*) → numpy.ndarray
Generates predictions for a basin.

**Parameters** **ds** (LumpedBasin) – Dataset of the basin to predict.

**Returns** Array of predictions.

**Return type** np.ndarray

**train**(*ds: mlstream.datasets.LumpedH5*) → None
Trains the model.

**Parameters** **ds** (LumpedH5) – Training dataset

### mlstream.models.lstm module

Large parts of this implementation are taken over from https://github.com/kratzert/ealstm_regional_modeling.

**class** mlstream.models.lstm.**EALSTM**(*input_size_dyn: int*, *input_size_stat: int*, *hidden_size: int*, *batch_first: bool = True*, *initial_forget_bias: int = 0*)

Bases: sphinx.ext.autodoc.importer._MockObject

Implementation of the Entity-Aware-LSTM (EA-LSTM)

Model details: https://arxiv.org/abs/1907.08456

**Parameters**

- **input_size_dyn** (*int*) – Number of dynamic features, which are those, passed to the LSTM at each time step.
- **input_size_stat** (*int*) – Number of static features, which are those that are used to modulate the input gate.
- **hidden_size** (*int*) – Number of hidden/memory cells.
- **batch_first** (*bool, optional*) – If True, expects the batch inputs to be of shape [batch, seq, features] otherwise, the shape has to be [seq, batch, features], by default True.
- **initial_forget_bias** (*int, optional*) – Value of the initial forget gate bias, by default 0

**forward**(*x_d: <sphinx.ext.autodoc.importer._MockObject object at 0x7f944ad151d0>*, *x_s: <sphinx.ext.autodoc.importer._MockObject object at 0x7f944ad15208>*) → Tuple[<sphinx.ext.autodoc.importer._MockObject object at 0x7f944ad15240>, <sphinx.ext.autodoc.importer._MockObject object at 0x7f944ad15278>]

Performs a forward pass on the model.

**Parameters**

- **x_d** (*torch.Tensor*) – Tensor, containing a batch of sequences of the dynamic features. Shape has to match the format specified with batch_first.
- **x_s** (*torch.Tensor*) – Tensor, containing a batch of static features.

**Returns**

- **h_n** (*torch.Tensor*) – The hidden states of each time step of each sample in the batch.
- **c_n** (*torch.Tensor*) – The cell states of each time step of each sample in the batch.

**reset_parameters**()

Initialize all learnable parameters of the LSTM

**class** mlstream.models.lstm.**LSTM**(*input_size: int*, *hidden_size: int*, *batch_first: bool = True*, *initial_forget_bias: int = 0*)

Bases: sphinx.ext.autodoc.importer._MockObject

Implementation of the standard LSTM.

**Parameters**

- **input_size** (*int*) – Number of input features
- **hidden_size** (*int*) – Number of hidden/memory cells.
- **batch_first** (*bool, optional*) – If True, expects the batch inputs to be of shape [batch, seq, features] otherwise, the shape has to be [seq, batch, features], by default True.

- **initial_forget_bias** (*int, optional*) – Value of the initial forget gate bias, by default 0

**forward**(*x: <sphinx.ext.autodoc.importer._MockObject object at 0x7f944ad15358>*) → Tuple[<sphinx.ext.autodoc.importer._MockObject object at 0x7f944ad15390>, <sphinx.ext.autodoc.importer._MockObject object at 0x7f944ad153c8>]

Performs a forward pass on the model.

**Parameters** **x** (*torch.Tensor*) – Tensor, containing a batch of input sequences. Format must match the specified format, defined by the batch_first agrument.

**Returns**

- **h_n** (*torch.Tensor*) – The hidden states of each time step of each sample in the batch.
- **c_n** (*torch.Tensor*) – The cell states of each time step of each sample in the batch.

**reset_parameters**()

Initializes all learnable parameters of the LSTM.

**class** mlstream.models.lstm.**LumpedLSTM**(*num_dynamic_vars: int, num_static_vars: int, use_mse: bool = True, no_static: bool = False, concat_static: bool = False, run_dir: pathlib.Path = None, n_jobs: int = 1, hidden_size: int = 256, learning_rate: float = 0.001, learning_rates: Dict[KT, VT] = {}, epochs: int = 30, initial_forget_bias: int = 5, dropout: float = 0.0, batch_size: int = 256, clip_norm: bool = True, clip_value: float = 1.0*)

Bases: *mlstream.models.base_models.LumpedModel*

(EA-)LSTM model for lumped data.

**load**(*model_file: pathlib.Path*) → None

Loads a trained and pickled model.

**Parameters** **model_file** (*Path*) – Path to the stored model.

**predict**(*ds: mlstream.datasets.LumpedBasin*) → numpy.ndarray

Generates predictions for a basin.

**Parameters** **ds** (LumpedBasin) – Dataset of the basin to predict.

**Returns** Array of predictions.

**Return type** np.ndarray

**train**(*ds: mlstream.datasets.LumpedH5*) → None

Trains the model.

**Parameters** **ds** (LumpedH5) – Training dataset

**class** mlstream.models.lstm.**Model**(*input_size_dyn: int, input_size_stat: int, hidden_size: int, initial_forget_bias: int = 5, dropout: float = 0.0, concat_static: bool = False, no_static: bool = False*)

Bases: sphinx.ext.autodoc.importer._MockObject

Wrapper class that connects LSTM/EA-LSTM with fully connceted layer

**forward**(*x_d: <sphinx.ext.autodoc.importer._MockObject object at 0x7f944ad06d30>, x_s: <sphinx.ext.autodoc.importer._MockObject object at 0x7f944ad15048> = None*) → Tuple[<sphinx.ext.autodoc.importer._MockObject object at 0x7f944ad15080>, <sphinx.ext.autodoc.importer._MockObject object at 0x7f944ad150b8>, <sphinx.ext.autodoc.importer._MockObject object at 0x7f944ad150f0>]

Run forward pass through the model. :param x_d: Tensor containing the dynamic input features of shape

[batch, seq_length, n_features] :type x_d: torch.Tensor :param x_s: Tensor containing the static catchment characteristics, by default None :type x_s: torch.Tensor, optional

**Returns**

- **out** (*torch.Tensor*) – Tensor containing the network predictions
- **h_n** (*torch.Tensor*) – Tensor containing the hidden states of each time step
- **c_n** (*torch,Tensor*) – Tensor containing the cell states of each time step

### mlstream.models.nseloss module

**class** mlstream.models.nseloss.**NSELoss**(*eps: float = 0.1*)

Bases: sphinx.ext.autodoc.importer._MockObject

Calculates (batch-wise) NSE Loss.

Each sample i is weighted by 1 / (std_i + eps)^2, where std_i is the standard deviation of the discharge of the basin to which the sample belongs.

**Parameters** **eps** (*float*) – Constant, added to the weight for numerical stability and smoothing, default to 0.1

**forward**(*y_pred: <sphinx.ext.autodoc.importer._MockObject object at 0x7f944ad06f60>, y_true: <sphinx.ext.autodoc.importer._MockObject object at 0x7f944ad06f98>, q_stds: <sphinx.ext.autodoc.importer._MockObject object at 0x7f944ad06fd0>*)

Calculates the batch-wise NSE loss function.

**Parameters**

- **y_pred** (*torch.Tensor*) – Tensor containing the network prediction.
- **y_true** (*torch.Tensor*) – Tensor containing the true discharge values
- **q_stds** (*torch.Tensor*) – Tensor containing the discharge std (calculated over training period) of each sample

**Returns** The batch-wise NSE-Loss

**Return type** torch.Tenor

**class** mlstream.models.nseloss.**XGBNSEObjective**(*dummy_target*, *actual_target*, *q_stds*, *eps: float = 0.1*)

Bases: object

Custom NSE XGBoost objective.

This is a bit of a hack: We use a unique dummy target value for each sample, allowing us to look up the q_std that corresponds to the sample's station. When calculating the loss, we replace the dummy with the actual target so the model learns the right thing.

**neg_nse_metric_sklearn**(*estimator*, *X*, *y_true*)

Negative NSE metric for sklearn.

**nse**(*y_pred*, *y_true*, *q_stds*)

**nse_metric_xgb**(*y_pred*, *y_true*)

NSE metric for XGBoost.

**nse_objective_xgb**(*y_pred*, *dtrain*)

NSE objective for XGBoost (non-sklearn API).

**nse_objective_xgb_sklearn_api**(*y_true*, *y_pred*)
NSE objective for XGBoost (sklearn API).

### mlstream.models.sklearn_models module

**class** mlstream.models.sklearn_models.**LumpedSklearnRegression**(*model: sklearn.base.BaseEstimator*, *no_static: bool = False*, *concat_static: bool = True*, *run_dir: pathlib.Path = None*, *n_jobs: int = 1*)

Bases: *mlstream.models.base_models.LumpedModel*

Wrapper for scikit-learn regression models on lumped data.

**load**(*model_file: pathlib.Path*) → None
Loads a trained and pickled model.

**Parameters** **model_file** (*Path*) – Path to the stored model.

**predict**(*ds: mlstream.datasets.LumpedBasin*) → numpy.ndarray
Generates predictions for a basin.

**Parameters** **ds** (LumpedBasin) – Dataset of the basin to predict.

**Returns** Array of predictions.

**Return type** np.ndarray

**train**(*ds: mlstream.datasets.LumpedH5*) → None
Trains the model.

**Parameters** **ds** (LumpedH5) – Training dataset

### mlstream.models.xgboost module

**class** mlstream.models.xgboost.**LumpedXGBoost**(*no_static: bool = False*, *concat_static: bool = True*, *use_mse: bool = False*, *run_dir: pathlib.Path = None*, *n_jobs: int = 1*, *seed: int = 0*, *n_estimators: int = 100*, *learning_rate: float = 0.01*, *early_stopping_rounds: int = None*, *n_cv: int = 5*, *param_dist: Dict[KT, VT] = None*, *param_search_n_estimators: int = None*, *param_search_n_iter: int = None*, *param_search_early_stopping_rounds: int = None*, *reg_search_param_dist: Dict[KT, VT] = None*, *reg_search_n_iter: int = None*, *model_path: pathlib.Path = None*)

Bases: *mlstream.models.base_models.LumpedModel*

Wrapper for XGBoost model on lumped data.

**load**(*model_path: pathlib.Path*)
Loads a trained and pickled model.

**Parameters** **model_file** (*Path*) – Path to the stored model.

**predict** (*ds: mlstream.datasets.LumpedBasin*) → numpy.ndarray
Generates predictions for a basin.

**Parameters** **ds** (LumpedBasin) – Dataset of the basin to predict.

**Returns** Array of predictions.

**Return type** np.ndarray

**train** (*ds: mlstream.datasets.LumpedH5*) → None
Trains the model.

**Parameters** **ds** (LumpedH5) – Training dataset

#### Module contents

### 3.1.2 Submodules

### 3.1.3 mlstream.datasets module

**class** mlstream.datasets.**LumpedBasin**(*data_root: pathlib.Path, basin: str, forcing_vars: List[T], dates: List[T], is_train: bool, train_basins: List[T], seq_length: int, with_attributes: bool = False, concat_static: bool = True, db_path: str = None, allow_negative_target: bool = False, scalers: Tuple[mlstream.scaling.InputScaler, mlstream.scaling.OutputScaler, Dict[str, mlstream.scaling.StaticAttributeScaler]] = None*)
Bases: sphinx.ext.autodoc.importer._MockObject

PyTorch data set to work with the raw text files for lumped (daily basin-aggregated) forcings and streamflow.

**Parameters**

- **data_root** (*Path*) – Path to the main directory of the data set
- **basin** (*str*) – Gauge-id of the basin
- **forcing_vars** (*List*) – Names of forcing variables to use
- **dates** (*List*) – Start and end date of the period.
- **is_train** (*bool*) – If True, discharge observations are normalized and invalid discharge samples are removed
- **train_basins** (*List*) – List of basins used in the training of the experiment this Dataset is part of. Needed to create the correct feature scalers (the ones that are calculated on these basins)
- **seq_length** (*int*) – Length of the input sequence
- **with_attributes** (*bool, optional*) – If True, loads and returns addtionaly attributes, by default False
- **concat_static** (*bool, optional*) – If true, adds catchment characteristics at each time step to the meteorological forcing input data, by default True
- **db_path** (*str, optional*) – Path to sqlite3 database file containing the catchment characteristics, by default None

- **allow_negative_target** (*bool, optional*) – If False, will remove samples with negative target value from the dataset.
- **scalers** (*Tuple[InputScaler, OutputScaler, Dict[str, StaticAttributeScaler]], optional*) – Scalers to normalize and resale input, output, and static variables. If not provided, the scalers will be initialized at runtime, which will result in poor performance if many datasets are created. Instead, it makes sense to re-use the scalers across datasets.

**class** mlstream.datasets.**LumpedH5**(*h5_file: pathlib.Path, basins: List[T], db_path: str, concat_static: bool = True, cache: bool = False, no_static: bool = False*)

Bases: sphinx.ext.autodoc.importer._MockObject

PyTorch data set to work with pre-packed hdf5 data base files. Should be used only in combination with the files processed from *create_h5_files* in the *utils* module.

**Parameters**

- **h5_file** (*Path*) – Path to hdf5 file, containing the bundled data
- **basins** (*List*) – List containing the basin ids
- **db_path** (*str*) – Path to sqlite3 database file, containing the catchment characteristics
- **concat_static** (*bool*) – If true, adds catchment characteristics at each time step to the meteorological forcing input data, by default True
- **cache** (*bool, optional*) – If True, loads the entire data into memory, by default False
- **no_static** (*bool, optional*) – If True, no catchment attributes are added to the inputs, by default False

### 3.1.4 mlstream.datautils module

mlstream.datautils.**get_basin_list**(*data_root: pathlib.Path*, *basin_type: str*) → List[T]

Returns the list of basin names.

If basin_type is 'C' or 'V', the gauge_info.csv needs to contain a column 'Cal_Val' that indicates the basin type.

**Parameters**

- **data_root** (*Path*) – Path to base data directory, which contains a folder 'gauge_info' with the gauge_info.csv file
- **basin_type** (*str*) – 'C' to return calibration stations only, 'V' to return validation stations only, '*' to return all stations

**Returns** List of basin name strings

**Return type** list

mlstream.datautils.**load_discharge**(*data_root: pathlib.Path*, *basins: List[T] = None*) → pandas.core.frame.DataFrame

Loads observed discharge for (calibration) gauging stations.

**Parameters**

- **data_root** (*Path*) – Path to base data directory, which contains a directory 'discharge' with one or more nc-files.
- **basins** (*List, optional*) – List of basins for which to return data. If None (default), all basins are returned

**Returns** A DataFrame with columns [date, basin, qobs], where 'qobs' contains the streamflow.

**Return type** pd.DataFrame

mlstream.datautils.**load_forcings_lumped**(*data_root: pathlib.Path*, *basins: List[T] = None*) → Dict[KT, VT]

Loads basin-lumped forcings.

**Parameters**

- **data_root** (*Path*) – Path to base data directory, which contains the directory 'forcings/lumped/', which contains one .rvt/.csv/.txt -file per basin.
- **basins** (*List, optional*) – List of basins for which to return data. Default (None) returns data for all basins.

**Returns** Dictionary of forcings (pd.DataFrame) per basin

**Return type** dict

mlstream.datautils.**load_static_attributes**(*db_path: pathlib.Path, basins: List[T], drop_lat_lon: bool = True, keep_features: List[T] = None*) → pandas.core.frame.DataFrame

Loads attributes from database file into DataFrame and one-hot-encodes non-numerical features.

**Parameters**

- **db_path** (*Path*) – Path to sqlite3 database file
- **basins** (*List*) – List containing the basin id
- **drop_lat_lon** (*bool*) – If True, drops latitude and longitude column from final data frame, by default True
- **keep_features** (*List*) – If a list is passed, a pd.DataFrame containing these features will be returned. By default, returns a pd.DataFrame containing the features used for training.

**Returns** Attributes in a pandas DataFrame. Index is basin id.

**Return type** pd.DataFrame

mlstream.datautils.**reshape_data**

Reshape data into LSTM many-to-one input samples

**Parameters**

- **x** (*np.ndarray*) – Input features of shape [num_samples, num_features]
- **y** (*np.ndarray*) – Output feature of shape [num_samples, 1]
- **seq_length** (*int*) – Length of the requested input sequences.

**Returns**

- **x_new** (*np.ndarray*) – Reshaped input features of shape [num_samples*, seq_length, num_features], where num_samples* is equal to num_samples - seq_length + 1, due to the need of a warm start at the beginning
- **y_new** (*np.ndarray*) – The target value for each sample in x_new

mlstream.datautils.**store_static_attributes**(*data_root: pathlib.Path*, *db_path: pathlib.Path = None*, *attribute_names: List[T] = None*)

Loads catchment characteristics from text file and stores them in a sqlite3 table

**Parameters**

- **data_root** (*Path*) – Path to the main directory of the data set
- **db_path** (*Path, optional*) – Path to where the database file should be saved. If None, stores the database in data_root/static_attributes.db. Default: None
- **attribute_names** (*List, optional*) – List of attribute names to use. Default: use all attributes.

**Raises** RuntimeError – If attributes folder can not be found.

### 3.1.5 mlstream.experiment module

**class** mlstream.experiment.**Experiment**(*data_root: pathlib.Path*, *is_train: bool*, *run_dir: pathlib.Path*, *start_date: str = None*, *end_date: str = None*, *basins: List[T] = None*, *forcing_attributes: List[T] = None*, *static_attributes: List[T] = None*, *seq_length: int = 10*, *concat_static: bool = False*, *no_static: bool = False*, *cache_data: bool = False*, *n_jobs: int = 1*, *seed: int = 0*, *allow_negative_target: bool = False*, *run_metadata: Dict[KT, VT] = {}*)

Bases: object

Main entrypoint for training and prediction.

**get_nses**() → Dict[KT, VT]
Calculates the experiment's NSE for each calibration basin.

Validation basins are ignored since they don't provide ground truth.

**Returns** **nses** – Dictionary mapping basin ids to their NSE

**Return type** Dict

**Raises** AttributeError – If called before predicting

**predict**() → Dict[KT, VT]
Generates predictions with a trained model.

**Returns** **results** – Dictionary containing the DataFrame of predictions and observations for each basin.

**Return type** Dict

**predict_basin**(*ds: mlstream.datasets.LumpedBasin*, *allow_negative_target: bool = False*) → Tuple[numpy.ndarray, numpy.ndarray]
Predicts a single basin.

**Parameters**

- **ds** (LumpedBasin) – Dataset for the basin to predict
- **allow_negative_target** (*bool, default False*) – If False, will clip predictions to values >= 0

**Returns**

- **preds** (*np.ndarray*) – Array containing the (rescaled) network prediction for the entire data period
- **obs** (*np.ndarray*) – Array containing the observed discharge for the entire data period

**set_model**(*model*) → None
Set the model to use in the experiment.

**train**() → None
Train model.

### 3.1.6 mlstream.scaling module

**class** mlstream.scaling.**InputScaler**(*data_root: pathlib.Path, basins: List[T], start_date: pandas._libs.tslibs.timestamps.Timestamp, end_date: pandas._libs.tslibs.timestamps.Timestamp, forcing_vars: List[T] = None*)
Bases: *mlstream.scaling.Scaler*

**class** mlstream.scaling.**OutputScaler**(*data_root: pathlib.Path, basins: List[T], start_date: pandas._libs.tslibs.timestamps.Timestamp, end_date: pandas._libs.tslibs.timestamps.Timestamp*)
Bases: *mlstream.scaling.Scaler*

**class** mlstream.scaling.**Scaler**
Bases: object

**normalize**(*feature: numpy.ndarray*) → numpy.ndarray

**rescale**(*feature: numpy.ndarray*) → numpy.ndarray

**class** mlstream.scaling.**StaticAttributeScaler**(*db_path: pathlib.Path, basins: List[T], variable_name: str*)
Bases: *mlstream.scaling.Scaler*

### 3.1.7 mlstream.utils module

mlstream.utils.**create_h5_files**(*data_root: pathlib.Path, out_file: pathlib.Path, basins: List[T], dates: List[T], forcing_vars: List[T], seq_length: int, allow_negative_target: bool*)
Creates H5 training set.

**Parameters**

- **data_root** (*Path*) – Path to the main directory of the data set
- **out_file** (*Path*) – Path of the location where the hdf5 file should be stored
- **basins** (*List*) – List containing the gauge ids
- **dates** (*List*) – List of start and end date of the discharge period to use, when combining the data.
- **forcing_vars** (*List*) – Names of forcing variables
- **seq_length** (*int*) – Length of the requested input sequences
- **allow_negative_target** (*bool, optional*) – If False, will remove samples with negative target value from the dataset.

**Raises** FileExistsError – If file at this location already exists.

mlstream.utils.**nse**(*qsim: numpy.ndarray*, *qobs: numpy.ndarray*) → float
Calculates NSE, ignoring NANs in qobs.

$$\text{NSE} = 1 - \frac{\sum_{t=1}^{T} (q_s^t - q_o^t)^2}{\sum_{t=1}^{T} (q_o^t - \bar{q}_o)^2}$$

**Parameters**

- **qsim** (*np.ndarray*) – Predicted streamflow
- **qobs** (*np.ndarray*) – Ground truth streamflow

**Returns** **nse** – The prediction's NSE

**Return type** float

**Raises** ValueError – If lenghts of qsim and qobs are not equal.

mlstream.utils.**prepare_data**(*cfg: Dict[KT, VT], basins: List[T]*) → Dict[KT, VT]
Pre-processes training data.

**Parameters**

- **cfg** (*Dict*) – Dictionary containing the run config
- **basins** (*List*) – List containing the gauge ids

**Returns** Dictionary containing the updated run config.

**Return type** Dict

mlstream.utils.**setup_run**(*cfg: Dict[KT, VT]*) → Dict[KT, VT]
Creates the folder structure for the experiment.

**Parameters** **cfg** (*Dict*) – Dictionary containing the run config

**Returns** Dictionary containing the updated run config

**Return type** Dict

mlstream.utils.**store_results**(*user_cfg: Dict[KT, VT], run_cfg: Dict[KT, VT], results: Dict[KT, VT]*)
Stores prediction results in a pickle file.

**Parameters**

- **user_cfg** (*Dict*) – Dictionary containing the user entered evaluation config
- **run_cfg** (*Dict*) – Dictionary containing the run config loaded from the cfg.json file
- **results** (*Dict*) – DataFrame containing the observed and predicted discharge.

### 3.1.8 Module contents

CHAPTER 4

# Indices and tables

- genindex
- modindex
- search

# Python Module Index

## m

# Index