Forecaster Class Documentation

timeseriesFC.py

Initialization

This python file contains the Class Forecaster, which is the top layer forecasting module. It wraps the Skilearn machine learning methods in order to work with the Class "Dataset" (from utils datawrapper)

The Forecaster Class can be initialized through defining the following Arguments:

- *modeltype*: instance of a Skilearn machine learning class.
  Example: modeltype=RandomForestRegressor(n_estimators=100)

- *datasets*: a list of Dataset instances, where the first one (index=0) is the targetdataset, the others are features.

- *leadtimes*: a list of lead- resp. tailtimes in the same order as the datasets. A tailtime is defined as 2-element list: [start,end]. The unit of start&end is the timestep of data in the corresponding Dataset (decadal Dataset: 1 decade). The value can be negative (feature) or positive (target). "0" is supposed to be the time when the forecast is made → issue date.
  Example: feature tailtime: [-3,-1] , target leadtime: [1] → If the forecast is done today, the value of the next decade is forecasted, based on values of the three decades before the current one.

Optional parameters are:

- *mode*:
  - 'default': forecasts with trained model.
  - 'naiveaverage': uses the historical average for the specific time of the year as forecast
  - 'naivelast': uses the value of the timestep before as forecast. Works only with one step ahead model setups resp. with a targetset leadtime = [1,1].

- *fillnan*: float between 0 and 1. Defines if missing values in a featureset will be interpolated (linearly) and to what fraction of missing values this is true. If set to 0, no interpolation will be done. If set to 0.1, a featureset with less then 10% of missing data will be filled by interpolation but a featureset with 10% or above of missing date will be discarded. The default is 0.

- *scoremethod*: Defines a standard scoring method which will be used by the function crossvalidation and evaluate. Can be R2 (default), soviet_longterm, soviet_shortterm. This setting can be manually overwritten when calling crossvalidate resp. evaluate.

Functions

The Forecaster has the following function. All other functions are supposed to be only used by the class itself.

- **Forecaster.crossvalidate(daterange,scoremethod,folds)**

  It returns the crossvalidated score of the forecasting model over the specified daterange.

  daterange: a list of dates in the format of the targetset Dataset.

  scoremethod is one of the following: R2,soviet_longterm,soviet_shortterm, default='R2'

  folds: number of cross validation folds, default=10

- **Forecaster.train_model(self, training_daterange)**

  trains the model over the given list of dates. If nor argument is given, all available data are used for training.

  daterange: a list of dates in the format of the targetset Dataset.

- **Forecaster.forecast(date)**

  gives the forecast of the trained model for the given date.

  date: a date in the format of the targetset Dataset. Corresponds to "0" in the lead- resp. tailtime definition.

- **Forecaster.evaluate(daterange, output)**

  returns the Score of the trained model over the specified daterange.

  daterange: a list of dates in the format of the targetset Dataset.

  output is one of the following: R2,soviet_longterm,soviet_shortterm, default='R2'

- **Forecaster.plot(daterange, output, filename)**

  Plots several informative graphs of the forecasting results.

  daterange: a list of dates in the format of the targetset Dataset.

  filename: path to the imagefile to be created, default='evaluate.png'

  output: can be timeseries, correlation, soviet_longterm, soviet_shortterm, importance

  timeseries*:* simple value plot of the forecasted, observed and average values of the target Dataset

  correlation: a scatter plot of forecasted and observed values

  soviet_longterm: the soviet_longterm scoring results plotted for every timestep of a year

  soviet_shortterm: the same as above but with the scoring method for shortterm forecasts

  importance: plots the importance of each feature dataset over its tailtime

utils/datawrapper.py

contains the class Dataset, which can handle timeseries datasets of different time resolutions. The current implementation can only handle decadal datsets. A Dataset instance is initialized with a datatype string and a database object. The latter is an instance of the database_loader class in utils/tools.py, which is a helper function, but has no real functionality at the moment. The database objects wraps the path to the folder which contains the decadal time series stored as .csv files. The datatype string is the name of the corresponding csv file.

Example for loading /home/jules/sample_database/chatkal/runoff.csv

```
# Select database (catchment)

database = DatabaseLoader('/home/jules/sample_database/chatkal')

# Create Dataset Instance

runoffdataset = Dataset('runoff', database)
```