Probabilistic Forecasting with `nnetsauce` (using Density Estimation, Bayesian inference, Conformal prediction and Vine copulas)

T. Moudiki (thierrymoudiki.github.io)

2024-07-26

Context

- **Quasi-randomized *neural* networks** (QRNs) applied to time series lags for forecasting
- **Uncertainty quantification** using Kernel Density Estimation, Bayesian inference, Conformal prediction and Vine copulas
- Implemented in Python package `nnetsauce` **version** `0.23.0`

Plan

- 1 - Key components of `nnetsauce` forecasting
  - 1 - 1 Quasi-randomized *neural* networks (QRNs)
  - 1 - 2 Uncertainty quantification in forecasting
- 2 - QRN forecasting with `nnetsauce`
  - 2 - 1 `nnetsauce`'s description (Python version)
  - 2 - 2 Install+import Python packages (including `nnetsauce`)
  - 2 - 3 Import data for the demo
  - 2 - 4 Using the `fit` + `predict` interface
  - 2 - 5 Using GPUs
  - 2 - 6 Time series cross-validation
  - 2 - 7 AutoML with `LazyMTS`

Plan

- **1 - Key components of `nnetsauce` forecasting**
  - 1 - 1 Quasi-randomized *neural* networks (QRNs)
  - 1 - 2 Uncertainty quantification in forecasting
- 2 - QRN forecasting with `nnetsauce`
  - 2 - 1 `nnetsauce`'s description (Python version)
  - 2 - 2 Install+import Python packages (including `nnetsauce`)
  - 2 - 3 Import data for the demo
  - 2 - 4 Using the `fit` + `predict` interface
  - 2 - 5 Using GPUs
  - 2 - 6 Time series cross-validation
  - 2 - 7 AutoML with `LazyMTS`

1 - Key components of `nnetsauce` forecasting

1 - 1 Quasi-randomized *neural* networks (QRNs)

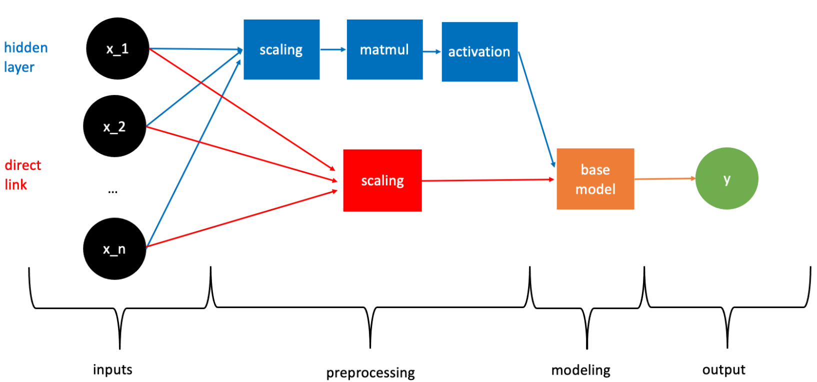


Figure 1: QRN principle

- See also this doc
- A simple case: **base learner** = Linear Regression (next page)

1 - 1 - Quasi-randomized *neural* networks (QRNs)

Simple case: **base learner** = Linear Regression; $y \in \mathbb{R}^n$, to be explained by $X^{(j)}, j \in \{1, \ldots, p\}$

$$y = \beta_0 + \sum_{j=1}^{p} \beta_j X^{(j)} + \sum_{l=1}^{L} \gamma_l g \left(\sum_{j=1}^{p} W^{(j,l)} X^{(j)} \right) + \epsilon$$

With:

- g : **activation function** $\rightarrow$ nonlinearity
- L : number of nodes in the **hidden layer**
- $W^{(j,l)}$, hidden layer: **pseudo/quasi-random**
- **Quasi-random**: designed to cover the space *parsimoniously*
- ***secret* sauce**: "*Layer normalization*" (centering and scaling twice)
- β_j and γ_l : linear model coefficients
- ϵ : residuals

1 - 1 - Quasi-randomized *neural* networks (QRNs)

QRNs applied to time series

- **Response** $y =$ most recent time series observations
- **Covariates** $X =$ time series lags
- **base learner**: can be any Machine Learning model
- **Multivariate forecasting case**: the base learner **shared** by all the time series

1 - 2 Uncertainty quantification in forecasting

Point forecasts/Uncertainty quantification

- **Point forecasts**: cool, but not very informative. How wrong can we be, based on the assumptions that we made: **answers** how ***"certain"*** can we be about the forecast?
- **Uncertainty quantification** needed: **prediction intervals** and/or **predictive simulations**.
- **prediction intervals**: point forecast +/- a term (with a level of confidence)
- **predictive simulations**: future scenarios for the variables of interest

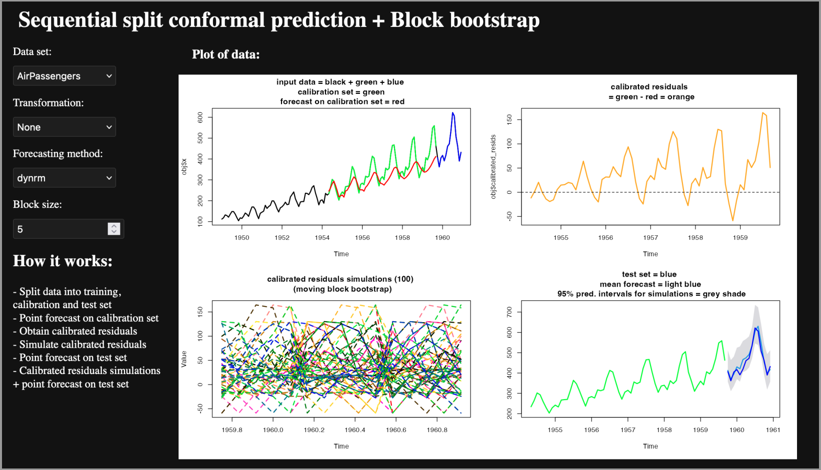
1 - 2 Uncertainty quantification in forecasting

In `nnetsauce`

- Based on:
  - **Bayesian priors**
  - **In-sample residuals** = model fit - true observation on the *whole training set*
  - **Calibrated residuals** = model fit - true observation on a *held-out calibration set*
  - **(Vine) Copulas** (since `nnetsauce v0.23.0`)
- **Calibrated residuals** used in `nnetsauce` for methods based on **sequential split conformal prediction** (more on this later)

1 - 2 Uncertainty quantification in forecasting

Short focus on **sequential split conformal prediction** (see also https://github.com/thierrymoudiki/2024-07-17-scp-block-bootstrap)



1 - 2 Uncertainty quantification in forecasting

Short focus on *copulas* (source vine-copula.org)

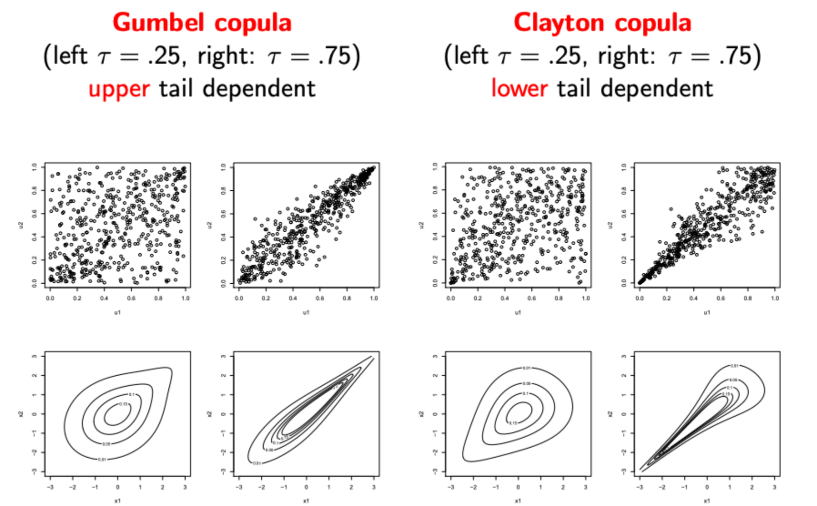


Figure 2: Copulas

1 - 2 Uncertainty quantification in forecasting

Recap

- In `nnetsauce` version `0.23.0`:
  - *Via* a **Bayesian** base learner
  - *Via* a **conformalized** base learner
  - *Via* **in-sample** residuals for methods based on:
    - parametric residuals distribution inference (`gaussian`)
    - density estimation and simulation of residuals (`kde`)
    - bootstrap resampling (`bootstrap` and `block-bootstrap`)
    - vine copulas (`vine-*`)
  - *Via* **calibrated** residuals for methods based on **sequential split conformal prediction (SCP)** (`scp*-kde`, `scp*-bootstrap`, `scp*-block-bootstrap`, `scp*-vine-*`)

Plan

- 1 - Key components of `nnetsauce` forecasting
  - 1 - 1 Quasi-randomized *neural* networks (QRNs)
  - 1 - 2 Uncertainty quantification in forecasting
- **2 - QRN forecasting with `nnetsauce`**
  - 2 - 1 `nnetsauce`'s description (Python version)
  - 2 - 2 Install+import Python packages (including `nnetsauce`)
  - 2 - 3 Import data for the demo
  - 2 - 4 Using the `fit` + `predict` interface
  - 2 - 5 Using GPUs
  - 2 - 6 Time series cross-validation
  - 2 - 7 AutoML with `LazyMTS`

2 - QRN forecasting with `nnetsauce`

2 - 1 `nnetsauce`'s description (Python version)

- General-purpose Machine Learning using **Randomized** and **Quasi-Randomized *neural* networks**
  - GitHub: https://github.com/Techtonique/nnetsauce
  - PyPI: https://pypi.org/project/nnetsauce/
  - Conda: https://anaconda.org/conda-forge/nnetsauce
- **Tasks:**
  - Classification
  - Regression
  - **Univariate/Multivariate time series forecasting**

2 - 1 `nnetsauce`'s description (Python version) (cont'd)

- **Simple interface** for each model:
  - `fit`: fitting model to training data
  - `predict`: model inference on unseen data
- **GPU** version optimizes matrices multiplication using **JAX** (not *magical*)
- Classes `MTS` and `DeepMTS` for **time series forecasting**
- `DeepMTS` seems to be more suited for *nearly* stationary data (but I encourage you to try and tell me)
- Automated Machine Learning (**AutoML**) with classes `LazyMTS` and `LazyDeepMTS`
- **Cross-validation**

2 - 2 Install+import Python packages (including nnetsauce)

```
pip install nnetsauce
```

```
pip install git+https://github.com/Techtonique/mlsauce.git
```

```
import nnetsauce as ns # import the package
import mlsauce as ms
import numpy as np
import pandas as pd
import seaborn as sns

from sklearn.linear_model import Ridge
from statsmodels.tsa.seasonal import STL

sns.set_theme(style="darkgrid")
```

2 - 3 Import data for the demo

Input format (univariate time series)

date	value
1949-01-01	112.0
1949-02-01	118.0
1949-03-01	132.0
1949-04-01	129.0
1949-05-01	121.0
1949-06-01	135.0

2 - 3 Import data for the demo

Input format (multivariate time series)

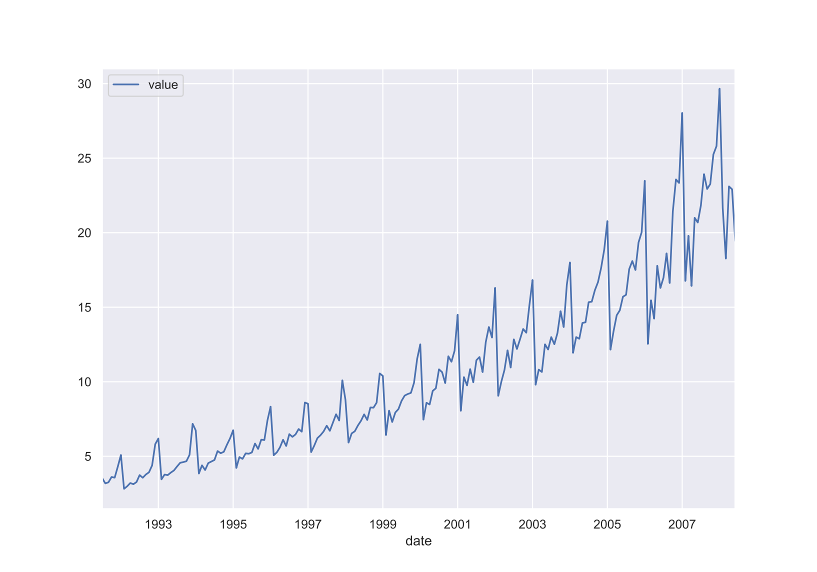
date	Consumption	Income
1970-01-01	0.615986218	0.972261043
1970-04-01	0.46037569	1.169084717
1970-07-01	0.876791423	1.55327055
1970-10-01	-0.274245141	-0.255272381
1971-01-01	1.897370758	1.987153628
1971-04-01	0.911992909	1.447334175

2 - 3 Import data for the demo

Univariate: **Monthly anti-diabetic drug sales in Australia from 1992 to 2008**

```
url = "https://raw.githubusercontent.com/Techtonique/"
url += "datasets/main/time_series/univariate/"
url += "a10.csv"
df_a10 = pd.read_csv(url)
df_a10.index = pd.DatetimeIndex(df_a10.date) # must have
df_a10.drop(columns=['date'], inplace=True)
```

```
df_a10.plot()
```

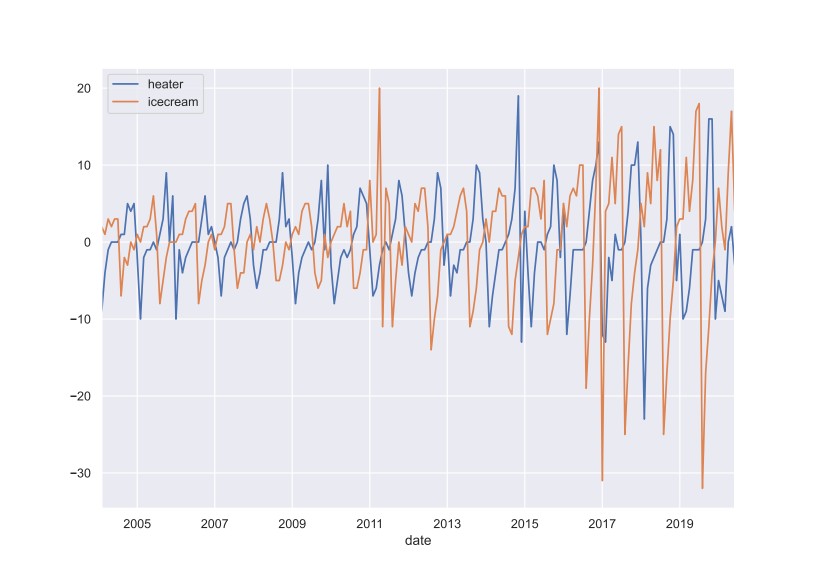

value
30
25
20
15
10
5
1993
1995
1997
1999
2001
2003
2005
2007
date

2 - 3 Import data for the demo (cont'd)

Multivariate: **Heater vs Ice cream sales** data set

```
url = "https://raw.githubusercontent.com/Techtonique/"
url += "datasets/main/time_series/multivariate/"
url += "ice_cream_vs_heater.csv"
df_temp = pd.read_csv(url)
df_temp.index = pd.DatetimeIndex(df_temp.date) # must have
# first other difference
df_icecream = df_temp.drop(columns=['date']).diff().\
dropna()
```

```
df_icecream.plot()
```



2 - 4 Using the `fit` + `predict` interface

A few examples of probabilistic forecasting with `nnetsauce`:

- Gaussian
- Bayesian (Gaussian prior on base learner)
- Kernel Density Estimation (KDE) and sequential split conformal prediction (SCP)
- Conformalized base learner: `TweedieRegressor` + SCP
- Vine Copula (combined with SCP)

Also use the **docs for exact spec.**:
https://techtonique.github.io/nnetsauce/nnetsauce.html#MTS

2 - 4 Using the `fit` + `predict` interface

- **Gaussian**

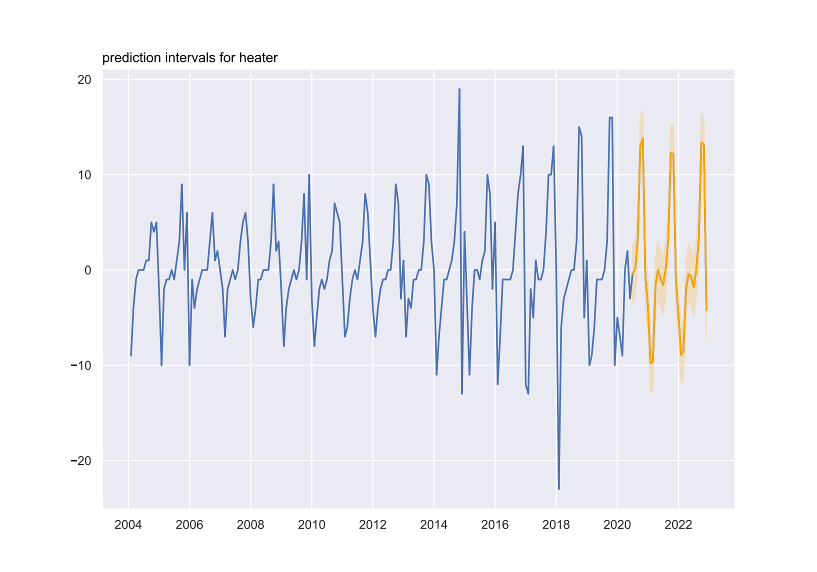
```
from sklearn.ensemble import BaggingRegressor

regr = ns.MTS(obj=BaggingRegressor(), # base learner
              type_pi="gaussian", # type of pred. int.
              lags=20, # number of time series lags
              show_progress=False)
regr.fit(df_icecream); # fit the model
```

```
regr.predict(h=30); # 30-steps ahead forecast
```

2 - 4 Using the `fit` + `predict` interface

```
regr.plot("heater", type_plot="pi") # plot pred. int.
```



2 - 4 Using the `fit` + `predict` interface

- **Bayesian (Gaussian prior)**

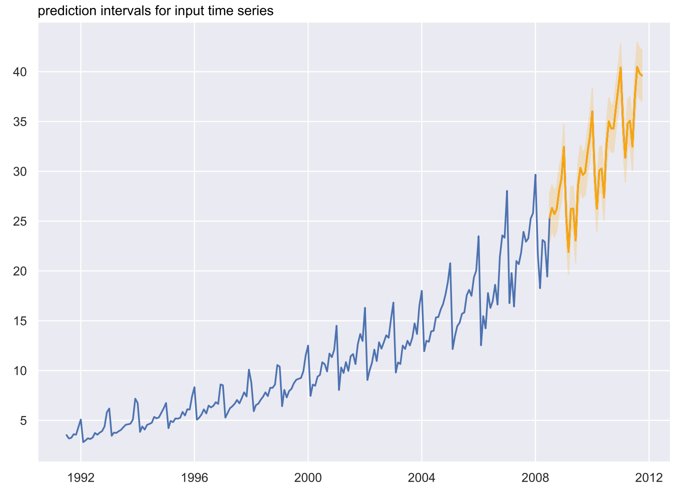
```
from sklearn.linear_model import BayesianRidge

regr = ns.MTS(obj=BayesianRidge(), # base learner
              lags=15, # no. of time series lags
              show_progress=False)
regr.fit(df_a10); # fit the model
```

```
regr.predict(h=40, return_std=True); # 40-steps ahead
```

2 - 4 Using the `fit` + `predict` interface

```
regr.plot(type_plot="pi")
```



2 - 4 Using the `fit` + `predict` interface

- **SCP-KDE**

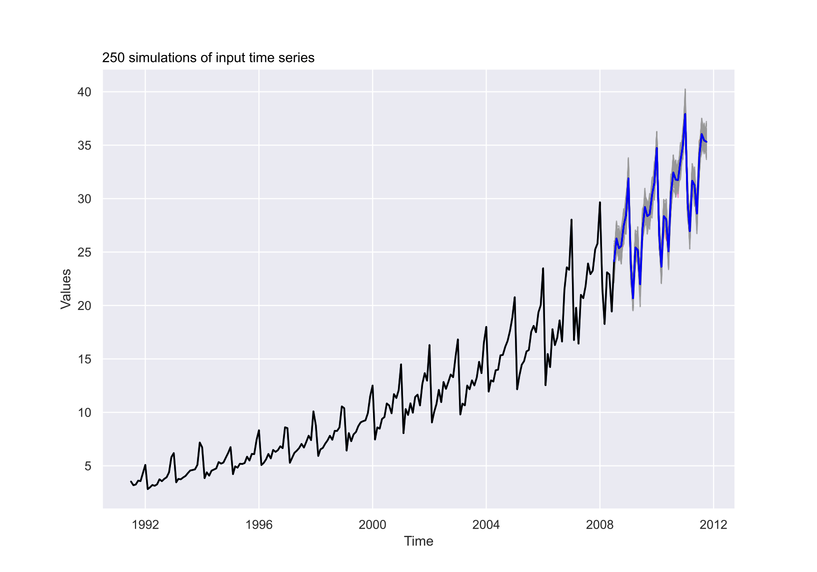
```python
from sklearn.linear_model import LassoLarsIC

regr = ns.MTS(obj=LassoLarsIC(), # base learner
              type_pi="scp-kde", # type of pred. int.
              replications=250, # no. of sample paths
              kernel='gaussian', # density kernel
              lags=15, # no. of time series lags
              show_progress=False)
regr.fit(df_a10); # fit the model
```

```python
regr.predict(h=40); # 40-steps ahead
```

2 - 4 Using the `fit` + `predict` interface

```
regr.plot(type_plot="spaghetti")
```



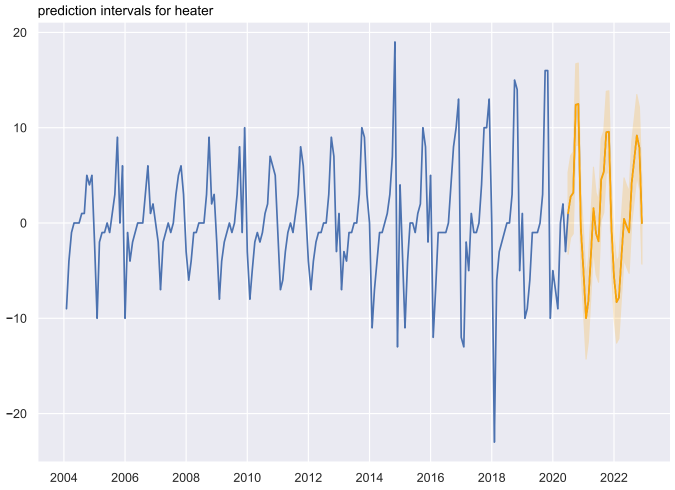
2 - 4 Using the `fit` + `predict` interface

- **Conformalized base learner**

```
from sklearn.linear_model import TweedieRegressor

obj0 = ns.PredictionInterval(obj=TweedieRegressor(),
                             method="splitconformal",
                             type_split="sequential",
                             level=95)
regr = ns.MTS(obj=obj0,
              lags=20,
              show_progress=False)
regr.fit(df_icecream);
regr.predict(h=30, return_pi=True);
```

```python
regr.plot("heater", type_plot="pi") # plot one series
```



2 - 4 Using the `fit` + `predict` interface

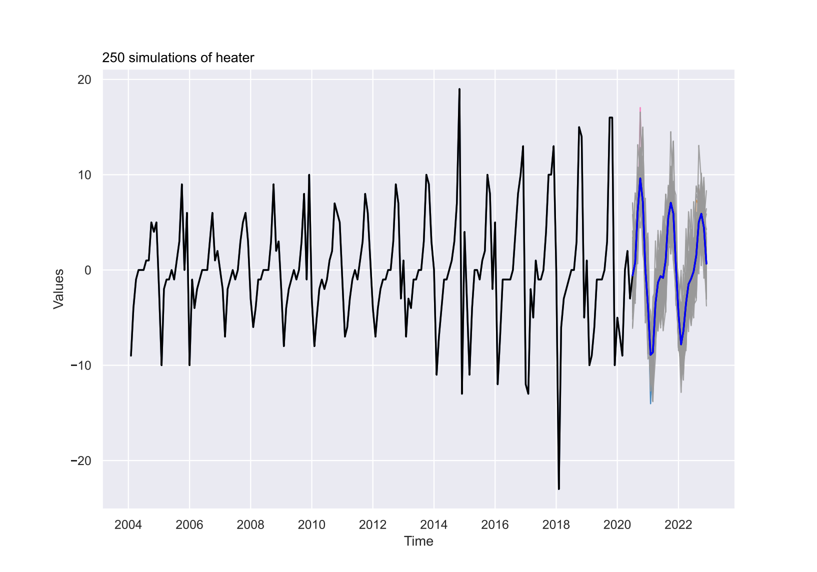
- **Vine copula + sequential split conformal**

```python
import nnetsauce as ns
from sklearn.linear_model import TweedieRegressor

regr = ns.MTS(obj=TweedieRegressor(),
              lags=25, # no. of time series lags
              type_pi="scp-vine-tll", # vine copula spec.
              replications=250, # no. of sample paths
              show_progress=False)
regr.fit(df_icecream);
regr.predict(h=30);
```

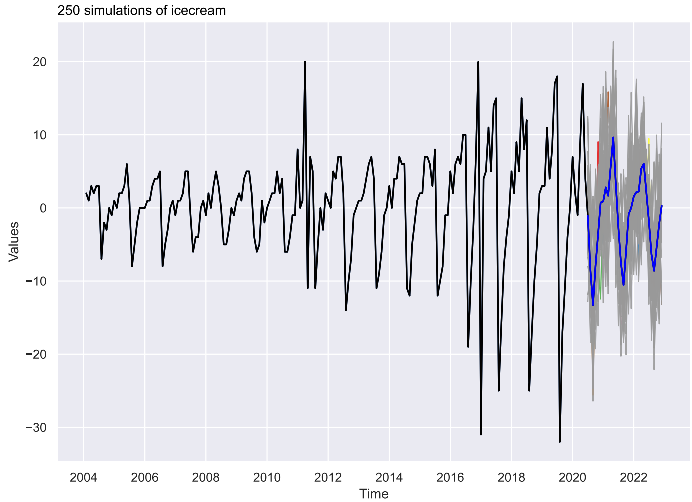
2 - 4 Using the `fit` + `predict` interface

```
regr.plot("heater", type_plot="spaghetti")
```



2 - 4 Using the `fit` + `predict` interface

```
regr.plot("icecream", type_plot="spaghetti")
```



2 - 5 Using GPUs

- Public notebook on GitHub (https://bit.ly/45RchgD)
- Simulated multivariate time series: 100 series, 10000 observations
- Ran on **Kaggle notebooks**, with **accelerator GPU P100**

```
import numpy as np
import pandas as pd
import nnetsauce as ns
from sklearn.linear_model import Ridge
from time import time
```

```
df = generate_synthetic_mts(n_steps=10000,
                            n_series=100,
                            amplitude=40, seed=14531)
df_ = df.diff().dropna()
```

2 - 5 Using GPUs (cont'd)

- **Example 1** on CPU, using Ridge regression

```
regr = Ridge()
obj_MTS = ns.MTS(regr,
                 lags = 15,
                 n_hidden_features=5,
                 nodes_sim="uniform",
                 backend="cpu", # specify backend
                 verbose = 1)
start = time()
obj_MTS.fit(df_)
print(f"Elapsed: {time()-start}")
```

```
Elapsed: 64.46652388572693
```

2 - 5 Using GPUs (cont'd)

- **Example 2** on GPU (uses **JAX** behind the scenes), using Ridge regression

```
regr = Ridge()
obj_MTS = ns.MTS(regr,
                 lags = 15,
                 n_hidden_features=5,
                 nodes_sim="uniform",
                 backend="gpu", # specify backend
                 verbose = 1)
start = time()
obj_MTS.fit(df_)
print(f"Elapsed: {time()-start}")
```

```
Elapsed: 40.53069853782654
```

- **37% time gain**

2 - 5 Using GPUs (cont'd)

- **Example 3** on GPU (uses **JAX** behind the scenes), using Ridge regression on GPU (`mlsauce` implementation) **too**

```
regr = ms.RidgeRegressor(reg_lambda=1.0, backend="gpu")
obj_MTS = ns.MTS(regr,
                 lags = 15,
                 n_hidden_features=5,
                 nodes_sim="uniform",
                 backend="gpu", # specify backend
                 verbose = 1)
start = time()
obj_MTS.fit(df_)
print(f"Elapsed: {time()-start}")
```

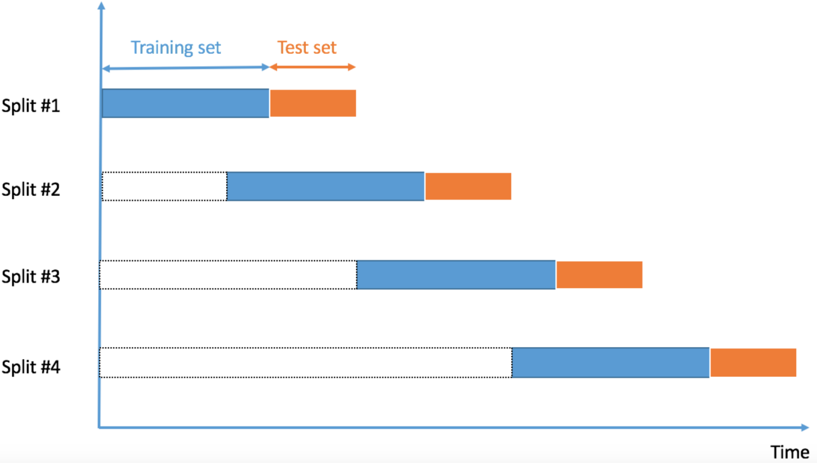
```
Elapsed: 23.551459312438965
```

- **63% time gain**
- Can also use xgboost with `tree_method='gpu_hist'` e.g

2 - 6 Time series cross-validation

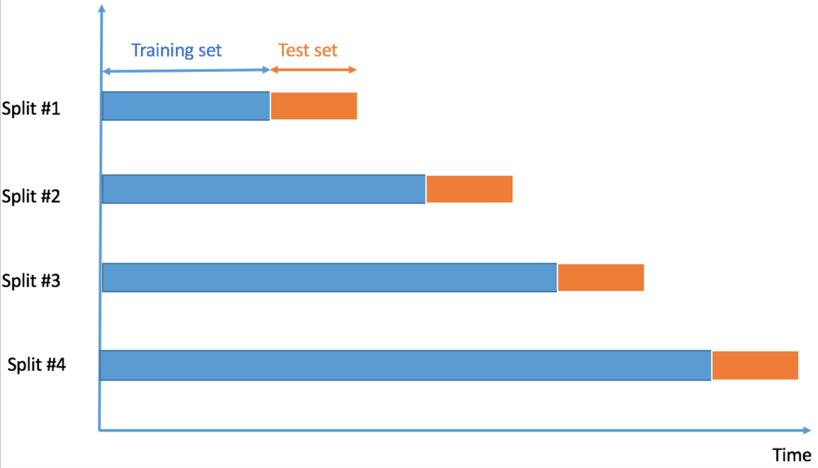
2 methods: fixed window and increasing window

fixed window



2 - 6 Time series cross-validation (cont'd)

increasing window



Example in `nnetsauce`

```python
import nnetsauce as ns
import numpy as np
import pandas as pd
import statsmodels.api as sm
from sklearn.linear_model import Ridge
from statsmodels.tsa.base.datetools import dates_from_str

# some example data
mdata = sm.datasets.macrodata.load_pandas().data
# prepare the dates index
dates = mdata[['year', 'quarter']].astype(int).astype(str)
quarterly = dates["year"] + "Q" + dates["quarter"]
quarterly = dates_from_str(quarterly)
mdata = mdata[['realgovt', 'tbilrate', 'cpi']]
mdata.index = pd.DatetimeIndex(quarterly)
data = np.log(mdata).diff().dropna()
```

2 - 6 Time series cross-validation (cont'd)

Example in `nnetsauce` (cont'd)

```
obj_MTS = ns.MTS(Ridge(), lags = 3,
                  n_hidden_features=7,
                   replications=100,
                   seed=24, verbose = 0,
                   type_pi="scp2-block-bootstrap",
                   show_progress=False)
cv = obj_MTS.cross_val_score(data,
                verbose = 0,
                initial_window=100,
                horizon=5,
                level=95,
                fixed_window=False, # True for rolling
                show_progress=False,
                scoring="coverage")[1]
```

2 - 6 Time series cross-validation (cont'd)

Example in `nnetsauce` (cont'd)

Average coverage on 98 samples

```
print(cv.mean)
```

```
## 92.51700680272108
```

Example in `nnetsauce` (cont'd)

```
obj_MTS = ns.MTS(Ridge(), lags = 3,
                 n_hidden_features=7,
                  replications=100,
                  seed=24, verbose = 0,
                  type_pi="scp2-block-bootstrap",
                  show_progress=False)
cv = obj_MTS.cross_val_score(data,
                verbose = 0,
                initial_window=100,
                horizon=5,
                level=95,
                fixed_window=True, # False for increasing
                show_progress=False,
                scoring="coverage")[1]
```

2 - 6 Time series cross-validation (cont'd)

Example in `nnetsauce` (cont'd)

Average coverage on 98 samples

```
print(cv.mean)
```

```
## 92.44897959183673
```

```python
# split data into training/testing set
n = data.shape[0]
max_idx_train = np.floor(n*0.9)
training_index = np.arange(0, max_idx_train)
testing_index = np.arange(max_idx_train, n)
df_train = data.iloc[training_index,:]
df_test = data.iloc[testing_index,:]
# Train + predict on 3 ML models
regr_mts = ns.LazyMTS(lags = 25,
                      type_pi="scp2-kde",
                      kernel="gaussian",
                      replications=250,
                      estimators=["Ridge",
                      "ElasticNet",
                      "RandomForestRegressor"],
                      show_progress=False);
models, predictions = regr_mts.fit(df_train, df_test);
```

2 - 7 Automated Machine Learning (AutoML) with LazyMTS (cont'd)

```python
print(models[['RMSE', 'COVERAGE']])
```

```
##                                RMSE  COVERAGE
## Model
## MTS(ElasticNet)                0.20     87.30
## MTS(RandomForestRegressor)     0.20     88.89
## MTS(Ridge)                     0.28     80.95
```

2 - 7 Automated Machine Learning (AutoML) with LazyMTS (cont'd)

```python
regr_mts = ns.LazyMTS(lags = 25,
                      type_pi="scp2-kde",
                      kernel="gaussian",
                      replications=250,
                      estimators="all", # the change
                      show_progress=False);
models, predictions = regr_mts.fit(df_train, df_test);
```

```
##   0%|          | 0/32 [00:00<?, ?it/s]  3%|3         |
```

2 - 7 Automated Machine Learning (AutoML) with LazyMTS (cont'd)

```python
print(models.sort_values(by='COVERAGE',
ascending=False)[['COVERAGE']].head(5))
```

```
##                                COVERAGE
## Model
## MTS(AdaBoostRegressor)            92.06
## MTS(BaggingRegressor)             92.06
## MTS(ExtraTreeRegressor)           90.48
## MTS(KNeighborsRegressor)          90.48
## MTS(ElasticNetCV)                 88.89
```