ForeTiS

Josef Eiglsperger, Florian Haselbeck, Dominik G. Grimm

Oct 18, 2023

CONTENTS

1 Contributors **3**

2 Citation **5**

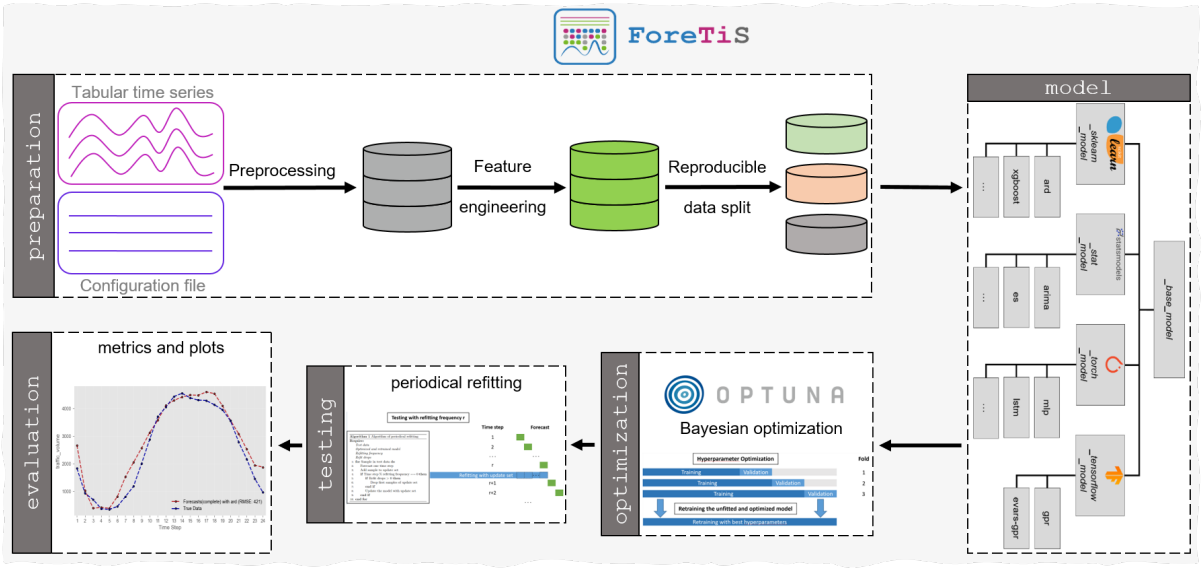
2.1 Installation Guide 5
2.2 Quickstart Guide 7
2.3 Tutorials 8
2.4 Data Guide 25
2.5 Prediction Models 26
2.6 `ForeTiS` 43

Python Module Index **105**

Index **107**

ForeTiS

ForeTiS is a Python framework that enables the rigorous training, comparison and analysis of predictions for a variety of different models. It is designed for seasonal time-series data. ForeTiS includes multiple state-of-the-art prediction models or machine learning methods, respectively. These range from classical models, such as regularized linear regression over ensemble learners, e.g. XGBoost, to deep learning-based architectures, such as Multilayer Perceptron (MLP). To enable automatic hyperparameter optimization, we leverage state-of-the-art and efficient Bayesian optimization techniques. Besides the named features, the forecasting models can adapt to changing trends and patterns in the data by being regularly updated in different periods with new data by so-called periodical refitting. Doing so can simulate a potential scenario for a productive operation. The subsequent scheme gives an overview of the structure of ForeTiS: In preparation, we summarize the fully automated and configurable data preprocessing and feature engineering. We already integrated several time series forecasting models in model from which the user can choose. Furthermore, the design of this module enables a straightforward integration of new prediction models. For automated hyperparameter optimization, we leverage state-of-the-art Bayesian optimization using the Python package Optuna. With the module testing, we allow the user to test different refitting procedures. Finally, we provide several methods to analyze results in evaluation. To start the optimization pipeline, users only need to supply a CSV file containing the data and a configuration file that enables pipeline customization. This design allows end users to apply time series forecasting with only a single-line command. In addition, we support researchers aiming to develop new forecasting methods with quick integration in a reliable framework and benchmarking against existing approaches.



In addition, our framework is designed to allow an easy and straightforward integration of further prediction models.

For more information, installation guides, tutorials and much more, see our documentation: https://ForeTiS.readthedocs.io/

Feel free to use the Docker workflow as described in our documentation: https://ForeTiS.readthedocs.io/en/latest/

tutorials.html#howto-run-ForeTiS-using-docker

CHAPTER

ONE

CONTRIBUTORS

This pipeline is developed and maintained by members of the Bioinformatics lab lead by Prof. Dr. Dominik Grimm:

- Florian Haselbeck, M.Sc.
- Josef Eiglsperger, M.Sc.

CHAPTER

TWO

CITATION

When using ForeTiS, please cite our publication:

ForeTiS: A comprehensive time series forecasting framework in Python.
Josef Eiglsperger*, Florian Haselbeck* and Dominik G. Grimm.
Machine Learning with Applications, 2023. doi: 10.1016/j.mlwa.2023.100467
** These authors have contributed equally to this work and share first authorship.*

2.1 Installation Guide

ForeTiS offers two ways of using it:

- *Docker workflow*: run the optimization pipeline with only one command in a Docker container

The whole framework was developed and tested using Ubuntu 20.04. Consequently, the subsequent guide is mainly written with regard to Ubuntu 20.04. The framework should work with Windows and Mac as well, but we do not officially provide support for these platforms. If you do not work on Ubuntu, we highly recommend the *Docker workflow*.

2.1.1 Docker workflow

If you want to do time series predictions without the need of integrating parts of your own pipeline, we recommend the *Docker workflow*: due to its easy-to-use interface and ready-to-use working environment within a Docker container. Besides the written tutorial, we provide a *Video tutorial: Docker workflow setup* embedded below.

Requirements

For the *Docker workflow*, Docker needs to be installed and running on your machine, see the Installation Guidelines at the Docker website. On Ubuntu, you can use `docker run hello-world` to check if Docker works (Caution: add sudo if you are not in the docker group).

If you want to use GPU support, you need to install nvidia-docker-2 (see this nvidia-docker Installation Guide) and a version of CUDA >= 11.2 (see this CUDA Installation Guide). To check your CUDA version, just run `nvidia-smi` in a terminal.

Setup

1. Open a Terminal and navigate to the directory where you want to set up the project
2. Clone this repository

```
git clone https://github.com/grimmlab/ForeTiS.git
```

3. Navigate to *Docker* after cloning the repository

```
cd ForeTiS/Docker
```

4. Build a Docker image using the provided Dockerfile tagged with the IMAGENAME of your choice

```
docker build -t IMAGENAME .
```

5. Run an interactive Docker container based on the created image with a CONTAINERNAME of your choice

```
docker run -it -v /PATH/TO/REPO/FOLDER:/REPO_DIRECTORY/IN/CONTAINER -v /
↪PATH/TO/DATA/DIRECTORY:/DATA_DIRECTORY/IN/CONTAINER -v /PATH/TO/RESULTS/
↪SAVE/DIRECTORY:/SAVE_DIRECTORY/IN/CONTAINER --name CONTAINERNAME IMAGENAME
```

   - Mount the directory where the repository is placed on your machine, the directory where your data is stored and the directory where you want to save your results using the option `-v`.
   - You can restrict the number of cpus using the option `cpuset-cpus CPU_INDEX_START-CPU_INDEX_STOP`.
   - Specify a gpu device using `--gpus device=DEVICE_NUMBER` if you want to use GPU support.

   Let's have a look at an example. We assume hat you created a Docker image called `foretis-image`, your repository and data is placed in (subfolders of) `/myhome/`, you want to save your results to `/myhome/` (so `/myhome/` is the only directory you need to mount in your container), you only want to use CPUs 0 to 9 and GPU 0 and you want to call your container `ForeTiS_container`. Then you have to run the following command:

```
docker run -it -v /myhome/:/myhome_in_my_container/ --cpuset-cpus 0-9 --
↪gpus device=0 --name ForeTiS_container foretis_image
```

Your setup is finished! Go to *HowTo: Run ForeTiS using Docker* to see how you can now use ForeTiS!

Useful Docker commands

The subsequent Docker commands might be useful when using ForeTiS. See here for a full guide on the Docker commands.

docker images
: List all Docker images on your machine

docker ps
: List all running Docker containers on your machine

docker ps -a
: List all Docker containers (including stopped ones) on your machine

docker start -i CONTAINERNAME
: Start a (stopped) Docker container interactively to enter its command line interface

Video tutorial: Docker workflow setup

https://youtu.be/SoTtsMZ70Uc

2.1.2 pip workflow

ForeTiS can be installed via `pip` and used as a common Python library:

```
pip install ForeTiS
```

Our setup is finished! Go to *HowTo: Use ForeTiS as a pip package* to see how you can now use ForeTiS!

The pipeline was developed and tested with Python 3.8 and Ubuntu 20.04. The framework should work with Windows and Mac as well, but we do not officially provide support for these platform. We neither officially support other Python versions, however ForeTiS should run as well for versions >= 3.8. If these requirements are not fulfilled, we recommend the *Docker workflow*.

2.2 Quickstart Guide

We offer ForeTiS both with a command line interface (CLI) and as a pip package.

For a quick start, see the following pages:

2.2.1 Command Line Interface

If you want to use ForeTiS' command line interface, we highly recommend the workflow using Docker to ensure a straightforward and proper setup of all dependencies.

To setup ForeTiS with Docker, see our Installation Guide: *Docker workflow*

To use ForeTiS' command line interface, see *HowTo: Run ForeTiS using Docker*

2.2.2 pip package

If you want to use ForeTiS as a pip package, e.g. to be able to integrate its functions to your own code, you can easily install it by just running:

```
pip install ForeTiS
```

More details can be found in the Installation Guide: *pip workflow*

For a walkthrough of ForeTiS' main time series forecasting pipeline using the pip workflow, see the following tutorial: *HowTo: Use ForeTiS as a pip package*

2.3 Tutorials

On the following pages, we show several tutorials for the usage of ForeTiS - often supported by videos.

2.3.1 HowTo: Run ForeTiS using Docker

We assume that you succesfully did all steps described in *Docker workflow*: to setup ForeTiS using Docker. Besides this written tutorial, we recorded a *Video tutorial: ForeTiS case studies*, where we run ForeTiS using Docker.

Workflow

You are at the **root directory within your Docker container**, i.e. after step 5 of the setup at *Docker workflow*:

If you closed the Docker container you created at the end of the installation, just use `docker start -i CONTAINERNAME` to start it in interactive mode again. If you did not create a container yet, go back to step 5 of the setup.

1. Navigate to the directory where the ForeTiS repository is placed within your container

   ```
   cd /REPO_DIRECTORY/IN/CONTAINER/ForeTiS
   ```

2. Run ForeTiS (as module). By default, ForeTiS starts the optimization procedure for 10 trials with XGBoost and a 5-fold nested cross-validation using the data we provide in `tutorials/tutorial_data`.

   ```
   python3 -m ForeTiS.run --save_dir SAVE_DIRECTORY
   ```

   That's it! Very easy! You can now find the results in the save directory you specified.

3. To get an overview of the different options you can set for running ForeTiS, just do:

   ```
   python3 -m ForeTiS.run --help
   ```

Feel free to test ForeTiS, e.g. with other prediction models. If you want to start using your own data, please carefully read our *Data Guide*: to ensure that your data fulfills all requirements.

2.3.2 HowTo: Use ForeTiS as a pip package

In this Jupyter notebook, we show how you can use ForeTiS as a pip package and also guide you through the steps that ForeTiS is doing when triggering an optimization run.

Please clone the whole GitHub repository if you want to run this tutorial on your own, as we need the tutorial data from our GitHub repository and to make sure that all paths we define are correct: `git clone https://github.com/grimmlab/ForeTiS.git`

Then, start a Jupyter notebook server on your machine and open this Jupyter notebook, which is placed at `docs/source/tutorials` in the repository.

However, you could also download the single files and define the paths yourself:

- The Jupyter notebook can be downloaded here: HowTo: Use ForeTiS as a pip package.ipynb
- The data we use can be found here: tutorial data

Installation, imports and paths

First, we may need to install ForeTiS (uncomment if it is not already installed). Then, we import ForeTiS as well as further libraries that we need in this tutorial. In the end, we define some paths and filenames which we will use more often throughout this tutorial. We will save the results in the same directory where this repository is placed.

[73]:
```
# !pip3 install --index-url https://test.pypi.org/simple/ --extra-index-url https://pypi.
↪org/simple/ ForeTiS
```

[74]:
```
import ForeTiS
import pathlib
import pandas as pd
import datetime
import pprint
```

[75]:
```
# Definition of paths and filenames
cwd = pathlib.Path.cwd()
data_dir = cwd.joinpath('tutorial_data')
save_dir = data_dir
model = 'xgboost'
```

Run whole optimization pipeline at once

As shown for the Docker workflow, ForeTiS offers a function optim_pipeline.run() that triggers the whole optimization run.

In the definition of `optim_pipeline.run()`, we set several default values. In order to run it using our tutorial data, we just need to define the data and directories we want to use as well as the models we want to optimize. Furthermore, we set values for the `datasplit` and `n_trials` to limit the waiting time for getting the results.

When calling the function, we first see some information regarding the data preprocessing and the configuration of our optimization run, e.g. the data that is used. Then, the current progress of the optuna optimization with results of the individual trials is shown. Afterwards, the optimized model with the best hyperparameters get retrained and tested on a unknown test set. This process of optimizing the hyperparameters and testing the model gets executed for every created feature set. In the end, we show a summary of the whole optimization run.

[76]:
```
ForeTiS.optim_pipeline.run(data_dir=data_dir, data='nike_sales', config_file_section=
↪'nike_sales', save_dir=data_dir, models=[model], n_trials=10, periodical_refit_
↪frequency=['complete'])
```

```
/home/josef/.local/lib/python3.10/site-packages/ForeTiS/optimization/optuna_optim.py:120:
↪ ExperimentalWarning: RetryFailedTrialCallback is experimental (supported from v2.8.0).
↪ The interface can change in the future.
  failed_trial_callback=optuna.storages.RetryFailedTrialCallback(max_retry=3))
[I 2023-03-03 21:50:28,146] A new study created in RDB with name: 2023-03-03_21-50-28_-
↪MODELxgboost-TRIALS10-FEATURESETdataset_full
```

```
---Dataset is already preprocessed---
### Dataset is loaded ###
### Starting Optuna Optimization for model xgboost and featureset dataset_full ###
## Starting Optimization
Params for Trial 0
{'n_estimators': 700, 'max_depth': 10, 'learning_rate': 0.225, 'gamma': 600, 'subsample':
↪ 0.2, 'colsample_bytree': 0.2, 'reg_lambda': 58.0, 'reg_alpha': 867.0}
```

```
[I 2023-03-03 21:50:30,959] Trial 0 finished with value: 12477267674.395584 and␣
↪parameters: {'n_estimators': 700, 'max_depth': 10, 'learning_rate': 0.225, 'gamma':␣
↪600, 'subsample': 0.2, 'colsample_bytree': 0.2, 'reg_lambda': 58.0, 'reg_alpha': 867.0}
↪. Best is trial 0 with value: 12477267674.395584.

Params for Trial 1
{'n_estimators': 800, 'max_depth': 8, 'learning_rate': 0.025, 'gamma': 970, 'subsample':␣
↪0.8500000000000001, 'colsample_bytree': 0.25, 'reg_lambda': 182.0, 'reg_alpha': 183.0}

[I 2023-03-03 21:50:34,236] Trial 1 finished with value: 23865390820.383522 and␣
↪parameters: {'n_estimators': 800, 'max_depth': 8, 'learning_rate': 0.025, 'gamma': 970,
↪ 'subsample': 0.8500000000000001, 'colsample_bytree': 0.25, 'reg_lambda': 182.0, 'reg_
↪alpha': 183.0}. Best is trial 0 with value: 12477267674.395584.

Params for Trial 2
{'n_estimators': 650, 'max_depth': 6, 'learning_rate': 0.15, 'gamma': 290, 'subsample':␣
↪0.6500000000000001, 'colsample_bytree': 0.15000000000000002, 'reg_lambda': 292.0, 'reg_
↪alpha': 366.0}

[I 2023-03-03 21:50:36,728] Trial 2 finished with value: 16570702631.08224 and␣
↪parameters: {'n_estimators': 650, 'max_depth': 6, 'learning_rate': 0.15, 'gamma': 290,
↪'subsample': 0.6500000000000001, 'colsample_bytree': 0.15000000000000002, 'reg_lambda':
↪ 292.0, 'reg_alpha': 366.0}. Best is trial 0 with value: 12477267674.395584.

Params for Trial 3
{'n_estimators': 750, 'max_depth': 9, 'learning_rate': 0.07500000000000001, 'gamma': 510,
↪ 'subsample': 0.6000000000000001, 'colsample_bytree': 0.05, 'reg_lambda': 608.0, 'reg_
↪alpha': 170.0}

[I 2023-03-03 21:50:39,557] Trial 3 finished with value: 32826934290.40348 and␣
↪parameters: {'n_estimators': 750, 'max_depth': 9, 'learning_rate': 0.07500000000000001,
↪ 'gamma': 510, 'subsample': 0.6000000000000001, 'colsample_bytree': 0.05, 'reg_lambda':
↪ 608.0, 'reg_alpha': 170.0}. Best is trial 0 with value: 12477267674.395584.

Params for Trial 4
{'n_estimators': 500, 'max_depth': 10, 'learning_rate': 0.3, 'gamma': 810, 'subsample':␣
↪0.35000000000000003, 'colsample_bytree': 0.1, 'reg_lambda': 684.0, 'reg_alpha': 440.0}

[I 2023-03-03 21:50:41,567] Trial 4 finished with value: 27304306183.488396 and␣
↪parameters: {'n_estimators': 500, 'max_depth': 10, 'learning_rate': 0.3, 'gamma': 810,
↪'subsample': 0.35000000000000003, 'colsample_bytree': 0.1, 'reg_lambda': 684.0, 'reg_
↪alpha': 440.0}. Best is trial 0 with value: 12477267674.395584.

Params for Trial 5
{'n_estimators': 550, 'max_depth': 6, 'learning_rate': 0.025, 'gamma': 910, 'subsample':␣
↪0.3, 'colsample_bytree': 0.7000000000000001, 'reg_lambda': 312.0, 'reg_alpha': 520.0}

[I 2023-03-03 21:50:43,798] Trial 5 finished with value: 41682236193.01747 and␣
↪parameters: {'n_estimators': 550, 'max_depth': 6, 'learning_rate': 0.025, 'gamma': 910,
↪ 'subsample': 0.3, 'colsample_bytree': 0.7000000000000001, 'reg_lambda': 312.0, 'reg_
↪alpha': 520.0}. Best is trial 0 with value: 12477267674.395584.

Params for Trial 6
{'n_estimators': 800, 'max_depth': 3, 'learning_rate': 0.3, 'gamma': 780, 'subsample': 0.
↪9500000000000001, 'colsample_bytree': 0.9000000000000001, 'reg_lambda': 598.0, 'reg_
↪alpha': 922.0}

[I 2023-03-03 21:50:46,862] Trial 6 finished with value: 10334330757.877611 and␣
↪parameters: {'n_estimators': 800, 'max_depth': 3, 'learning_rate': 0.3, 'gamma': 780,
```

(continues on next page)

(continued from previous page)

```
↪'subsample': 0.9500000000000001, 'colsample_bytree': 0.9000000000000001, 'reg_lambda':␣
↪598.0, 'reg_alpha': 922.0}. Best is trial 6 with value: 10334330757.877611.
```

```
Params for Trial 7
{'n_estimators': 500, 'max_depth': 3, 'learning_rate': 0.025, 'gamma': 320, 'subsample':␣
↪0.4, 'colsample_bytree': 0.3, 'reg_lambda': 829.0, 'reg_alpha': 357.0}
```

```
[I 2023-03-03 21:50:48,724] Trial 7 finished with value: 57564233462.41369 and␣
↪parameters: {'n_estimators': 500, 'max_depth': 3, 'learning_rate': 0.025, 'gamma': 320,
↪ 'subsample': 0.4, 'colsample_bytree': 0.3, 'reg_lambda': 829.0, 'reg_alpha': 357.0}.␣
↪Best is trial 6 with value: 10334330757.877611.
```

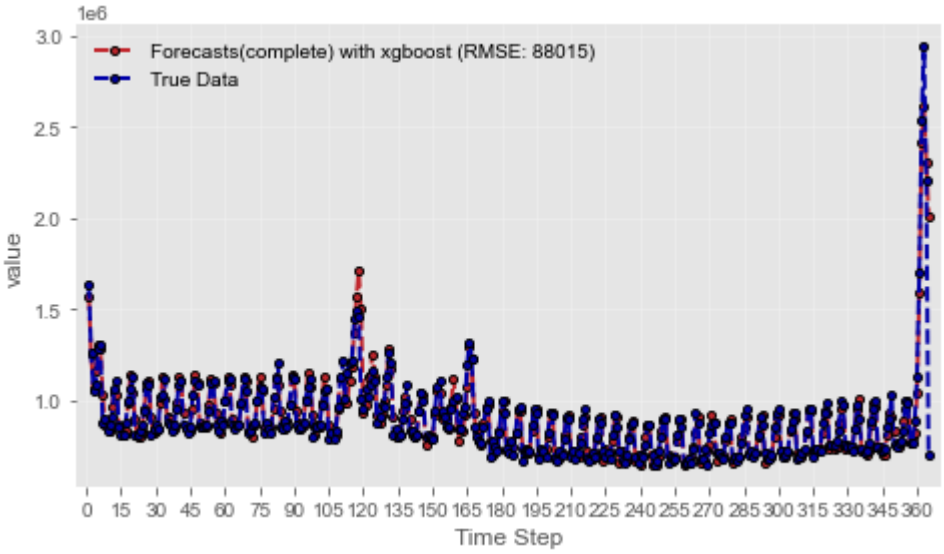
```
Params for Trial 8
{'n_estimators': 650, 'max_depth': 6, 'learning_rate': 0.05, 'gamma': 810, 'subsample':␣
↪0.1, 'colsample_bytree': 1.0, 'reg_lambda': 773.0, 'reg_alpha': 198.0}
```

```
[I 2023-03-03 21:50:51,289] Trial 8 finished with value: 59564589526.27223 and␣
↪parameters: {'n_estimators': 650, 'max_depth': 6, 'learning_rate': 0.05, 'gamma': 810,
↪'subsample': 0.1, 'colsample_bytree': 1.0, 'reg_lambda': 773.0, 'reg_alpha': 198.0}.␣
↪Best is trial 6 with value: 10334330757.877611.
```

```
Params for Trial 9
{'n_estimators': 500, 'max_depth': 9, 'learning_rate': 0.225, 'gamma': 730, 'subsample':␣
↪0.8, 'colsample_bytree': 0.1, 'reg_lambda': 358.0, 'reg_alpha': 115.0}
```

```
[I 2023-03-03 21:50:53,199] Trial 9 finished with value: 21784303396.630066 and␣
↪parameters: {'n_estimators': 500, 'max_depth': 9, 'learning_rate': 0.225, 'gamma': 730,
↪ 'subsample': 0.8, 'colsample_bytree': 0.1, 'reg_lambda': 358.0, 'reg_alpha': 115.0}.␣
↪Best is trial 6 with value: 10334330757.877611.
```

```
## Optuna Study finished ##
Study statistics:
  Finished trials:  10
  Pruned trials:  0
  Completed trials:  10
  Best Trial:  6
  Value:  10334330757.877611
  Params:
    colsample_bytree: 0.9000000000000001
    gamma: 780
    learning_rate: 0.3
    max_depth: 3
    n_estimators: 800
    reg_alpha: 922.0
    reg_lambda: 598.0
    subsample: 0.9500000000000001
## Retrain best model and test ##
## Results on test set with refitting period: complete ##
{'test_refitting_period_complete_mse': 7746675083.946438, 'test_refitting_period_
↪complete_rmse': 88015.19802821804, 'test_refitting_period_complete_r2_score': 0.
↪849709510924217, 'test_refitting_period_complete_explained_variance': 0.
↪8500353551866467, 'test_refitting_period_complete_MAPE': 4.376419954227729, 'test_
↪refitting_period_complete_sMAPE': 4.09452881498149}
```




```
### Finished Optuna Optimization for xgboost and featureset dataset_full ###
# Optimization runs done for models ['xgboost'] and ['dataset_full']
Results overview on the test set(s)
{'xgboost': {'dataset_full': {'Test': {'best_params': {'colsample_bytree': 0.
↪9000000000000001,
                                                       'gamma': 780,
                                                       'learning_rate': 0.3,
                                                       'max_depth': 3,
                                                       'n_estimators': 800,
                                                       'reg_alpha': 922.0,
                                                       'reg_lambda': 598.0,
                                                       'subsample': 0.9500000000000001},
                                       'eval_metrics': {'test_refitting_period_complete_
↪MAPE': 4.376419954227729,
                                                        'test_refitting_period_complete_
↪explained_variance': 0.8500353551866467,
                                                        'test_refitting_period_complete_
↪mse': 7746675083.946438,
                                                        'test_refitting_period_complete_
↪r2_score': 0.849709510924217,
                                                        'test_refitting_period_complete_
↪rmse': 88015.19802821804,
                                                        'test_refitting_period_complete_
↪sMAPE': 4.09452881498149},
                                       'retries': 0,
                                       'runtime_metrics': {'process_time_max': 18.
↪45159076599998,
                                                           'process_time_mean': 13.
↪805673200400008,
                                                           'process_time_min': 10.
↪010519391999992,
                                                           'process_time_std': 2.
↪932392361664868,
                                                           'real_time_max': 3.
```

(continues on next page)

(continued from previous page)

```
↪247318983078003,
                                                        'real_time_mean': 2.
↪4725477933883666,
                                                        'real_time_min': 1.
↪8300788402557373,
                                                        'real_time_std': 0.
↪4933237490143201}}}}}
```

```
<Figure size 432x288 with 0 Axes>
```

Within the defined `save_dir`, a `results` folder will be created.

Then, ForeTiS' default folder structure follows: `model/featureset/`.

We can see this structure below with all optimization results for the defined `model`.

[77]:
```
result_folders = list(save_dir.joinpath('results', model).glob('*'))
for results_dir in result_folders:
    print(results_dir)
```

```
/home/josef/Schreibtisch/01_HorticulturalSalesPrediction/ForeTiS/docs/source/tutorials/
↪tutorial_data/results/xgboost/2023-03-03_21-50-28_dataset_full
```

In the example below, we can see that each result folder contains different files with detailed results for each of the optimized models.

[78]:
```
result_elements = list(result_folders[0].glob('*'))
for result_element in result_elements:
    print(result_element)
```

```
/home/josef/Schreibtisch/01_HorticulturalSalesPrediction/ForeTiS/docs/source/tutorials/
↪tutorial_data/results/xgboost/2023-03-03_21-50-28_dataset_full/xgboost_dataset_full_
↪best_refitting_cycle_complete.png
/home/josef/Schreibtisch/01_HorticulturalSalesPrediction/ForeTiS/docs/source/tutorials/
↪tutorial_data/results/xgboost/2023-03-03_21-50-28_dataset_full/xgboost_runtime_
↪overview.csv
/home/josef/Schreibtisch/01_HorticulturalSalesPrediction/ForeTiS/docs/source/tutorials/
↪tutorial_data/results/xgboost/2023-03-03_21-50-28_dataset_full/final_model_test_
↪results.csv
/home/josef/Schreibtisch/01_HorticulturalSalesPrediction/ForeTiS/docs/source/tutorials/
↪tutorial_data/results/xgboost/2023-03-03_21-50-28_dataset_full/final_model_feature_
↪importances.csv
/home/josef/Schreibtisch/01_HorticulturalSalesPrediction/ForeTiS/docs/source/tutorials/
↪tutorial_data/results/xgboost/2023-03-03_21-50-28_dataset_full/unfitted_model_trial6
/home/josef/Schreibtisch/01_HorticulturalSalesPrediction/ForeTiS/docs/source/tutorials/
↪tutorial_data/results/xgboost/2023-03-03_21-50-28_dataset_full/Optuna_DB.db
/home/josef/Schreibtisch/01_HorticulturalSalesPrediction/ForeTiS/docs/source/tutorials/
↪tutorial_data/results/xgboost/2023-03-03_21-50-28_dataset_full/validation_results_
↪trial6.csv
```

The `*_runtime_overview.csv` file contains the best parameters, evaluation as well as runtime metrics for each of the optimized models as we can see in the example below.

[79]:
```
runtime_overview_file = [overview_file for overview_file in result_elements if '_runtime_
↪overview' in str(overview_file)][0]
```

(continues on next page)

(continued from previous page)

```
pd.read_csv(runtime_overview_file)
```

[79]:

	Trial	refitting_cycle	process_time_s	real_time_s	params	note
0	0	NaN	15.554622	2.766639	{'n_estimators': 700, 'max_depth': 10, 'learni...	successful
1	1	NaN	18.451591	3.247319	{'n_estimators': 800, 'max_depth': 8, 'learnin...	successful
2	2	NaN	13.801943	2.463260	{'n_estimators': 650, 'max_depth': 6, 'learnin...	successful
3	3	NaN	15.702649	2.794137	{'n_estimators': 750, 'max_depth': 9, 'learnin...	successful
4	4	NaN	10.769471	1.975961	{'n_estimators': 500, 'max_depth': 10, 'learni...	successful
5	5	NaN	12.254852	2.201480	{'n_estimators': 550, 'max_depth': 6, 'learnin...	successful
6	6	NaN	17.035096	3.030404	{'n_estimators': 800, 'max_depth': 3, 'learnin...	successful
7	7	NaN	10.010519	1.830079	{'n_estimators': 500, 'max_depth': 3, 'learnin...	successful
8	8	NaN	14.209953	2.536838	{'n_estimators': 650, 'max_depth': 6, 'learnin...	successful
9	9	NaN	10.266035	1.879361	{'n_estimators': 500, 'max_depth': 9, 'learnin...	successful
10	mean	NaN	13.805673	2.472548	NaN	NaN
11	std	NaN	2.932392	0.493324	NaN	NaN
12	max	NaN	18.451591	3.247319	NaN	NaN
13	min	NaN	10.010519	1.830079	NaN	NaN
14	retraining_after_10_trials	complete	8.822604	1.510213	{'colsample_bytree': 0.9000000000000001, 'gamm...	successful

Beyond that, we see below that the detailed results for each optimized model contain validation and test results, saved prediction models, an optuna database, a runtime overview with information for each trial (good for debugging, as pruning reasons are also documented) and for some prediction models also feature importances.

```
final_model_test_results_file = [overview_file for overview_file in result_elements if
↪'final_model_test_results' in str(overview_file)][0]
pd.read_csv(final_model_test_results_file)
```

[80]:

	y_pred_retrain	y_true_retrain	y_true_test
0	1.172851e+06	1.189235e+06	1.632762e+06
1	1.138915e+06	1.185445e+06	1.255720e+06
2	8.919995e+05	9.519060e+05	1.046709e+06
3	9.541891e+05	9.149935e+05	1.080526e+06
4	8.826827e+05	8.574261e+05	1.281535e+06
...	...	...	...
2183	NaN	NaN	NaN
2184	NaN	NaN	NaN

(continues on next page)

(continued from previous page)

```
2185             NaN             NaN           NaN
2186             NaN             NaN           NaN
2187             NaN             NaN           NaN

      y_pred_test_refitting_period_complete  \
0                                1572810.375
1                                1246359.750
2                                1061496.375
3                                1157195.750
4                                1304759.625
...                                      ...
2183                                     NaN
2184                                     NaN
2185                                     NaN
2186                                     NaN
2187                                     NaN

      y_pred_test_var_refitting_period_complete  \
0                                           0.0
1                                           0.0
2                                           0.0
3                                           0.0
4                                           0.0
...                                         ...
2183                                        NaN
2184                                        NaN
2185                                        NaN
2186                                        NaN
2187                                        NaN

      test_refitting_period_complete_mse  test_refitting_period_complete_rmse  \
0                           7.746675e+09                          88015.198028
1                                    NaN                                   NaN
2                                    NaN                                   NaN
3                                    NaN                                   NaN
4                                    NaN                                   NaN
...                                  ...                                   ...
2183                                 NaN                                   NaN
2184                                 NaN                                   NaN
2185                                 NaN                                   NaN
2186                                 NaN                                   NaN
2187                                 NaN                                   NaN

      test_refitting_period_complete_r2_score  \
0                                     0.84971
1                                         NaN
2                                         NaN
3                                         NaN
4                                         NaN
...                                       ...
2183                                      NaN
2184                                      NaN
```

(continues on next page)

(continued from previous page)

```
2185                                        NaN
2186                                        NaN
2187                                        NaN

      test_refitting_period_complete_explained_variance  \
0                                              0.850035
1                                                   NaN
2                                                   NaN
3                                                   NaN
4                                                   NaN
...                                                 ...
2183                                                NaN
2184                                                NaN
2185                                                NaN
2186                                                NaN
2187                                                NaN

      test_refitting_period_complete_MAPE  \
0                                 4.37642
1                                     NaN
2                                     NaN
3                                     NaN
4                                     NaN
...                                   ...
2183                                  NaN
2184                                  NaN
2185                                  NaN
2186                                  NaN
2187                                  NaN

      test_refitting_period_complete_sMAPE
0                                 4.094529
1                                      NaN
2                                      NaN
3                                      NaN
4                                      NaN
...                                    ...
2183                                   NaN
2184                                   NaN
2185                                   NaN
2186                                   NaN
2187                                   NaN

[2188 rows x 11 columns]
```

Further information

This notebooks shows how the use the ForeTiS pip package to run an optimization. Furthermore, we give an overview of the individual steps within `optim_pipeline.run()`.

For more information on specific topcis, see the following links:

- Documentation of the whole package
- easyPheno's GitHub repository
- Prepare your data according to our format: Data Guide
- The Installation Guide as well as basic tutorial for the Docker workflow as an alternative
- Several advanced topics such as adjusting existing prediction models or creation of new ones

[ ]:

2.3.3 Advanced Topics

The following subpackages contain information on advanced topics:

HowTo: Adjust existing prediction models and their hyperparameters

Every ForeTiS prediction model based on BaseModel needs to implement several methods. Most of them are already implemented in SklearnModel, SklearnModel, TorchModel and TensorflowModel. So if you make use of these, a prediction model only has to implement define_model() and define_hyperparams_to_tune(). We will therefore focus on these two methods in this tutorial.

If you want to create your own model, see *HowTo: Integrate your own prediction model*.

We already integrated several predictions models (see *Prediction Models*) , e.g. RidgeRegression and Mlp, which we will use for demonstration purposes in this HowTo.

Besides the written documentation, we recorded the tutorial video shown below with similar content.

Adjust prediction model

If you want to adjust the prediction model itself, you can change its definition in its implementation of `define_model()`. Let's discuss an example using RidgeRegression:

```
def define_model(self) -> sklearn.linear_model.Ridge:
    # Optimize if X gets standardized or not
    self.standardize_X = self.suggest_hyperparam_to_optuna('standardize_X')

    # Optimize the hyperparamter alpha
    alpha = self.suggest_hyperparam_to_optuna('alpha')

    # Set some hyperparameters that should not be optimized
    params = {}
    params.update({'random_state': 42})
    params.update({'fit_intercept': True})
    params.update({'copy_X': True})
    params.update({'max_iter': None})
```

(continues on next page)

(continued from previous page)

```
    params.update({'tol': 1e-3})
    params.update({'solver': 'auto'})
    return sklearn.linear_model.Ridge(alpha=alpha, **params)
```

You can change the `alpha` term that is actually used by setting the related variable to a fixed value or suggest it as a hyperparameter for tuning (see below for information on how to add or adjust a hyperparameter or its range). Beyond that, you could also adjust currently fixed parameters such as `max_iter`.

Another example can be found in Mlp:

```
def define_model(self) -> torch.nn.Sequential:
    n_layers = self.suggest_hyperparam_to_optuna('n_layers')
    model = []
    act_function = self.get_torch_object_for_string(string_to_get=self.suggest_
↪hyperparam_to_optuna('act_function'))
    self.n_features = self.dataset.shape[1] - 1
    in_features = self.n_features
    out_features = int(in_features * self.suggest_hyperparam_to_optuna('n_
↪initial_units_factor'))
    p = self.suggest_hyperparam_to_optuna('dropout')
    perc_decrease = self.suggest_hyperparam_to_optuna('perc_decrease_per_layer
↪')
    batch_norm = self.suggest_hyperparam_to_optuna('batch_norm')
    for layer in range(n_layers):
        model.append(torch.nn.Linear(in_features=in_features, out_features=out_
↪features))
        if act_function is not None:
            model.append(act_function)
        if batch_norm:
            model.append(torch.nn.BatchNorm1d(num_features=out_features))
        model.append(torch.nn.Dropout(p))
        in_features = out_features
        out_features = int(in_features * (1-perc_decrease))
    model.append(torch.nn.Linear(in_features=in_features, out_features=self.n_
↪outputs))
    model.append(torch.nn.Dropout(p))
    return torch.nn.Sequential(*model)
```

Currently, the model consists of `n_layers` of a sequence of a Linear, BatchNorm and Dropout layer, finally followed by a Linear output layer. You can easily adjust this by e.g. adding further layers or setting `n_layers` to a fixed value. Furthermore, the dropout rate `p` is optimized during hyperparameter search and the same rate is used for each Dropout layer. You could set this to a fixed value or suggest a different value for each Dropout layer (e.g. by suggesting it via `self.suggest_hyperparam_to_optuna('dropout')` within the `for`-loop). Some hyperparameters are already defined in TorchModel.common_hyperparams() ,which you can directly use here. Furthermore, some of them are already suggested in TorchModel.

Beyond that, you can also change the complete architecture of the model if you prefer to do so, maybe by copying the file and adding your changes there (see also *HowTo: Integrate your own prediction model*).

Adjust hyperparameters

Besides changing the model definition, you can adjust the hyperparameters that are optimized as well as their ranges. To set a hyperparameter to a fixed value, comment its suggestion and directly set a value, as described above. If you want to optimize a hyperparameter which is currently set to a fixed value, do it the other way round. If the hyperparameter is not yet defined in `define_hyperparams_to_tune()` Under construction. you have to add it to `define_hyperparams_to_tune()`.

Let's have a look at an example using Mlp:

```
def define_hyperparams_to_tune(self) -> dict:
    return {
        'n_initial_units_factor': {
            # Number of units in the first linear layer in relation to the␣
↪number of inputs
            'datatype': 'float',
            'lower_bound': 0.1,
            'upper_bound': 5,
            'step': 0.05
        },
        'perc_decrease_per_layer': {
            # Percentage decrease of the number of units per layer
            'datatype': 'float',
            'lower_bound': 0.05,
            'upper_bound': 0.5,
            'step': 0.05
        },
        'batch_norm': {
            'datatype': 'categorical',
            'list_of_values': [True, False]
        }
    }
```

There are multiple options to define a hyperparameter in ForeTiS, see define_hyperparams_to_tune() for more information regarding the format. In the example above, three parameters are optimized depending on the number of features, besides the ones which are defined in the parent class TorchModel in common_hyperparams(). The method has to return a dictionary. So if you want to add a further hyperparameter, you need to add it to the dictionary with its name as the key and a dictionary defining its characteristics such as the `datatype` and `lower_bound` in case of a float or int as the value. If you only want to change the range of an existing hyperparameter, you can just change the values in this method.

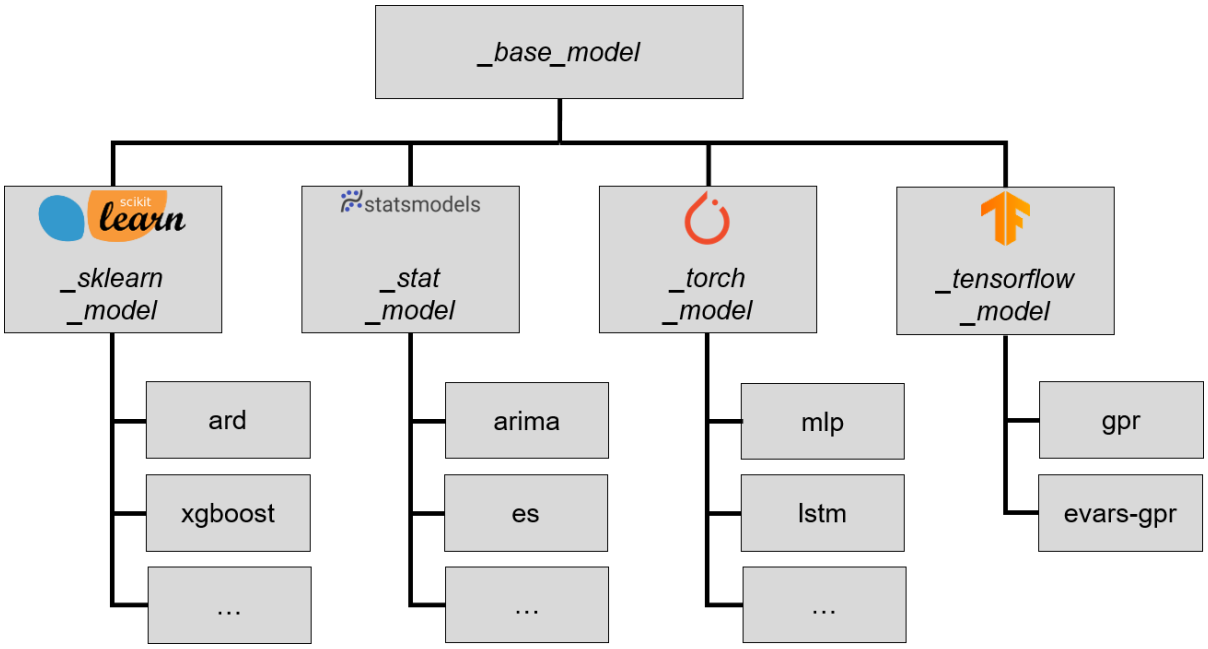
HowTo: Integrate your own prediction model

In this tutorial, we will show you how to integrate your own new prediction model into ForeTiS using an example. We recommend to first watch the *Code walkthrough video* for a better understanding of ForeTiS' structure.

We further recorded a *Video tutorial: Integrate new model*, which is embedded below .

Overview

The design of the model class makes ForeTiS easy extendable with new prediction models. The subsequent figure gives an overview on its structure.



All prediction models based on BaseModel. This base model defines some methods that are common for all prediction models as well as all methods that each prediction model needs to implement. ForeTiS already contains child classes of BaseModel implementing some of its obligatory methods for TensorFlow, PyTorch , sklearn and statsmodels. As a consequence, adding a new prediction model based on one of these four very common machine learning frameworks only requires the definition of two attributes and implementation of two methods, which makes ForeTiS easy extendable.

An example: Integrating k-nearest-neighbors

We provide template files for all three frameworks and focus on TemplateSklearnModel for the remainder of this tutorial:

```
import sklearn

from . import _sklearn_model
```

(continues on next page)

(continued from previous page)

```
class TemplateSklearnModel(_sklearn_model.SklearnModel):
    """
    Template file for a prediction model based on :obj:`~ForeTiS.model._sklearn_
↪model.SklearnModel`

    See :obj:`~ForeTiS.model._base_model.BaseModel` for more information on the␣
↪attributes.

    **Steps you have to do to add your own model:**

        1. Copy this template file and rename it according to your model (will␣
↪be the name to call it later on on the command line)

        2. Rename the class and add it to *ForeTiS.model.__init__.py*

        3. Adjust the class attributes if necessary

        4. Define your model in *define_model()*

        5. Define the hyperparameters and ranges you want to use for␣
↪optimization in *define_hyperparams_to_tune()*

        6. Test your new prediction model using toy data
    """

    def define_model(self):
        """
        Definition of the actual prediction model.

        Use *param = self.suggest_hyperparam_to_optuna(PARAM_NAME_IN_DEFINE_
↪HYPERPARAMS_TO_TUNE)* if you want to use
        the value of a hyperparameter that should be optimized.
        The function needs to return the model object.

        See :obj:`~ForeTiS.model._base_model.BaseModel` for more information.
        """
        ...

    def define_hyperparams_to_tune(self) -> dict:
        """
        Define the hyperparameters and ranges you want to optimize.
        Caution: they will only be optimized if you add them via *self.suggest_
↪hyperparam_to_optuna(PARAM_NAME)* in *define_model()*

        See :obj:`~ForeTiS.model._base_model.BaseModel` for more information on␣
↪the format and options.
        """
        return {
            'example_param_1': {
                'datatype': 'categorical',
                'list_of_values': ['cat', 'dog', 'elephant']
            },
```

(continues on next page)

(continued from previous page)

```
            'example_param_2': {
                'datatype': 'float',
                'lower_bound': 0.05,
                'upper_bound': 0.95,
                'step': 0.05
            },
            'example_param_3': {
                'datatype': 'int',
                'lower_bound': 1,
                'upper_bound': 100
            }
        }
```

As an example, we will integrate k-nearest-neighbors (knn) as a new prediction model.

First, we copy the template file into the folder containing ForeTiS's subpackage *model* and rename it to *knn.py*. Further, we rename the class within the file to `Knn` and add `"knn"` to `__all__` in ForeTiS.model.__init__.py.

So with updated comments (with `:obj:` references for linking in the auto-generated API documentation), our file now contains the following code:

```
import sklearn

from . import _sklearn_model


class Knn(_sklearn_model.SklearnModel):
    """
    Implementation of a class for k nearest neighbours regressor.

    def define_model(self):
        """
        Definition of the actual prediction model.

        See :obj:`~ForeTiS.model._base_model.BaseModel` for more information.
        """
        ...

    def define_hyperparams_to_tune(self) -> dict:
        """
        Definition of hyperparameters and ranges to optimize.

        See :obj:`~ForeTiS.model._base_model.BaseModel` for more information␣
↪on the format.
        """
        ...
```

Now we need to define the two attributes and implement the two methods. Further, we optimize the two hyperparameters `n_neighbors` and `weights`. These need to be suggested to optuna via `self.suggest_hyperparam_to_optuna(PARAM_NAME` in `define_model()` and defined with their ranges in `define_hyperparams_to_tune()` (see here for more information regarding the format and possible options for hyperparameter definition).

```
import sklearn

from . import _sklearn_model


class Knn(_sklearn_model.SklearnModel):
    """
    Implementation of a class for k nearest neighbours regressor.

    def define_model(self):
        """
        Definition of the actual prediction model.

        See :obj:`~ForeTiS.model._base_model.BaseModel` for more information.
        """
        n_neighbors = self.suggest_hyperparam_to_optuna('n_neighbors')
        weights = self.suggest_hyperparam_to_optuna('weights')
        return sklearn.neighbors.KNeighborsRegressor(n_neighbors=n_neighbors,␣
↪weights=weights)

    def define_hyperparams_to_tune(self) -> dict:
        """
        Definition of hyperparameters and ranges to optimize.

        See :obj:`~ForeTiS.model._base_model.BaseModel` for more information␣
↪on the format.
        """
        return {
            'n_neighbors': {
                'datatype': 'int',
                'lower_bound': 2,
                'upper_bound': 50,
                'step': 2
            },
            'weights': {
                'datatype': 'categorical',
                'list_of_values': ['uniform', 'distance']
            }
        }
```

Now we are able to test our new prediction model with toy data by calling `python3 -m ForeTiS.run` with the option `-mod knn` (see *HowTo: Run ForeTiS using Docker*).

This example gives an overview on how to integrate your own prediction model. Feel free to get guidance from existing prediction models as well. We are always happy to welcome new contributors and appreciate if you help improving ForeTiS by providing your code.

Video tutorial: Integrate new model

https://youtu.be/K9ZTd6rf-4M

Case studies video

In the subsequent video, we conduct case studies using ForeTiS:

Video tutorial: ForeTiS case studies

https://youtu.be/IKN5sjGuG-8

Code walkthrough video

In the subsequent video, we outline the structure and code of ForeTiS:

https://youtu.be/5ovp8K-JVhk

HowTo: Reuse optimized model

ForeTiS enables the reuse of an optimized model:

- Run inference using final model on new data with or without retraining on the whole old data using best hyperparameter combination

We provide scripts to run these functions (prefix *run_*) with our *Docker workflow*, on which we will also focus in this tutorial. If you want to use the functions directly (e.g. with the pip installed package), please check the scripts and see which functions are called.

Run inference on new data

The main use case for this is that you get new samples and you want to apply a previously optimized model on them. The final model therefore has to be saved and can be retrained using the found hyperparameters and old dataset. To apply this prediction model on new data, the structure of the old and new dataset need to match exactly!

To apply a final prediction model on new data, you have to run the following command:

```
python3 -m easypheno.postprocess.run_inference -rd full_path_to_model_results -
↪odd path_to_old_data -od name_of_old_dataset -ndd path_to_new_data -nd name_
↪of_new_dataset -sd path_to_save_directory
python3 -m ForeTiS.postprocess.run_inference -rd /Users/ge35tuv/Desktop/
↪ForeTiS/docs/source/tutorials/tutorial_data/results/ard/2023-10-17_12-35-47_
↪featureset-full_timeseries-cv_True_20_20 -odd /Users/ge35tuv/Desktop/ForeTiS/
↪docs/source/tutorials/tutorial_data -od nike_sales_old -ndd /Users/ge35tuv/
↪Desktop/ForeTiS/docs/source/tutorials/tutorial_data -nd nike_sales_new -sd␣
↪docs/source/tutorials/tutorial_data
```

By doing so, a .csv file containing the predictions on the whole new dataset will be created by applying the final prediction model, eventually after retraining on the old dataset if the final model was not saved.

2.4 Data Guide

To run ForeTiS on your data, you need to provide a CSV or HDF5/H5/H5PY file like described below. ForeTiS is designed to work with several file types besides the one we provide in this repository as tutorial data. For a better understanding, we provide a tutorial video, where we conduct case studies and in this course also go into detail about the data to be provided: tut_adv_casestudies.

2.4.1 CSV

To use your own CSV data, the dataset must be in such a manner that the dataset_specific_config.ini file can be filled like you can see in the figure below or as described under *dataset_specific_config.ini*. This figure is an example for a csv file that is suitable for ForeTiS. Important is that it contains a header with the naming of the columns, one sample per row, features and target value in the columns, and a column with the time information.

dteday	cal_season	cal_yr	cal_holiday	cal_workingday	weather_sit	weather_temp	weather_atemp	weather_hum	weather_windspeed	cnt
01.01.2011	1	0	0	0	2	0.344167	0.363625	0.805833	0.160446	985
02.01.2011	1	0	0	0	2	0.363478	0.353739	0.696087	0.248539	801
03.01.2011	1	0	0	1	1	0.196364	0.189405	0.437273	0.248309	1349
04.01.2011	1	0	0	1	1	0.2	0.212122	0.590435	0.160296	1562
05.01.2011	1	0	0	1	1	0.226957	0.22927	0.436957	0.1869	1600
06.01.2011	1	0	0	1	1	0.204348	0.233209	0.518261	0.0895652	1606
07.01.2011	1	0	0	1	2	0.196522	0.208839	0.498696	0.168726	1510
08.01.2011	1	0	0	0	2	0.165	0.162254	0.535833	0.266804	959
09.01.2011	1	0	0	0	1	0.138333	0.116175	0.434167	0.36195	822
10.01.2011	1	0	0	1	1	0.150833	0.150888	0.482917	0.223267	1321
11.01.2011	1	0	0	1	2	0.169091	0.191464	0.686364	0.122132	1263
12.01.2011	1	0	0	1	1	0.172727	0.160473	0.599545	0.304627	1162
13.01.2011	1	0	0	1	1	0.165	0.150883	0.470417	0.301	1406
14.01.2011	1	0	0	1	1	0.16087	0.188413	0.537826	0.126548	1421
15.01.2011	1	0	0	0	2	0.233333	0.248112	0.49875	0.157963	1248
16.01.2011	1	0	0	0	1	0.231667	0.234217	0.48375	0.188433	1204

dataset_specific_config.ini

In this file you can define some characteristics of your data. The following points should be adjusted:

- **values_for_counter:** the values that should trigger the counter adder
- **columns_for_counter:** the columns where the counter adder should be applied
- **columns_for_lags:** the columns that should be lagged by one sample
- **columns_for_rolling_mean:** the columns where the rolling mean should be applied
- **columns_for_lags_rolling_mean:** the columns where seasonal lagged rolling mean should be applied
- **imputation:** whether to perfrom imputation or not
- **resample_weekly:** whether to resample weekly or not
- **string_columns:** columns containing strings
- **float_columns:** columns containing floats
- **time_column:** columns containing the time information
- **time_format:** the time format, either “W”, “D”, or “H”
- **seasonal_periods:** how many datapoints one season has
- **featuresets_regex:** regular expression with which the feature sets should be filtered
- **features:** the features of the dataset

- **categorical_columns:** the categorical columns of the dataset
- **max_seasonal_lags:** maximal number of seasonal lags to be applied
- **target_column:** the target column for the prediction

In the *Video tutorial: ForeTiS case studies*, you can see exemplary on two case studies, how the configuration file should look like.

Preprocessing

In the preprocessing step, the actions defined in the dataset_specific_config.ini file will be performed. Additionally, useless columns get dropped and, if the amount of a focus product gets predicted, the correlating products gets calculated. Afterwards, in the featureadding and resampling step, the feature engineering happens where additional useful statisical and calendar features (like defined in the dataset_specific_config.ini file) get added and the categorical features get on hot encoded. Then, if defined, the data gets resampled and the datasets like described in HDF5 / H5 / H5PY will be created and saved. Once the dataset is preprocessed and the HDF5 file is generated, the algorithm recognized this when restarting experiments and directly reads in the HDF5 file to avoid redoing the time consuming preprocessing step.

2.5 Prediction Models

ForeTiS includes various time series forecasting models, both classical forecasting models as well as machine and deep learning-based methods. In the following pages, we will give some details for all of the currently implemented models. We further included a subpage explaining the Bayesian optimization that we use for our automatic hyperparameter search.

We provide both a workflow running ForeTiS with a command line interface using Docker and as a pip package, see the following tutorials for more details:

- *HowTo: Run ForeTiS using Docker*
- *HowTo: Use ForeTiS as a pip package*

In both cases, you need to select the prediction model you want to run - or also multiple ones within the same optimization run. A specific prediction model can be selected by giving the name of the *.py* file in which it is implemented (without the *.py* suffix). For instance, if you want to run Extreme Gradient Boost implemented in *xgboost.py*, you need to specify *xgboost*.

In the following table, we give the keys for all prediction models as well as links to detailed descriptions and the source code:

Table 1: Time Series Forecasting Models

Model	Key in ForeTiS	Description	Source Code
Automatic Relevance Determination Regression	ard	*Bayesian Regression*	ard.py
SARIMA	sarima	*SARIMA(X)*	sarima.py
SARIMAX	sarimax	*SARIMA(X)*	sarimax.py
Average Historical	averagehsitorical	*Baseline Models*	averagehistorical.py
Average Moving	averagemoving	*Baseline Models*	averagemoving.py
Average Seasonal	averageseasonal	*Baseline Models*	averageseasonal.py
Average Seasonal Lag	averageseasonal-lag	*Baseline Models*	averageseasonallag.py
Bayesian Ridge Regression	bayesridge	*Bayesian Regression*	bayesridge.py
Exponential Smoothing	es	*Exponential Smoothing*	es.py
EVARS-GPR	evars-gpr	*Gaussian Process Regression*	evars-gpr.py
Gaussian Process Regression (TensorFlow Implemetation)	gprtf	*Gaussian Process Regression*	gprtf.py
Lasso Regression	lasso	*Linear Regression*	lasso.py
Long Short-Term Memory (LSTM) Network	lstm	*LSTM Network*	lstm.py
Bayesian Long Short-Term Memory (LSTM) Network	lstmbayes	*LSTM Network*	lstmbayes.py
Elastic Net Regression	elasticnet	*Linear Regression*	elasticnet.py
Multilayer Perceptron	mlp	*Multilayer Perceptron*	mlp.py
Bayesian Multilayer Perceptron	mlpbayes	*Multilayer Perceptron*	bayesmlp.py
Ridge Regression	ridge	*Linear Regression*	ridge.py
XGBoost	xgboost	*XGBoost*	xgboost.py

Also, ForeTiS contains the follwing additional models. These are easy to integrate by just copy and pasting them to the models folder. They do not get support or updates anymore by the authors, as we decided to use other framework for the algorithms, but they still should work.

Table 2: Additional Time Series Forecasting Models

Model	Key in ForeTiS	Description	Source Code
Gaussian Process Regression (with sklearn framework)	gpr_sklearn	*Gaussian Process Regression*	gpr_sklearn.py
Bayesian Multilayer Perceptron (with IntelLabs BayesianTorch framework)	mlpbayes_intel	*Multilayer Perceptron*	mlpbayes_intel.py
Bayesian Long Short-Term Memory (LSTM) Network (with IntelLabs BayesianTorch framework)	lstmbayes_intel	*LSTM Network*	lstmbayes_intel.py

If you are interested in adjusting an existing model or its hyperparameters: *HowTo: Adjust existing prediction models and their hyperparameters*.

If you want to integrate your own prediction model: *HowTo: Integrate your own prediction model*.

2.5.1 Baseline Models

Subsequently, we give details on the baseline models that are integrated in ForeTiS.

Often, the Root-Mean-Squared-Error (RMSE) gets used as an evaluation metric in forecasting tasks. But due to the quadratic term, RMSE is sensitive to outliers. On the basis of these weaknesses and the lack of a universal evaluation metric for forecasting, it is common to assess performance compared to baseline methods. In the following, the in ForeTiS integrated four baseline models are listed:

$$AverageHistorical: \hat{y}_t = \frac{1}{t-1}\sum_{t=1}^{t-1} y_t$$

$$AverageMoving/RandomWalk: \hat{y}_t = \frac{1}{w}\sum_{t=t-w}^{t-1} y_t$$

$$AverageSeasonal: \hat{y}_t = \frac{1}{m}\sum_{t=t-m}^{t-1} y_t$$

$$AverageSeasonalLag: \hat{y}_t = \frac{1}{w}\sum_{t=t-m-w}^{t-m-1} y_t$$

All these four approaches - averagehistorical, averagemoving, averageseasonal, and averageseasonallag - are currently implemented in ForeTiS.

The following code block shows the implementation of averagemoving in averagemoving.py.

```
class AverageMoving(_baseline_model.BaselineModel):
    """
    Implementation of a class for AverageMoving.

    See :obj:`~ForeTiS.model._base_model.BaseModel` for more information on the␣
↪attributes.
    """

    def define_model(self):
        """
        Definition of the actual prediction model.

        See :obj:`~ForeTiS.model._base_model.BaseModel` for more information.
        """
        self.window = self.suggest_hyperparam_to_optuna('window')
        return AverageMoving

    def define_hyperparams_to_tune(self) -> dict:
        """
        See :obj:`~ForeTiS.model._base_model.BaseModel` for more information on␣
↪the format.
        """
        return {
            'window': {
                'datatype': 'int',
                'lower_bound': 1,
```

(continues on next page)

(continued from previous page)

```
                'upper_bound': 20
            }
        }
```

The other baseline models are implemented in a separate files containing very similar code. Its implementation can be found in averagehistorical.py, averageseasonal.py, and averageseasonallag.py.

References

1. Hyndman, R.J., Koehler, A.B., 2006. Another look at measures of forecast accuracy. International Journal of Forecasting 22, 679–688.

2.5.2 Exponential Smoothing

Subsequently, we give details on the Exponential Smoothing approache that is integrated in ForeTiS. For our implementation, we use the machine learning framework statsmodels, which also provides a user guide for these models.

Exponential smoothing is a univariate time series forecasting method.

The following code block shows the implementation of ES in es.py.

```
class Es(_stat_model.StatModel):
    """
    Implementation of a class for an Exponential Smoothing (ES) model.
    See :obj:`~ForeTiS.model._base_model.BaseModel` for more information on the␣
↪attributes.
    """

    def define_model(self) -> statsmodels.tsa.api.ExponentialSmoothing:
        """
        Definition of the actual prediction model.

        See :obj:`~ForeTiS.model._base_model.BaseModel` for more information.
        """
        self.remove_bias = self.suggest_hyperparam_to_optuna('remove_bias')
        self.use_brute = self.suggest_hyperparam_to_optuna('use_brute')
        endog = self.featureset[self.target_column].copy()

        trend = self.suggest_hyperparam_to_optuna('trend')
        damped_trend = self.suggest_hyperparam_to_optuna('damped_trend')
        seasonal = self.suggest_hyperparam_to_optuna('seasonal')
        seasonal_periods = self.suggest_hyperparam_to_optuna('seasonal_periods
↪')

        self.model_results = None

        if endog.eq(0).any().any() and seasonal == 'mul':
            endog += 0.01
        endog.index.freq = endog.index.inferred_freq

        if trend is None:
            damped_trend = False
```

(continues on next page)

(continued from previous page)

```
        return statsmodels.tsa.api.ExponentialSmoothing(endog=endog,␣
↪trend=trend, damped_trend=damped_trend,
                                                        seasonal=seasonal,␣
↪seasonal_periods=seasonal_periods)

    def define_hyperparams_to_tune(self) -> dict:
        """
        See :obj:`~ForeTiS.model._base_model.BaseModel` for more information on␣
↪the format.
        """
        return {
            'trend': {
                'datatype': 'categorical',
                'list_of_values': ['add', 'mul', None]
            },
            'damped_trend': {
                'datatype': 'categorical',
                'list_of_values': [False, True]
            },
            'seasonal': {
                'datatype': 'categorical',
                'list_of_values': ['add', 'mul', None]
            },
            'seasonal_periods': {
                'datatype': 'categorical',
                'list_of_values': [None, 52]
            },
            'use_brute': {
                'datatype': 'categorical',
                'list_of_values': [True, False]
            },
            'remove_bias': {
                'datatype': 'categorical',
                'list_of_values': [True, False]
            }
        }
```

2.5.3 SARIMA(X)

Subsequently, we give details on the SARIMAX approaches that are integrated in ForeTiS. For our implementation, we use the machine learning framework statsmodels, which also provides a user guide for these models.

We implemented the ARIMA method with seasonal component, called SARIMA or SARIMAX, respectively. ARIMAX is the abbreviation for Auto-Regressive Integrated Moving Average with eXogenous variables. These models consist of autoregressive components (AR), moving average component (MA), and a difference order I. (S)ARIMAX takes exogenous variables into account.

Both approaches - SARIMA and SARIMAX - are currently implemented in ForeTiS.

The following code block shows the implementation of SARIMA in sarima.py.

```
def define_model(self) -> pmdarima.ARIMA:
    """
    Definition of the actual prediction model.

    See :obj:`~ForeTiS.model._base_model.BaseModel` for more information.
    """
    self.conf = True

    self.use_exog = False

    P = self.suggest_hyperparam_to_optuna('P')
    D = self.suggest_hyperparam_to_optuna('D')
    Q = self.suggest_hyperparam_to_optuna('Q')
    seasonal_periods = self.suggest_hyperparam_to_optuna('seasonal_periods')
    p = self.suggest_hyperparam_to_optuna('p')
    d = self.suggest_hyperparam_to_optuna('d')
    q = self.suggest_hyperparam_to_optuna('q')

    self.trend = None

    order = [p, d, q]
    seasonal_order = [P, D, Q, seasonal_periods]
    model = pmdarima.ARIMA(order=order, seasonal_order=seasonal_order,␣
↪maxiter=50, disp=1, method='lbfgs',
                           with_intercept=True, enforce_stationarity=False,␣
↪suppress_warnings=True)
    return model

def define_hyperparams_to_tune(self) -> dict:
    """
    See :obj:`~ForeTiS.model._base_model.BaseModel` for more information on the␣
↪format.
    """
    return {
        'p': {
            'datatype': 'int',
            'lower_bound': 0,
            'upper_bound': 3
        },
        'd': {
            'datatype': 'int',
            'lower_bound': 0,
            'upper_bound': 1
        },
        'q': {
            'datatype': 'int',
            'lower_bound': 0,
            'upper_bound': 3
        },
        'P': {
            'datatype': 'int',
            'lower_bound': 0,
            'upper_bound': 3
```

(continues on next page)

(continued from previous page)

```
        },
        'D': {
            'datatype': 'int',
            'lower_bound': 0,
            'upper_bound': 1
        },
        'Q': {
            'datatype': 'int',
            'lower_bound': 0,
            'upper_bound': 3
        },
        'seasonal_periods': {
            'datatype': 'categorical',
            'list_of_values': [52]
        }
    }
```

SARIMAX is implemented in a separate files containing very similar code. Its implementation can be found in sarimax.py.

2.5.4 Linear Regression

Subsequently, we give details on the regularized linear regression approaches that are integrated in ForeTiS. References for a more detailed theoretical background can be found at the end of this page, which were also used for writing this text. For our implementation, we use the machine learning framework scikit-learn, which also provides a user guide for these models.

With respect to regularized linear regressions models, the model weights can be optimized by minimizing the deviation between predicted and true values, often with considering an additive penalty term for regularization:

$$\text{argmin}_{\mathbf{w}} \frac{1}{2}|\mathbf{y} - \mathbf{X}^*\mathbf{w}|_2^2 + \alpha\Omega(\mathbf{w})$$

In case of the Least Absolute Shrinkage and Selection Operator, usually abbreviated with LASSO, the L1-norm, so the sum of the absolute value of the weights, is used for regularization. This constraint usually leads to sparse solutions forcing unimportant weights to zero. Intuitively speaking, this can be seen as an automatic feature selection. The L2-norm, also known as the Euclidean norm, is defined as the square root of the summed up quadratic weights. Regularized linear regression using the L2-norm is called Ridge Regression. This penalty term has the effect of grouping correlated features. Elastic Net combines both the L1- and the L2-norm, introducing a further hyperparameter controlling the influence of each of the two parts.

All these three approaches - LASSO, Ridge and Elastic Net Regression - are currently implemented in ForeTiS.

The following code block shows the implementation of LASSO in lasso.py.

```
class Lasso(_sklearn_model.SklearnModel):
    """
    Implementation of a class for Lasso.

    See :obj:`~ForeTiS.model._base_model.BaseModel` for more information on the␣
↪attributes.
    """
```

(continues on next page)

(continued from previous page)

```
    def define_model(self) -> sklearn.linear_model.Lasso:
        """
        Definition of the actual prediction model.

        See :obj:`~ForeTiS.model._base_model.BaseModel` for more information.
        """
        # all hyperparameters defined for XGBoost are suggested for␣
↪optimization
        self.standardize_X = self.suggest_hyperparam_to_optuna('standardize_X')
        if self.standardize_X:
            self.x_scaler = sklearn.preprocessing.StandardScaler()

        alpha = self.suggest_hyperparam_to_optuna('alpha')
        params = {}
        params.update({'random_state': 42})
        params.update({'fit_intercept': True})
        params.update({'copy_X': True})
        params.update({'precompute': False})
        params.update({'max_iter': 10000})
        params.update({'tol': 1e-4})
        params.update({'warm_start': False})
        params.update({'positive': False})
        params.update({'selection': 'cyclic'})
        return sklearn.linear_model.Lasso(alpha=alpha, **params)

    def define_hyperparams_to_tune(self) -> dict:
        """
        See :obj:`~ForeTiS.model._base_model.BaseModel` for more information on␣
↪the format.
        """
        return {
            'alpha': {
                'datatype': 'float',
                'lower_bound': 10**-3,
                'upper_bound': 10**3,
                'log': True
            },
            'standardize_X': {
                'datatype': 'categorical',
                'list_of_values': [True, False]
            }
        }
```

The other two regression models are implemented in a separate files containing very similar code. Its implementation can be found in elasticnet.py and`ridge.py <https://github.com/grimmlab/ForeTiS/blob/main/ForeTiS/model/ridge.py>`_.

References

1. Hastie, T., Tibshirani, R., & Friedman, J. H. (2009). The elements of statistical learning: data mining, inference, and prediction. 2nd ed. New York, Springer.

2. Tibshirani, R. (1996). Regression shrinkage and selection via the lasso. Journal of the Royal Statistical Society:

Series B (Methodological), 58(1), 267–288.

3. Zou, H. and Hastie, T. (2005). Regularization and variable selection via the elastic net. Journal of the Royal Statistical Society, Series B, 67, 301–320.

4. Pedregosa, F. et al. (2011). Scikit-learn: Machine learning in Python. Journal of Machine Learning Research, 12, 2825–2830.

2.5.5 Bayesian Regression

Subsequently, we give details on the bayesian regression approaches that are integrated in ForeTiS. References for a more detailed theoretical background can be found at the end of this page, which were also used for writing this text. For our implementation, we use the machine learning framework scikit-learn, which also provides a user guide for these models.

Bayesian regression techniques can be understood as regularized linear regressions models (*Linear Regression*) where the regularization parameter is not set by introducing uninformative priors over the hyper parameters of the model. The L2-regularization used in Ridge regression is equivalent to finding a maximum a posteriori estimation under a Gaussian prior over the coefficients w with precision

$$\lambda^{-1}$$

Instead of setting lambda manually, it is possible to treat it as a random variable to be estimated from the data.

To obtain a fully probabilistic model, the output y is assumed to be Gaussian distributed around Xw:

$$p(y|X, w, \alpha) = N(y|Xw, \alpha)$$

where alpha is again treated as a random variable that is to be estimated from the data.

The difference between ARD and Bayesian Ridge Regression is a different prior over w.

Both approaches - ARD and Bayesian Regression - are currently implemented in ForeTiS.

The following code block shows the implementation of ARD in ard.py.

```
def define_model(self) -> sklearn.linear_model.ARDRegression:
    """
    Definition of the actual prediction model.

    See :obj:`~ForeTiS.model._base_model.BaseModel` for more information.
    """
    self.conf = True

    self.standardize_X = self.suggest_hyperparam_to_optuna('standardize_X')
    if self.standardize_X:
        self.x_scaler = sklearn.preprocessing.StandardScaler()

    alpha_1 = self.suggest_hyperparam_to_optuna('alpha_1')
    alpha_2 = self.suggest_hyperparam_to_optuna('alpha_2')
    lambda_1 = self.suggest_hyperparam_to_optuna('lambda_1')
    lambda_2 = self.suggest_hyperparam_to_optuna('lambda_2')
```

(continues on next page)

(continued from previous page)

```
    threshold_lambda = self.suggest_hyperparam_to_optuna('threshold_lambda')
    params = {}
    params.update({'fit_intercept': True})
    params.update({'n_iter': 10000})
    params.update({'tol': 1e-3})
    params.update({'copy_X': True})
    params.update({'verbose': False})
    params.update({'compute_score': False})
    return sklearn.linear_model.ARDRegression(alpha_1=alpha_1, alpha_2=alpha_2,
↪ lambda_1=lambda_1,
                                              lambda_2=lambda_2, threshold_
↪lambda=threshold_lambda, **params)

def define_hyperparams_to_tune(self) -> dict:
    """
    See :obj:`~ForeTiS.model._base_model.BaseModel` for more information on the␣
↪format.
    """
    return {
        'alpha_1': {
            'datatype': 'float',
            'lower_bound': 10**-3,
            'upper_bound': 10**3,
            'log': True
        },
        'alpha_2': {
            'datatype': 'float',
            'lower_bound': 10**-3,
            'upper_bound': 10**3,
            'log': True
        },
        'lambda_1': {
            'datatype': 'categorical',
            'list_of_values': [1e-6, 1e-5, 1e-4, 1e-3, 1e-2, 1e-1, 1, 10]
        },
        'lambda_2': {
            'datatype': 'categorical',
            'list_of_values': [1e-6, 1e-5, 1e-4, 1e-3, 1e-2, 1e-1, 1, 10]
        },
        'threshold_lambda': {
            'datatype': 'categorical',
            'list_of_values': [1e2, 1e3, 1e4, 1e5, 1e6]
        },
        'standardize_X': {
            'datatype': 'categorical',
            'list_of_values': [True, False]
        }
    }
```

The other regression model is implemented in a separate files containing very similar code. Its implementation can be found in bayesridge.py.

References

1. Pedregosa, F. et al. (2011). Scikit-learn: Machine learning in Python. Journal of Machine Learning Research, 12, 2825–2830.
2. D. J. C. MacKay, Bayesian Interpolation, Computation and Neural Systems, Vol. 4, No. 3, 1992.
3. M. E. Tipping, Sparse Bayesian Learning and the Relevance Vector Machine, Journal of Machine Learning Research, Vol. 1, 2001.
4. D. J. C. MacKay, Bayesian nonlinear modeling for the prediction competition, ASHRAE Transactions, 1994.
5. Christopher M. Bishop: Pattern Recognition and Machine Learning, 2006
6. Wipf, David, und Srikantan Nagarajan. „A New View of Automatic Relevance Determination“. In Advances in Neural Information Processing Systems, Bd. 20. Curran Associates, Inc., 2007.
7. Tristan Fletcher: Relevance Vector Machines Explained

2.5.6 Gaussian Process Regression

Subsequently, we give details on the Gaussian Process Regression approaches that are integrated in ForeTiS. References for a more detailed theoretical background can be found at the end of this page, which were also used for writing this text. For our implementation, we use the machine learning framework GPflow, which also provides a user guide for these models.

Gaussian Processes (GP) are a generic supervised learning method. When designed to solve regression, it is called Gaussian Process Regression (GPR). The prediction is probabilistic (Gaussian). Therefore, a empirical confidence intervals can be computed. EVent-triggered Augmented Refitting of Gaussian Process Regression for Seasonal Data (EVARS-GPR) handles sudden shifts in the target variable scale of seasonal data by combining online change point detection with a refitting of the GPR model using data augmentation for samples prior to a change point.

Both approaches - GPR and EVARS-GPR - are currently implemented in ForeTiS.

The following code block shows the implementation of GPR in gprtf.py. We completely outsourced the method in the parent class: _tensorflow_model.py. This is becasue both approaches share the almost the same method, except for the predict method that is redefined in the EVARS-GPR class: evars-gpr.py.

```
class Gpr(_tensorflow_model.TensorflowModel):
    """
    Implementation of a class for Gpr.

    See :obj:`~ForeTiS.model._base_model.BaseModel` for more information on the␣
↪attributes.
    """
```

References

1. Haselbeck, F., Grimm, D.G., 2021. EVARS-GPR: EVent-triggered Augmented Refitting of Gaussian Process Regression for SeasonalData.
2. Alexander G. de G. Matthews, Mark van der Wilk, Tom Nickson, Keisuke. Fujii, Alexis Boukouvalas, Pablo León-Villagrá, Zoubin Ghahramani, and James Hensman. GPflow: A Gaussian process library using TensorFlow. Journal of Machine Learning Research, 18(40):1–6, apr 2017.
3. Mark van der Wilk, Vincent Dutordoir, ST John, Artem Artemev, Vincent Adam, and James Hensman. A framework for interdomain and multioutput Gaussian processes. arXiv:2003.01115, 2020.

2.5.7 XGBoost

Subsequently, we give details on our implementation of extreme gradient Boosting, usually abbreviated with XGBoost. References for a more detailed theoretical background can be found at the end of this page, which were also used for writing this text. For our implementation, we use the library xgboost, which also provides a user guide.

XGBoost applies a technique called Boosting. Similar to Random Forest, XGBoost is also an ensemble learner, i.e. trying to build a strong prediction model based on multiple weak learners. But as a conceptual difference, weak learners in XGBoost are not independent. Instead, they are constructed sequentially, with putting more focus on the errors of the current ensemble for the training of a new weak learner. With Gradient Boosting, the sequential construction of the ensemble is formalized as a gradient descent algorithm on a loss function that needs to be minimized.

In comparison with Bagging, which is employed in Random Forest, Boosting aims to reduce bias instead of variance. This might lead to overfitting, which is aimed to be prevented by certain measures. One example is constraining the weak learners, so e.g. limiting the number of estimators or the depth of the Decision Trees. Further methods against overfitting are similar to concepts of bagging, e.g. using random subsets of samples and features for the training of each weak learner. Besides this, for XGBoost a learning rate shrinking the weights update for correcting ensemble errors during the learning process is typically used.

XGBoost is an efficient implementation that leverages Gradient Boosting, for which further details can be found in the original paper. It has proven its predictive power in many application areas, e.g. in Kaggle competitions.

For XGBoost, we use a specific library that is also available as a Python package. In the code block below, you can see our implementation. Furthermore, we optimize several hyperparameters, such as the number of weak learners (`n_estimators`) or the `learning_rate`. A full explanation of all XGBoost parameters can be found in their documentation: XGBoost Parameter Guide

```
class XgBoost(_sklearn_model.SklearnModel):
    """
    Implementation of a class for XGBoost.

    See :obj:`~ForeTiS.model._base_model.BaseModel` for more information on the␣
↪attributes.
    """

    def define_model(self) -> xgboost.XGBModel:
        """
        Definition of the actual prediction model.

        See :obj:`~ForeTiS.model._base_model.BaseModel` for more information.
        """
        # all hyperparameters defined for XGBoost are suggested for␣
↪optimization
        params = self.suggest_all_hyperparams_to_optuna()
        params.update({'random_state': 42})
        params.update({'verbosity': 0})
        params.update({'objective': 'reg:squarederror'})
        params.update({'tree_method': 'auto'})
        return xgboost.XGBRegressor(**params)


    def define_hyperparams_to_tune(self) -> dict:
        """
        See :obj:`~ForeTiS.model._base_model.BaseModel` for more information on␣
↪the format.
```

(continues on next page)

(continued from previous page)

```
        """
        return {
            'n_estimators': {
                'datatype': 'int',
                'lower_bound': 500,
                'upper_bound': 1000,
                'step': 50
            },
            'max_depth': {
                'datatype': 'int',
                'lower_bound': 2,
                'upper_bound': 10
            },
            'learning_rate': {
                'datatype': 'float',
                'lower_bound': 0.025,
                'upper_bound': 0.3,
                'step': 0.025
            },
            'gamma': {
                'datatype': 'int',
                'lower_bound': 0,
                'upper_bound': 1000,
                'step': 10
            },
            'subsample': {
                'datatype': 'float',
                'lower_bound': 0.05,
                'upper_bound': 1.0,
                'step': 0.05
            },
            'colsample_bytree': {
                'datatype': 'float',
                'lower_bound': 0.05,
                'upper_bound': 1.0,
                'step': 0.05
            },
            'reg_lambda': {
                'datatype': 'float',
                'lower_bound': 0,
                'upper_bound': 1000,
                'step': 1
            },
            'reg_alpha': {
                'datatype': 'float',
                'lower_bound': 0,
                'upper_bound': 1000,
                'step': 1
            }
        }
```

References

1. Chen, T., & Guestrin, C. (2016). XGBoost: A Scalable Tree Boosting System. In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (pp. 785–794). New York, NY, USA: ACM.

2.5.8 LSTM Network

Subsequently, we give details on our implementation of a Long Short-Term Memory (LSTM) Network. References for a more detailed theoretical background can be found at the end of this page, which were also used for writing this text. We use PyTorch for our implementation. For more information on specific PyTorch objects that we use, e.g. layers, see the PyTorch documentation.

Some of the methods and attributes relevant for the LSTM are already defined in its parent class TorchModel. There, you can e.g. find the epoch- and batch-wise training loop. In the code block below, we show the constructor of TorchModel.

```
class TorchModel(_base_model.BaseModel, abc.ABC):
    def __init__(self, optuna_trial: optuna.trial.Trial, datasets: list,␣
↪featureset_name: str, optimize_featureset: bool,
    pca_transform: bool = None, current_model_name: str = None, batch_size:␣
↪int = None,
    n_epochs: int = None, target_column: str = None):
    self.all_hyperparams = self.common_hyperparams()
    self.current_model_name = current_model_name
    super().__init__(optuna_trial=optuna_trial, datasets=datasets, featureset_
↪name=featureset_name,
                     target_column=target_column, pca_transform=pca_transform,
                     optimize_featureset=optimize_featureset)
    self.batch_size = \
        batch_size if batch_size is not None else 2**self.suggest_hyperparam_
↪to_optuna('batch_size_exp')
    self.n_epochs = n_epochs if n_epochs is not None else self.suggest_
↪hyperparam_to_optuna('n_epochs')
    self.optimizer = torch.optim.Adam(params=self.model.parameters(),
                                      lr=self.suggest_hyperparam_to_optuna(
↪'learning_rate'))
    self.loss_fn = torch.nn.MSELoss()
    # early stopping if there is no improvement on validation loss for a␣
↪certain number of epochs
    self.early_stopping_patience = self.suggest_hyperparam_to_optuna('early_
↪stopping_patience')
    self.early_stopping_point = None
    self.device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu
↪')
    self.X_scaler = sklearn.preprocessing.StandardScaler()
    self.enabled = True
```

We define attributes and suggest hyperparameters that are relevant for all neural network implementations, e.g. the `optimizer` to use and the `learning_rate` to apply. Some attributes are also set to fixed values, for instance the loss function (`self.loss_fn`) depending on the detected machine learning task. Furthermore, early stopping is parametrized, which we use as a measure to prevent overfitting. With early stopping, the validation loss is monitored and if it does not improve for a certain number of epochs (`self.early_stopping_patience`), the training process is stopped. When working with our MLP implementation, it is important to keep in mind that some relevant code and hyperparameters can also be found in TorchModel.

The definition of the LSTM model itself as well as of some specific hyperparameters and ranges can be found in the

LSTM class. In the code block below, we show its `define_model()` method. Our LSTM model consists of one layer, which include a `LSTM()`, `Dropout`, and `Linear()` layer. The number of layers of the LSTM layer s is defined by a hyperparameter (`n_lstm_layers`). Further, we use `Dropout` for regularization and define the dropout rate as the hyperparameter `p`. Finally, we transform the list to which we added all network layers into a `torch.nn.Sequential()` object.

```
def define_model(self) -> torch.nn.Sequential:
    """
    Definition of a LSTM network.

    Architecture:
        - LSTM, Dropout, Linear
        - Linear output layer

    Number of output channels of the first layer, dropout rate, frequency of a␣
↪doubling of the output channels and
    number of units in the first linear layer. may be fixed or optimized.
    """
    self.y_scaler = sklearn.preprocessing.StandardScaler()
    self.sequential = True
    self.seq_length = self.suggest_hyperparam_to_optuna('seq_length')
    model = []
    p = self.suggest_hyperparam_to_optuna('dropout')
    n_feature = self.dataset.shape[1]
    lstm_hidden_dim = self.suggest_hyperparam_to_optuna('lstm_hidden_dim')

    model.append(PrepareForlstm())
    model.append(torch.nn.LSTM(input_size=n_feature, hidden_size=lstm_hidden_
↪dim,
                               num_layers=self.suggest_hyperparam_to_optuna('n_
↪lstm_layers'), dropout=p))
    model.append(GetOutputZero())
    model.append(PrepareForDropout())
    model.append(torch.nn.Dropout(p))
    model.append(torch.nn.Linear(in_features=lstm_hidden_dim, out_
↪features=self.n_outputs))
    return torch.nn.Sequential(*model)
```

`self.n_outputs` is inherited from `BaseModel`, where it is set to 1 for the regression task (one continuous output).

Also, we implemented the Bayesian form of the LSTM model which can be found in the LSTMbayes class.

References

1. Bishop, Christopher M. (2006). Pattern recognition and machine learning. New York, Springer.
2. Goodfellow, I., Bengio, Y.,, Courville, A. (2016). Deep Learning. MIT Press. Available at https://www.deeplearningbook.org/
3. Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. Weight uncertainty in neural networks. arXiv preprint arXiv:1505.05424, 2015.
4. Hochreiter, Sepp & Schmidhuber, Jürgen. (1997). Long Short-term Memory. Neural computation. 9. 1735-80. 10.1162/neco.1997.9.8.1735.

2.5.9 Multilayer Perceptron

Subsequently, we give details on our implementation of a Multilayer Perceptron (MLP, also known as feedforward neural network). References for a more detailed theoretical background can be found at the end of this page, which were also used for writing this text. We use PyTorch for our implementation. For more information on specific PyTorch objects that we use, e.g. layers, see the PyTorch documentation.

Some of the methods and attributes relevant for the MLP are already defined in its parent class TorchModel. There, you can e.g. find the epoch- and batch-wise training loop. In the code block below, we show the constructor of TorchModel.

```
class TorchModel(_base_model.BaseModel, abc.ABC):
    def __init__(self, optuna_trial: optuna.trial.Trial, datasets: list,␣
↪featureset_name: str, optimize_featureset: bool,
    pca_transform: bool = None, current_model_name: str = None, batch_size:␣
↪int = None,
    n_epochs: int = None, target_column: str = None):
    self.all_hyperparams = self.common_hyperparams()
    self.current_model_name = current_model_name
    super().__init__(optuna_trial=optuna_trial, datasets=datasets, featureset_
↪name=featureset_name,
                     target_column=target_column, pca_transform=pca_transform,
                     optimize_featureset=optimize_featureset)
    self.batch_size = \
        batch_size if batch_size is not None else 2**self.suggest_hyperparam_
↪to_optuna('batch_size_exp')
    self.n_epochs = n_epochs if n_epochs is not None else self.suggest_
↪hyperparam_to_optuna('n_epochs')
    self.optimizer = torch.optim.Adam(params=self.model.parameters(),
                                      lr=self.suggest_hyperparam_to_optuna(
↪'learning_rate'))
    self.loss_fn = torch.nn.MSELoss()
    # early stopping if there is no improvement on validation loss for a␣
↪certain number of epochs
    self.early_stopping_patience = self.suggest_hyperparam_to_optuna('early_
↪stopping_patience')
    self.early_stopping_point = None
    self.device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu
↪')
    self.X_scaler = sklearn.preprocessing.StandardScaler()
    self.enabled = True
```

We define attributes and suggest hyperparameters that are relevant for all neural network implementations, e.g. the `optimizer` to use and the `learning_rate` to apply. Some attributes are also set to fixed values, for instance the loss function (`self.loss_fn`) depending on the detected machine learning task. Furthermore, early stopping is parametrized, which we use as a measure to prevent overfitting. With early stopping, the validation loss is monitored and if it does not improve for a certain number of epochs (`self.early_stopping_patience`), the training process is stopped. When working with our MLP implementation, it is important to keep in mind that some relevant code and hyperparameters can also be found in TorchModel.

The definition of the MLP model itself as well as of some specific hyperparameters and ranges can be found in the Mlp class. In the code block below, we show its `define_model()` method. Our MLP model consists of `n_layers` of blocks, which include a `Linear()`, `BatchNorm()` and `Dropout` layer. The last of these blocks is followed by a `Linear()` output layer. The number of outputs in the first layers is defined by a hyperparameter (`n_initial_units_factor`), that is multiplied with the number of inputs. Then, with each of the above-mentioned blocks, the number of outputs decreases by a percentage parameter `perc_decrease`. Further, we use `Dropout` for

regularization and define the dropout rate as the hyperparameter `p`. Finally, we transform the list to which we added all network layers into a `torch.nn.Sequential()` object.

```
def define_model(self) -> torch.nn.Sequential:
    """
    Definition of an MLP network.

    Architecture:

        - N_LAYERS of (Linear (+ ActivationFunction) (+ BatchNorm) + Dropout)
        - Linear output layer
        - Dropout layer

    Number of units in the first linear layer and percentage decrease after␣
↪each may be fixed or optimized.
    """
    n_layers = self.suggest_hyperparam_to_optuna('n_layers')
    model = []
    act_function = self.get_torch_object_for_string(string_to_get=self.suggest_
↪hyperparam_to_optuna('act_function'))
    self.n_features = self.featureset.shape[1] - 1
    in_features = self.n_features
    out_features = int(in_features * self.suggest_hyperparam_to_optuna('n_
↪initial_units_factor'))
    p = self.suggest_hyperparam_to_optuna('dropout')
    perc_decrease = self.suggest_hyperparam_to_optuna('perc_decrease_per_layer
↪')
    batch_norm = self.suggest_hyperparam_to_optuna('batch_norm')
    for layer in range(n_layers):
        model.append(torch.nn.Linear(in_features=in_features, out_features=out_
↪features))
        if act_function is not None:
            model.append(act_function)
        if batch_norm:
            model.append(torch.nn.BatchNorm1d(num_features=out_features))
        model.append(torch.nn.Dropout(p=p))
        in_features = out_features
        out_features = int(in_features * (1-perc_decrease))
    model.append(torch.nn.Linear(in_features=in_features, out_features=self.n_
↪outputs))

    return torch.nn.Sequential(*model)
```

`self.n_outputs` is inherited from `BaseModel`, where it is set to 1 for the regression task (one continuous output).

Also, we implemented the Bayesian form of the MLP model which can be found in the Mlpbayes class.

References

1. Bishop, Christopher M. (2006). Pattern recognition and machine learning. New York, Springer.
2. Goodfellow, I., Bengio, Y.,, Courville, A. (2016). Deep Learning. MIT Press. Available at https://www.deeplearningbook.org/
3. Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. Weight uncertainty in neural networks. arXiv preprint arXiv:1505.05424, 2015.

2.5.10 Hyperparameter optimization

Subsequently, we give details on our implementation of Bayesian optimization for the automatic hyperparameter search. References for a more detailed theoretical background can be found at the end of this page, which were also used for writing this text. For our implementation, we use the optimization framework Optuna, for which its developers also provide a comprehensive online documentation.

Common hyperparameter optimization methods, e.g. Grid Search or Random Search, do not make use of information gained during the optimization process. However, Bayesian optimization uses this knowledge and tries to direct the hyperparameter search towards more promising parameter candidates. The search is guided by a so-called objective value, which is in the machine learning context usually the performance on validation data. With this objective value, a probability model mapping from parameter candidates to a probability of an objective value can be defined. As a result, the most promising parameters can be selected for further trials. This trial-wise optimization with using existing knowledge makes Bayesian optimization potentially more efficient than Grid or Random Search, despite the computational resources needed for the selection of parameter candidates.

Our implementation can be found in the class OptunaOptim. Besides results saving, the main part of this class can be found in the `objective()` method. This method is called for each new trial. At the beginning of a new trial, a prediction model using the suggested parameter set is defined. Then, we loop over the whole training and validation data to retrieve the objective value. In case of multiple validation sets, we take the mean value.

To improve efficiency, we implemented pruning based on intermediate results - so results on validation sets within the cross-validation - and stop a trial if the intermediate result is worse than the 80th percentile of previous ones at the same time. The probability that such a parameter set would let to better results in the end is pretty low. Furthermore, we set the number of finished trials before we start with pruning to 20. Besides that, we check for parameter set duplicates, as the implementation of Optuna does not prevent to suggest the same parameters again (if they are most likely the best ones to suggest in the current state). The whole optimization process is saved in a database for debugging purposes.

For more detailed information regarding the objects and functions we use from Optuna, see the Optuna documentation.

References

1. Akiba, T., Sano, S., Yanase, T., Ohta, T., & Koyama, M. (2019). Optuna: A Next-generation Hyperparameter Optimization Framework.
2. Bergstra, J., Bardenet, Ré., Bengio, Y. & Kégl, B. (2011). Algorithms for Hyper-parameter Optimization.
3. Snoek, J., Larochelle, H. & Adams, R. P. (2012). Practical bayesian optimization of machine learning algorithms.

2.6 ForeTiS

2.6.1 Subpackages

ForeTiS.evaluation

Submodules

ForeTiS.evaluation.eval_metrics

Module Contents

Functions

smape(y_true, y_pred)	Function delivering Symmetric Mean Absolute Percentage Error between prediction and actual values
mape(y_true, y_pred)	Function delivering Mean Absolute Percentage Error between prediction and actual values
get_evaluation_report(y_true, y_pred[, prefix, ...])	Get values for common evaluation metrics

ForeTiS.evaluation.eval_metrics.**smape**(*y_true*, *y_pred*)

Function delivering Symmetric Mean Absolute Percentage Error between prediction and actual values :param y_true: actual values :param y_pred: prediction values :return: sMAPE between prediction and actual values

Parameters

- **y_true** (*numpy.array*) –
- **y_pred** (*numpy.array*) –

Return type
float

ForeTiS.evaluation.eval_metrics.**mape**(*y_true*, *y_pred*)

Function delivering Mean Absolute Percentage Error between prediction and actual values :param y_true: actual values :param y_pred: prediction values :return: MAPE between prediction and actual values

Parameters

- **y_true** (*numpy.array*) –
- **y_pred** (*numpy.array*) –

Return type
float

ForeTiS.evaluation.eval_metrics.**get_evaluation_report**(*y_true*, *y_pred*, *prefix=''*, *current_model_name=None*)

Get values for common evaluation metrics

Parameters

- **y_true** (*numpy.array*) – true values
- **y_pred** (*numpy.array*) – predicted values
- **prefix** (*str*) – prefix to be added to the key if multiple eval metrics are collected
- **current_model_name** (*str*) – name of the current model according to naming of .py file in package model

Returns
dictionary with common metrics

Return type
dict

ForeTiS.model

Subpackages

ForeTiS.model._additionalmodels

Submodules

ForeTiS.model._additionalmodels.gpr_sklearn

Module Contents

Classes

Gpr	Implementation of a class for Gpr.

class ForeTiS.model._additionalmodels.gpr_sklearn.**Gpr**(*optuna_trial*, *datasets*, *featureset_name*, *pca_transform*, *target_column*, *optimize_featureset*)

Bases: *ForeTiS.model._sklearn_model.SklearnModel*

Implementation of a class for Gpr.

See *BaseModel* for more information on the attributes.

Parameters

- **optuna_trial** (*optuna.trial.Trial*) –
- **datasets** (*list*) –
- **featureset_name** (*str*) –
- **pca_transform** (*bool*) –
- **target_column** (*str*) –
- **optimize_featureset** (*bool*) –

define_model()

Definition of the actual prediction model.

See *BaseModel* for more information.

Return type
sklearn.gaussian_process.GaussianProcessRegressor

define_hyperparams_to_tune()

See *BaseModel* for more information on the format.

Return type
dict

extend_kernel_combinations()

Function extending kernels list with combinations based on base_kernels

ForeTiS.model._additionalmodels.lstmbayes_intel

Module Contents

Classes

LSTM	Implementation of a class for a bayesian Long Short-Term Memory (LSTM) network.

class ForeTiS.model._additionalmodels.lstmbayes_intel.**LSTM**(*optuna_trial*, *datasets*, *featureset_name*, *optimize_featureset*, *pca_transform=None*, *current_model_name=None*, *batch_size=None*, *n_epochs=None*, *target_column=None*)

Bases: *ForeTiS.model._torch_model.TorchModel*

Implementation of a class for a bayesian Long Short-Term Memory (LSTM) network.

See *BaseModel* and *TorchModel* for more information on the attributes.

Parameters

- **optuna_trial** (*optuna.trial.Trial*) –
- **datasets** (*list*) –
- **featureset_name** (*str*) –
- **optimize_featureset** (*bool*) –
- **pca_transform** (*bool*) –
- **current_model_name** (*str*) –
- **batch_size** (*int*) –
- **n_epochs** (*int*) –
- **target_column** (*str*) –

define_model()

Definition of a bayesian LSTM network.

Architecture:

- Bayesian LSTM, Dropout, Linear
- Bayesian Linear output layer

Number of output channels of the first layer, dropout rate, frequency of a doubling of the output channels and number of units in the first linear layer. may be fixed or optimized.

Return type
torch.nn.Sequential

define_hyperparams_to_tune()

See *BaseModel* for more information on the format.

See *TorchModel* for more information on hyperparameters common for all torch models.

Return type
dict

`train_val_loader`(*train*, *val*)

Get the Dataloader with training and validation data

Poram train
training data

Parameters

- **val** (*pandas.DataFrame*) – validation data
- **train** (*pandas.DataFrame*) –

Returns
train_loader, val_loader, val

`predict`(*X_in*)

Implementation of a prediction based on input features for the bayes lstm model. See *BaseModel* for more information

Parameters
X_in (*pandas.DataFrame*) –

Return type
numpy.array

`get_dataloader`(*X*, *y=None*, *only_transform=None*, *predict=False*, *shuffle=False*)

Get a Pytorch DataLoader using the specified data and batch size

Parameters

- **X** (*numpy.array*) – feature matrix to use
- **y** (*numpy.array*) – optional target vector to use
- **only_transform** (*bool*) – whether to only transform or not
- **predict** (*bool*) – weather to use the data for predictions or not
- **shuffle** (*bool*) – shuffle parameter for DataLoader

Returns
Pytorch DataLoader

Return type
torch.utils.data.DataLoader

`create_sequences`(*X*, *y*)

Create sequenced data according to self.seq_length

Returns
sequenced data and labels

Parameters

- **X** (*numpy.array*) –
- **y** (*numpy.array*) –

Return type
tuple

ForeTiS.model._additionalmodels.mlpbayes_intel

Module Contents

Classes

Mlp	Implementation of a class for a bayesian feedforward Multilayer Perceptron (MLP).

class ForeTiS.model._additionalmodels.mlpbayes_intel.**Mlp**(*optuna_trial*, *datasets*, *featureset_name*, *optimize_featureset*, *pca_transform=None*, *current_model_name=None*, *batch_size=None*, *n_epochs=None*, *target_column=None*)

Bases: *ForeTiS.model._torch_model.TorchModel*

Implementation of a class for a bayesian feedforward Multilayer Perceptron (MLP).

See *BaseModel* and *TorchModel* for more information on the attributes.

Parameters

- **optuna_trial** (*optuna.trial.Trial*) –
- **datasets** (*list*) –
- **featureset_name** (*str*) –
- **optimize_featureset** (*bool*) –
- **pca_transform** (*bool*) –
- **current_model_name** (*str*) –
- **batch_size** (*int*) –
- **n_epochs** (*int*) –
- **target_column** (*str*) –

define_model()

Definition of an MLP network.

Architecture:

- N_LAYERS of (bayesian Linear (+ ActivationFunction) (+ BatchNorm) + Dropout)
- Bayesian Linear output layer
- Dropout layer

Number of units in the first bayesian linear layer and percentage decrease after each may be fixed or optimized.

Return type
torch.nn.Sequential

define_hyperparams_to_tune()

See *BaseModel* for more information on the format.

See *TorchModel* for more information on hyperparameters common for all torch models.

Return type
dict

predict(*X_in*)

Implementation of a prediction based on input features for PyTorch models. See *BaseModel* for more information

Parameters
X_in (*pandas.DataFrame*) –

Return type
numpy.array

Submodules

ForeTiS.model._base_model

Module Contents

Classes

BaseModel	BaseModel parent class for all models that can be used within the framework.

class ForeTiS.model._base_model.**BaseModel**(*optuna_trial*, *datasets*, *featureset_name*, *pca_transform*, *target_column*, *optimize_featureset*)

Bases: abc.ABC

BaseModel parent class for all models that can be used within the framework.

Every model must be based on *BaseModel* directly or BaseModel's child classes, e.g. *SklearnModel* or *TorchModel*

** Attributes **

- Instance attributes *
- optuna_trial (*optuna.trial.Trial*): trial of optuna for optimization
- datasets (*list<pd.DataFrame>*): all datasets that are available
- n_outputs (*int*): number of outputs of the prediction model
- all_hyperparams (*dict*): dictionary with all hyperparameters with related info that can be tuned (structure see *define_hyperparams_to_tune*)
- dataset (*pd.DataFrame*): the dataset for this optimization trial
- model: model object
- target_column: the target column for the prediction

- pca_transform: whether conducting pca transformation should be a hyperparameter to optimize or not
- featureset: the recent featureset

Parameters

- **optuna_trial** (*optuna.trial.Trial*) – Trial of optuna for optimization
- **datasets** (*list*) – all datasets that are available
- **featureset_name** (*str*) – the name of the recent feature set
- **target_column** (*str*) – the target column for the prediction
- **pca_transform** (*bool*) – whether conducting pca transformation should be a hyperparameter to optimize or not
- **optimize_featureset** (*bool*) – whether the feature set should be optimized or not

abstract define_model()

Method that defines the model that needs to be optimized. Hyperparams to tune have to be specified in all_hyperparams and suggested via suggest_hyperparam_to_optuna(). The hyperparameters have to be included directly in the model definiton to be optimized. e.g. if you want to optimize the number of layers, do something like

```
n_layers = self.suggest_hyperparam_to_optuna('n_layers') # same name in␣
↪define_hyperparams_to_tune()
for layer in n_layers:
    do something
```

Then the number of layers will be optimized by optuna.

abstract define_hyperparams_to_tune()

Method that defines the hyperparameters that should be tuned during optimization and their ranges. Required format is a dictionary with:

```
{
    'name_hyperparam_1':
        {
        # MANDATORY ITEMS
        'datatype': 'float' | 'int' | 'categorical',
        FOR DATATYPE 'categorical':
            'list_of_values': []  # List of all possible values
        FOR DATATYPE ['float', 'int']:
            'lower_bound': value_lower_bound,
            'upper_bound': value_upper_bound,
            # OPTIONAL ITEMS (only for ['float', 'int']):
            'log': True | False  # sample value from log domain or not
            'step': step_size # step of discretization.
                              # Caution: cannot be combined with log=True
                                            # - in case of 'float' in␣
↪general and
                                            # - for step!=1 in case of 'int'
        },
    'name_hyperparam_2':
        {
```

(continues on next page)

(continued from previous page)

```
        ...
        },
    ...
    'name_hyperparam_k':
        {
        ...
        }
}
```

If you want to use a similar hyperparameter multiple times (e.g. Dropout after several layers), you only need to specify the hyperparameter once. Individual parameters for every suggestion will be created.

Return type
dict

abstract retrain(*retrain*)

Method that runs the retraining of the model

Parameters
retrain (*pandas.DataFrame*) – data for retraining

abstract update(*update*, *period*)

Method that runs the updating of the model

Parameters

- **update** (*pandas.DataFrame*) – data for updating
- **period** (*int*) –

abstract predict(*X_in*)

Method that predicts target values based on the input X_in

Parameters
X_in (*pandas.DataFrame*) – feature matrix as input

Returns
numpy array with the predicted values

Return type
numpy.array

abstract train_val_loop(*train*, *val*)

Method that runs the whole training and validation loop

Parameters

- **train** (*pandas.DataFrame*) – data for the training
- **val** (*pandas.DataFrame*) – data for validation

Returns
predictions on validation set

Return type
numpy.array

suggest_hyperparam_to_optuna(*hyperparam_name*)

Suggest a hyperparameter of hyperparam_dict to the optuna trial to optimize it.

If you want to add a parameter to your model / in your pipeline to be optimized, you need to call this method

Parameters
hyperparam_name (*str*) – name of the hyperparameter to be tuned (see *define_hyperparams_to_tune*)

Returns
suggested value

suggest_all_hyperparams_to_optuna()

Some models accept a dictionary with the model parameters. This method suggests all hyperparameters in all_hyperparams and gives back a dictionary containing them.

Returns
dictionary with suggested hyperparameters

Return type
dict

featureset_hyperparam()

Method that defines the feature set hyperparameter that should be tuned during optimization and its ranges.

pca_transform()

Method that defines the pca transform hyperparameter that should be tuned during optimization and its ranges.

save_model(*path*, *filename*)

Persist the whole model object on a hard drive (can be loaded with *load_model*)

Parameters

- **path** (*str*) – path where the model will be saved
- **filename** (*str*) – filename of the model

ForeTiS.model._baseline_model

Module Contents

Classes

BaselineModel	Parent class based on BaseModel for all baseline models to share functionalities

class ForeTiS.model._baseline_model.**BaselineModel**(*optuna_trial*, *datasets*, *featureset_name*, *pca_transform*, *target_column*, *optimize_featureset*)

Bases: *ForeTiS.model._base_model.BaseModel*, abc.ABC

Parent class based on BaseModel for all baseline models to share functionalities See *BaseModel* for more information.

Parameters

- **optuna_trial** (*optuna.trial.Trial*) –
- **datasets** (*list*) –
- **featureset_name** (*str*) –

- **pca_transform** (*bool*) –
- **target_column** (*str*) –
- **optimize_featureset** (*bool*) –

retrain(*retrain*)

Implementation of the retraining for baseline models. See *BaseModel* for more information.

Parameters
retrain (*pandas.DataFrame*) –

update(*update*, *period*)

Implementation of the retraining for baseline models. See *BaseModel* for more information.

Parameters

- **update** (*pandas.DataFrame*) –
- **period** (*int*) –

predict(*X_in*)

Implementation of a prediction based on input features for baseline models. See *BaseModel* for more information.

Parameters
X_in (*pandas.DataFrame*) –

Return type
numpy.array

train_val_loop(*train*, *val*)

Implementation of a train and validation loop for baseline models. See *BaseModel* for more information.

Parameters

- **train** (*pandas.DataFrame*) –
- **val** (*pandas.DataFrame*) –

Return type
numpy.array

ForeTiS.model._model_classes

Module Contents

Classes

PrintLayer	Base class for all neural network modules.
GetOutputZero	Base class for all neural network modules.
PrepareForlstm	Base class for all neural network modules.
PrepareForDropout	Base class for all neural network modules.
SafeMatern52	The Matern 5/2 kernel. Functions drawn from a GP with this kernel are twice

class ForeTiS.model._model_classes.**PrintLayer**

Bases: torch.nn.Module

Base class for all neural network modules.

Your models should also subclass this class.

Modules can also contain other Modules, allowing to nest them in a tree structure. You can assign the submodules as regular attributes:

```
import torch.nn as nn
import torch.nn.functional as F

class Model(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(1, 20, 5)
        self.conv2 = nn.Conv2d(20, 20, 5)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        return F.relu(self.conv2(x))
```

Submodules assigned in this way will be registered, and will have their parameters converted too when you call to(), etc.

Note: As per the example above, an __init__() call to the parent class must be made before assignment on the child.

Variables

training (*bool*) – Boolean represents whether this module is in training or evaluation mode.

forward(*x*)

class ForeTiS.model._model_classes.**GetOutputZero**

Bases: torch.nn.Module

Base class for all neural network modules.

Your models should also subclass this class.

Modules can also contain other Modules, allowing to nest them in a tree structure. You can assign the submodules as regular attributes:

```
import torch.nn as nn
import torch.nn.functional as F

class Model(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(1, 20, 5)
        self.conv2 = nn.Conv2d(20, 20, 5)

    def forward(self, x):
```

(continues on next page)

(continued from previous page)

```
        x = F.relu(self.conv1(x))
        return F.relu(self.conv2(x))
```

Submodules assigned in this way will be registered, and will have their parameters converted too when you call `to()`, etc.

Note: As per the example above, an `__init__()` call to the parent class must be made before assignment on the child.

Variables
training (*bool*) – Boolean represents whether this module is in training or evaluation mode.

forward(*x*)

class ForeTiS.model._model_classes.**PrepareForlstm**

Bases: `torch.nn.Module`

Base class for all neural network modules.

Your models should also subclass this class.

Modules can also contain other Modules, allowing to nest them in a tree structure. You can assign the submodules as regular attributes:

```
import torch.nn as nn
import torch.nn.functional as F

class Model(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(1, 20, 5)
        self.conv2 = nn.Conv2d(20, 20, 5)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        return F.relu(self.conv2(x))
```

Submodules assigned in this way will be registered, and will have their parameters converted too when you call `to()`, etc.

Note: As per the example above, an `__init__()` call to the parent class must be made before assignment on the child.

Variables
training (*bool*) – Boolean represents whether this module is in training or evaluation mode.

forward(*x*)

class ForeTiS.model._model_classes.**PrepareForDropout**

Bases: `torch.nn.Module`

Base class for all neural network modules.

Your models should also subclass this class.

Modules can also contain other Modules, allowing to nest them in a tree structure. You can assign the submodules as regular attributes:

```
import torch.nn as nn
import torch.nn.functional as F

class Model(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(1, 20, 5)
        self.conv2 = nn.Conv2d(20, 20, 5)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        return F.relu(self.conv2(x))
```

Submodules assigned in this way will be registered, and will have their parameters converted too when you call `to()`, etc.

Note: As per the example above, an `__init__()` call to the parent class must be made before assignment on the child.

Variables

training (*bool*) – Boolean represents whether this module is in training or evaluation mode.

forward(*lstm_out*)

class ForeTiS.model._model_classes.**SafeMatern52**(*variance=1.0*, *lengthscales=1.0*, ***kwargs*)

Bases: gpflow.kernels.Matern52

The Matern 5/2 kernel. Functions drawn from a GP with this kernel are twice differentiable. The kernel equation is

k(r) = ^{2} (1 + 5r + 5/3r^2) exp{-5 r}

where: r is the Euclidean distance between the input points, scaled by the lengthscales parameter , ^{2} is the variance parameter.

Parameters

- **variance** (*gpflow.base.TensorType*) –
- **lengthscales** (*gpflow.base.TensorType*) –
- **kwargs** (*Any*) –

euclid_dist(*X*, *X2*)

ForeTiS.model._model_functions

Module Contents

Functions

load_model(path, filename)	Load persisted model
retrain_model_with_results_file(featureset, model)	Retrain a model based on information saved in a results overview file.

ForeTiS.model._model_functions.**load_model**(*path*, *filename*)

Load persisted model :param path: path where the model is saved :param filename: filename of the model :return: model instance

Parameters

- **path** (*str*) –
- **filename** (*str*) –

Return type
ForeTiS.model._base_model.BaseModel

ForeTiS.model._model_functions.**retrain_model_with_results_file**(*featureset*, *model*)

Retrain a model based on information saved in a results overview file.

Parameters

- **featureset** (ForeTiS.preprocess.base_dataset.Dataset) – dataset for training
- **model** – model that you want to retrain on the whole data

Returns
retrained model

Return type
ForeTiS.model._base_model.BaseModel

ForeTiS.model._sklearn_model

Module Contents

Classes

SklearnModel	Parent class based on BaseModel for all models with a sklearn-like API to share

class ForeTiS.model._sklearn_model.**SklearnModel**(*optuna_trial*, *datasets*, *featureset_name*, *pca_transform*, *target_column*, *optimize_featureset*)

Bases: *ForeTiS.model._base_model.BaseModel*, abc.ABC

Parent class based on *BaseModel* for all models with a sklearn-like API to share functionalities. See *BaseModel* for more information.

Attributes

Inherited attributes

See *BaseModel*

Parameters

- **optuna_trial** (*optuna.trial.Trial*) –
- **datasets** (*list*) –
- **featureset_name** (*str*) –
- **pca_transform** (*bool*) –
- **target_column** (*str*) –
- **optimize_featureset** (*bool*) –

retrain(*retrain*)

Implementation of the retraining for models with sklearn-like API. See *BaseModel* for more information.

Parameters
retrain (*pandas.DataFrame*) –

update(*update*, *period*)

Implementation of the retraining for models with sklearn-like API. See *BaseModel* for more information

Parameters

- **update** (*pandas.DataFrame*) –
- **period** (*int*) –

predict(*X_in*)

Implementation of a prediction based on input features for models with sklearn-like API. See *BaseModel* for more information.

Parameters
X_in (*pandas.DataFrame*) –

Return type
numpy.array

train_val_loop(*train*, *val*)

Implementation of a train and validation loop for models with sklearn-like API. See *BaseModel* for more information.

Parameters

- **train** (*pandas.DataFrame*) –
- **val** (*pandas.DataFrame*) –

Return type
numpy.array

ForeTiS.model._stat_model

Module Contents

Classes

StatModel	Parent class based on BaseModel for all models with a statsmodels-like API to share functionalities.

class ForeTiS.model._stat_model.**StatModel**(*optuna_trial*, *datasets*, *featureset_name*, *optimize_featureset*, *current_model_name=None*, *target_column=None*, *pca_transform=None*)

Bases: ForeTiS.model._base_model.BaseModel, abc.ABC

Parent class based on BaseModel for all models with a statsmodels-like API to share functionalities. See BaseModel for more information.

Parameters

- **optuna_trial** (*optuna.trial.Trial*) –
- **datasets** (*list*) –
- **featureset_name** (*str*) –
- **optimize_featureset** (*bool*) –
- **current_model_name** (*str*) –
- **target_column** (*str*) –
- **pca_transform** (*bool*) –

retrain(*retrain*)

Implementation of the retraining for models with statsmodels-like API. See BaseModel for more information.

Parameters

retrain (*pandas.DataFrame*) –

update(*update*, *period*)

Update existing model due to new samples. See BaseModel for more information.

Parameters

- **update** (*pandas.DataFrame*) –
- **period** (*int*) –

predict(*X_in*)

Implementation of a prediction based on input features for models with statsmodels-like API. See BaseModel for more information.

Parameters

X_in (*pandas.DataFrame*) –

Return type

numpy.array

`train_val_loop`(*train*, *val*)

Implementation of a train and validation loop for models with statsmodels-like API. See *BaseModel* for more information.

Parameters

- **train** (*pandas.DataFrame*) –
- **val** (*pandas.DataFrame*) –

Return type
numpy.array

`get_transformed_set`(*df*, *target_column*, *transf*, *power_transformer*, *only_transform=False*)

Function returning dataset with (log or power) transformed column

Parameters

- **df** (*pandas.DataFrame*) – dataset to transform
- **target_column** (*str*) – column to transform
- **transf** (*str*) – type of transformation
- **power_transformer** (*sklearn.preprocessing.PowerTransformer*) – if power transforming was applied, the used transformer
- **only_transform** – whether to only transform or not

Returns
dataset with transformed column

Return type
pandas.DataFrame

`get_inverse_transformed_set`(*y*, *transf*, *power_transformer*, *is_conf=False*)

Function returning inverse (log or power) transformed column

Parameters

- **y** (*numpy.array*) – array to be inverse transformed
- **power_transformer** – if power transforming was applied, the used transformer
- **transf** (*str*) – type of transformation
- **is_conf** (*bool*) –

Returns
transformed column

Return type
numpy.array

`static common_hyperparams`()

Add hyperparameters that are common for PyTorch models. Do not need to be included in optimization for every child model. See *BaseModel* for more information.

ForeTiS.model._template_sklearn_model

Module Contents

Classes

TemplateSklearnModel	Template file for a prediction model based on *SklearnModel*

class ForeTiS.model._template_sklearn_model.**TemplateSklearnModel**(*optuna_trial*, *datasets*, *featureset_name*, *pca_transform*, *target_column*, *optimize_featureset*)

Bases: *ForeTiS.model._sklearn_model.SklearnModel*

Template file for a prediction model based on *SklearnModel*

See *BaseModel* for more information on the attributes.

Steps you have to do to add your own model:

1. Copy this template file and rename it according to your model (will be the name to call it later on on the command line)
2. Rename the class and add it to *ForeTiS.model.__init__.py*
3. Adjust the class attributes if necessary
4. Define your model in *define_model()*
5. Define the hyperparameters and ranges you want to use for optimization in *define_hyperparams_to_tune()*
6. Test your new prediction model using toy data

Parameters

- **optuna_trial** (*optuna.trial.Trial*) –
- **datasets** (*list*) –
- **featureset_name** (*str*) –
- **pca_transform** (*bool*) –
- **target_column** (*str*) –
- **optimize_featureset** (*bool*) –

define_model()

Definition of the actual prediction model.

Use *param = self.suggest_hyperparam_to_optuna(PARAM_NAME_IN_DEFINE_HYPERPARAMS_TO_TUNE)* if you want to use the value of a hyperparameter that should be optimized. The function needs to return the model object.

See *BaseModel* for more information.

define_hyperparams_to_tune()

Define the hyperparameters and ranges you want to optimize. Caution: they will only be optimized if you add them via *self.suggest_hyperparam_to_optuna(PARAM_NAME)* in *define_model()*

See *BaseModel* for more information on the format and options.

Return type
dict

ForeTiS.model._template_stat_model

Module Contents

Classes

TemplateStatModel	Template file for a prediction model based on *StatModel*

class ForeTiS.model._template_stat_model.**TemplateStatModel**(*optuna_trial*, *datasets*, *featureset_name*, *optimize_featureset*, *current_model_name=None*, *target_column=None*, *pca_transform=None*)

Bases: *ForeTiS.model._stat_model.StatModel*

Template file for a prediction model based on *StatModel*

See *BaseModel* and *StatModel* for more information on the attributes.

Steps you have to do to add your own model:

1. Copy this template file and rename it according to your model (will be the name to call it later on the command line)
2. Rename the class and add it to *ForeTiS.model.__init__.py*
3. Adjust the class attributes if necessary
4. Define your model in *define_model()*
5. Define the hyperparameters and ranges you want to use for optimization in *define_hyperparams_to_tune()*.

   CAUTION: Some hyperparameters are already defined in *common_hyperparams()*, which you can directly use here. Some of them are already suggested in *StatModel*.
6. Test your new prediction model using toy data

Parameters

- **optuna_trial** (*optuna.trial.Trial*) –
- **datasets** (*list*) –
- **featureset_name** (*str*) –
- **optimize_featureset** (*bool*) –
- **current_model_name** (*str*) –

- **target_column** (*str*) –
- **pca_transform** (*bool*) –

define_model()

Definition of the actual prediction model.

Use *param = self.suggest_hyperparam_to_optuna(PARAM_NAME_IN_DEFINE_HYPERPARAMS_TO_TUNE)* if you want to use the value of a hyperparameter that should be optimized. The function needs to return the model object.

See *BaseModel* for more information.

Return type
pmdarima.ARIMA

define_hyperparams_to_tune()

Define the hyperparameters and ranges you want to optimize. Caution: they will only be optimized if you add them via *self.suggest_hyperparam_to_optuna(PARAM_NAME)* in *define_model()*

See *BaseModel* for more information on the format and options.

Check *TorchModel* for already defined (and for some cases also suggested) hyperparameters.

Return type
dict

ForeTiS.model._template_tensorflow_model

Module Contents

Classes

TemplateTensorflowModel	Template file for a prediction model based on *TensorflowModel*

class ForeTiS.model._template_tensorflow_model.**TemplateTensorflowModel**(*optuna_trial*, *datasets*, *featureset_name*, *optimize_featureset*, *pca_transform=None*, *target_column=None*)

Bases: *ForeTiS.model._tensorflow_model.TensorflowModel*

Template file for a prediction model based on *TensorflowModel*

See *BaseModel* and *TensorflowModel* for more information on the attributes.

Steps you have to do to add your own model:

1. Copy this template file and rename it according to your model (will be the name to call it later on the command line)
2. Rename the class and add it to *ForeTiS.model.__init__.py*
3. Adjust the class attributes if necessary
4. Define your model in *define_model()*

5. Define the hyperparameters and ranges you want to use for optimization in *define_hyperparams_to_tune()*.

   CAUTION: Some hyperparameters are already defined in *common_hyperparams()*, which you can directly use here. Some of them are already suggested in `TensorflowModel`.

6. Test your new prediction model using toy data

   Parameters

   - **optuna_trial** (*optuna.trial.Trial*) –
   - **datasets** (*list*) –
   - **featureset_name** (*str*) –
   - **optimize_featureset** (*bool*) –
   - **pca_transform** (*bool*) –
   - **target_column** (*str*) –

ForeTiS.model._template_torch_model

Module Contents

Classes

TemplateTorchModel	Template file for a prediction model based on *TorchModel*

class ForeTiS.model._template_torch_model.**TemplateTorchModel**(*optuna_trial*, *datasets*, *featureset_name*, *optimize_featureset*, *pca_transform=None*, *current_model_name=None*, *batch_size=None*, *n_epochs=None*, *target_column=None*)

Bases: *ForeTiS.model._torch_model.TorchModel*

Template file for a prediction model based on *TorchModel*

See *BaseModel* and *TorchModel* for more information on the attributes.

Steps you have to do to add your own model:

1. Copy this template file and rename it according to your model (will be the name to call it later on the command line)
2. Rename the class and add it to *ForeTiS.model.__init__.py*
3. Adjust the class attributes if necessary
4. Define your model in *define_model()*
5. Define the hyperparameters and ranges you want to use for optimization in *define_hyperparams_to_tune()*.

   CAUTION: Some hyperparameters are already defined in *common_hyperparams()*, which you can directly use here. Some of them are already suggested in *TorchModel*.

6. Test your new prediction model using toy data

Parameters

- **optuna_trial** (*optuna.trial.Trial*) –
- **datasets** (*list*) –
- **featureset_name** (*str*) –
- **optimize_featureset** (*bool*) –
- **pca_transform** (*bool*) –
- **current_model_name** (*str*) –
- **batch_size** (*int*) –
- **n_epochs** (*int*) –
- **target_column** (*str*) –

define_model()

Definition of the actual prediction model.

Use *param = self.suggest_hyperparam_to_optuna(PARAM_NAME_IN_DEFINE_HYPERPARAMS_TO_TUNE)* if you want to use the value of a hyperparameter that should be optimized. The function needs to return the model object.

See *BaseModel* for more information.

define_hyperparams_to_tune()

Define the hyperparameters and ranges you want to optimize. Caution: they will only be optimized if you add them via *self.suggest_hyperparam_to_optuna(PARAM_NAME)* in *define_model()*

See *BaseModel* for more information on the format and options.

Check *TorchModel* for already defined (and for some cases also suggested) hyperparameters.

Return type
dict

ForeTiS.model._tensorflow_model

Module Contents

Classes

TensorflowModel	Parent class based on *BaseModel* for all TensorFlow models to share functionalities.

class ForeTiS.model._tensorflow_model.**TensorflowModel**(*optuna_trial*, *datasets*, *featureset_name*, *optimize_featureset*, *pca_transform=None*, *target_column=None*)

Bases: *ForeTiS.model._base_model.BaseModel*, abc.ABC

Parent class based on *BaseModel* for all TensorFlow models to share functionalities. See *BaseModel* for more information.

Attributes

Inherited attributes

See BaseModel.

Additional attributes

- x_scaler (*sklearn.preprocessing.StandardScaler*): Standard scaler for the x data
- y_scaler (*sklearn.preprocessing.StandardScaler*): Standard scaler for the y data

Parameters

- **optuna_trial** (*optuna.trial.Trial*) –
- **datasets** (*list*) –
- **featureset_name** (*str*) –
- **optimize_featureset** (*bool*) –
- **pca_transform** (*bool*) –
- **target_column** (*str*) –

define_model()

Definition of the actual prediction model.

See BaseModel for more information.

Return type
gpflow.models.GPR

retrain(*retrain*)

Implementation of the retraining for models with sklearn-like API. See BaseModel for more information

Parameters
retrain (*pandas.DataFrame*) –

update(*update*, *period*)

Implementation of the retraining for models with sklearn-like API. See BaseModel for more information

Parameters

- **update** (*pandas.DataFrame*) –
- **period** (*int*) –

predict(*X_in*)

Implementation of a prediction based on input features for models with sklearn-like API. See BaseModel for more information

Parameters
X_in (*pandas.DataFrame*) –

Return type
numpy.array

train_val_loop(*train*, *val*)

Implementation of a train and validation loop for models with sklearn-like API. See BaseModel for more information

Parameters

- **train** (*pandas.DataFrame*) –
- **val** (*pandas.DataFrame*) –

Return type
numpy.array

extend_kernel_combinations()

Function extending kernels list with combinations based on base_kernels

static common_hyperparams()

See *BaseModel* for more information on the format.

Return type
dict

ForeTiS.model._torch_model

Module Contents

Classes

TorchModel	Parent class based on *BaseModel* for all PyTorch models to share functionalities.

class ForeTiS.model._torch_model.**TorchModel**(*optuna_trial*, *datasets*, *featureset_name*, *optimize_featureset*, *pca_transform=None*, *current_model_name=None*, *batch_size=None*, *n_epochs=None*, *target_column=None*)

Bases: *ForeTiS.model._base_model.BaseModel*, abc.ABC

Parent class based on *BaseModel* for all PyTorch models to share functionalities. See *BaseModel* for more information.

Attributes

Inherited attributes

See *BaseModel*.

Additional attributes

- batch_size (*int*): Batch size for batch-based training
- n_epochs (*int*): Number of epochs for optimization
- num_monte_carlo (*int*): Number of monte carlo iteration for the bayesian neural networks
- optimizer (*torch.optim.optimizer.Optimizer*): optimizer for model fitting
- loss_fn: loss function for model fitting
- early_stopping_patience (*int*): epochs without improvement before early stopping
- early_stopping_point (*int*): epoch at which early stopping occured
- device (*torch.device*): device to use, e.g. GPU
- X_scaler (*sklearn.preprocessing.StandardScaler*): Standard scaler for the X data

Parameters

- **optuna_trial** (*optuna.trial.Trial*) – Trial of optuna for optimization
- **datasets** (*list*) – all datasets that are available
- **current_model_name** (*str*) – name of the current model according to naming of .py file in package model
- **batch_size** (*int*) – batch size for neural network models
- **n_epochs** (*int*) – number of epochs for neural network models
- **target_column** (*str*) – the target column for the prediction
- **featureset_name** (*str*) –
- **optimize_featureset** (*bool*) –
- **pca_transform** (*bool*) –

train_val_loop(*train*, *val*)

Implementation of a train and validation loop for PyTorch models. See *BaseModel* for more information

Parameters

- **train** (*pandas.DataFrame*) –
- **val** (*pandas.DataFrame*) –

Return type
numpy.array

train_val_loader(*train*, *val*)

Get the Dataloader with training and validation data

Poram train
training data

Parameters

- **val** (*pandas.DataFrame*) – validation data
- **train** (*pandas.DataFrame*) –

Returns
train_loader, val_loader, val

train_one_epoch(*train_loader*, *scaler*)

Train one epoch

Parameters
train_loader (*torch.utils.data.DataLoader*) – DataLoader with training data

validate_one_epoch(*val_loader*)

Validate one epoch

Parameters
val_loader (*torch.utils.data.DataLoader*) – DataLoader with validation data

Returns
loss based on loss-criterion

Return type
float

retrain(*retrain*)

Implementation of the retraining for PyTorch models. See *BaseModel* for more information

Parameters
retrain (*pandas.DataFrame*) –

update(*update*, *period*)

Implementation of the retraining for PyTorch models. See *BaseModel* for more information

Parameters

- **update** (*pandas.DataFrame*) –
- **period** (*int*) –

predict(*X_in*)

Implementation of a prediction based on input features for PyTorch models. See *BaseModel* for more information

Parameters
X_in (*pandas.DataFrame*) –

Return type
numpy.array

get_loss(*outputs*, *targets*)

Calculate the loss based on the outputs and targets

Parameters

- **outputs** (*torch.Tensor*) – outputs of the model
- **targets** (*torch.Tensor*) – targets of the dataset

Returns
loss

Return type
torch.Tensor

get_dataloader(*X*, *y=None*, *only_transform=None*, *predict=False*, *shuffle=False*)

Get a Pytorch DataLoader using the specified data and batch size

Parameters

- **X** (*numpy.array*) – feature matrix to use
- **y** (*numpy.array*) – optional target vector to use
- **only_transform** (*bool*) – whether to only transform or not
- **predict** (*bool*) – weather to use the data for predictions or not
- **shuffle** (*bool*) – shuffle parameter for DataLoader

Returns
Pytorch DataLoader

Return type
torch.utils.data.DataLoader

static common_hyperparams()

Add hyperparameters that are common for PyTorch models. Do not need to be included in optimization for every child model. Also See *BaseModel* for more information

static get_torch_object_for_string(*string_to_get*)

Get the torch object for a specific string, e.g. when suggesting to optuna as hyperparameter

Parameters
string_to_get (*str*) – string to retrieve the torch object

Returns
torch object

ForeTiS.model.ard

Module Contents

Classes

ARDRegression	Implementation of a class for ARDRegression.

class ForeTiS.model.ard.**ARDRegression**(*optuna_trial*, *datasets*, *featureset_name*, *pca_transform*, *target_column*, *optimize_featureset*)

Bases: *ForeTiS.model._sklearn_model.SklearnModel*

Implementation of a class for ARDRegression.

See *BaseModel* for more information on the attributes.

Parameters

- **optuna_trial** (*optuna.trial.Trial*) –
- **datasets** (*list*) –
- **featureset_name** (*str*) –
- **pca_transform** (*bool*) –
- **target_column** (*str*) –
- **optimize_featureset** (*bool*) –

define_model()

Definition of the actual prediction model.

See *BaseModel* for more information.

Return type
sklearn.linear_model.ARDRegression

define_hyperparams_to_tune()

See *BaseModel* for more information on the format.

Return type
dict

ForeTiS.model.averagehistorical

Module Contents

Classes

AverageHistorical	Implementation of a class for AverageHistorical.

class ForeTiS.model.averagehistorical.**AverageHistorical**(*optuna_trial*, *datasets*, *featureset_name*, *pca_transform*, *target_column*, *optimize_featureset*)

Bases: *ForeTiS.model._baseline_model.BaselineModel*

Implementation of a class for AverageHistorical.

See *BaseModel* for more information on the attributes.

Parameters

- **optuna_trial** (*optuna.trial.Trial*) –
- **datasets** (*list*) –
- **featureset_name** (*str*) –
- **pca_transform** (*bool*) –
- **target_column** (*str*) –
- **optimize_featureset** (*bool*) –

define_model()

Definition of the actual prediction model.

See *BaseModel* for more information.

define_hyperparams_to_tune()

See *BaseModel* for more information on the format.

Return type
dict

ForeTiS.model.averagemoving

Module Contents

Classes

AverageMoving	Implementation of a class for AverageMoving.

class ForeTiS.model.averagemoving.**AverageMoving**(*optuna_trial*, *datasets*, *featureset_name*, *pca_transform*, *target_column*, *optimize_featureset*)

Bases: *ForeTiS.model._baseline_model.BaselineModel*

Implementation of a class for AverageMoving.

See *BaseModel* for more information on the attributes.

Parameters

- **optuna_trial** (*optuna.trial.Trial*) –
- **datasets** (*list*) –
- **featureset_name** (*str*) –
- **pca_transform** (*bool*) –
- **target_column** (*str*) –
- **optimize_featureset** (*bool*) –

define_model()

Definition of the actual prediction model.

See *BaseModel* for more information.

define_hyperparams_to_tune()

See *BaseModel* for more information on the format.

Return type
dict

ForeTiS.model.averageseasonal

Module Contents

Classes

AverageSeasonal	Implementation of a class for AverageSeasonal.

class ForeTiS.model.averageseasonal.**AverageSeasonal**(*optuna_trial*, *datasets*, *featureset_name*, *pca_transform*, *target_column*, *optimize_featureset*)

Bases: *ForeTiS.model._baseline_model.BaselineModel*

Implementation of a class for AverageSeasonal.

See *BaseModel* for more information on the attributes.

Parameters

- **optuna_trial** (*optuna.trial.Trial*) –
- **datasets** (*list*) –
- **featureset_name** (*str*) –
- **pca_transform** (*bool*) –
- **target_column** (*str*) –

- **optimize_featureset** (*bool*) –

define_model()

Definition of the actual prediction model.

See *BaseModel* for more information.

define_hyperparams_to_tune()

See *BaseModel* for more information on the format.

Return type
dict

ForeTiS.model.averageseasonallag

Module Contents

Classes

AverageSeasonal	Implementation of a class for AverageSeasonal.

class ForeTiS.model.averageseasonallag.**AverageSeasonal**(*optuna_trial*, *datasets*, *featureset_name*, *pca_transform*, *target_column*, *optimize_featureset*)

Bases: *ForeTiS.model._baseline_model.BaselineModel*

Implementation of a class for AverageSeasonal.

See *BaseModel* for more information on the attributes.

Parameters

- **optuna_trial** (*optuna.trial.Trial*) –
- **datasets** (*list*) –
- **featureset_name** (*str*) –
- **pca_transform** (*bool*) –
- **target_column** (*str*) –
- **optimize_featureset** (*bool*) –

define_model()

Definition of the actual prediction model.

See *BaseModel* for more information.

define_hyperparams_to_tune()

See *BaseModel* for more information on the format.

Return type
dict

retrain(*retrain*)

Implementation of the retraining for the AverageSeasonal model. See *BaseModel* for more information.

Parameters
retrain (*pandas.DataFrame*) –

update(*update*, *period*)

Implementation of the retraining for the AverageSeasonal model. See *BaseModel* for more information

Parameters

- **update** (*pandas.DataFrame*) –
- **period** (*int*) –

ForeTiS.model.bayesridge

Module Contents

Classes

BayesianRidge	Implementation of a class for BayesianRidge.

class ForeTiS.model.bayesridge.**BayesianRidge**(*optuna_trial*, *datasets*, *featureset_name*, *pca_transform*, *target_column*, *optimize_featureset*)

Bases: *ForeTiS.model._sklearn_model.SklearnModel*

Implementation of a class for BayesianRidge.

See *BaseModel* for more information on the attributes.

Parameters

- **optuna_trial** (*optuna.trial.Trial*) –
- **datasets** (*list*) –
- **featureset_name** (*str*) –
- **pca_transform** (*bool*) –
- **target_column** (*str*) –
- **optimize_featureset** (*bool*) –

define_model()

Definition of the actual prediction model.

See *BaseModel* for more information.

Return type
sklearn.linear_model.BayesianRidge

define_hyperparams_to_tune()

See *BaseModel* for more information on the format.

Return type
dict

ForeTiS.model.elasticnet

Module Contents

Classes

ElasticNet	Implementation of a class for ElasticNet.

class ForeTiS.model.elasticnet.**ElasticNet**(*optuna_trial*, *datasets*, *featureset_name*, *pca_transform*, *target_column*, *optimize_featureset*)

Bases: *ForeTiS.model._sklearn_model.SklearnModel*

Implementation of a class for ElasticNet.

See *BaseModel* for more information on the attributes.

Parameters

- **optuna_trial** (*optuna.trial.Trial*) –
- **datasets** (*list*) –
- **featureset_name** (*str*) –
- **pca_transform** (*bool*) –
- **target_column** (*str*) –
- **optimize_featureset** (*bool*) –

define_model()

Definition of the actual prediction model.

See *BaseModel* for more information.

Return type
sklearn.linear_model.ElasticNet

define_hyperparams_to_tune()

See *BaseModel* for more information on the format.

Return type
dict

ForeTiS.model.es

Module Contents

Classes

Es	Implementation of a class for an Exponential Smoothing (ES) model.

class ForeTiS.model.es.**Es**(*optuna_trial*, *datasets*, *featureset_name*, *optimize_featureset*, *current_model_name=None*, *target_column=None*, *pca_transform=None*)

Bases: *ForeTiS.model._stat_model.StatModel*

Implementation of a class for an Exponential Smoothing (ES) model. See *BaseModel* for more information on the attributes.

Parameters

- **optuna_trial** (*optuna.trial.Trial*) –
- **datasets** (*list*) –
- **featureset_name** (*str*) –
- **optimize_featureset** (*bool*) –
- **current_model_name** (*str*) –
- **target_column** (*str*) –
- **pca_transform** (*bool*) –

define_model()

Definition of the actual prediction model.

See *BaseModel* for more information.

Return type
statsmodels.tsa.api.ExponentialSmoothing

define_hyperparams_to_tune()

See *BaseModel* for more information on the format.

Return type
dict

ForeTiS.model.gprtf

Module Contents

Classes

Gpr	Implementation of a class for Gpr.

class ForeTiS.model.gprtf.**Gpr**(*optuna_trial*, *datasets*, *featureset_name*, *optimize_featureset*, *pca_transform=None*, *target_column=None*)

Bases: *ForeTiS.model._tensorflow_model.TensorflowModel*

Implementation of a class for Gpr.

See *BaseModel* for more information on the attributes.

Parameters

- **optuna_trial** (*optuna.trial.Trial*) –
- **datasets** (*list*) –
- **featureset_name** (*str*) –

- **optimize_featureset** (*bool*) –
- **pca_transform** (*bool*) –
- **target_column** (*str*) –

ForeTiS.model.lasso

Module Contents

Classes

Lasso	Implementation of a class for Lasso.

class ForeTiS.model.lasso.**Lasso**(*optuna_trial*, *datasets*, *featureset_name*, *pca_transform*, *target_column*, *optimize_featureset*)

Bases: *ForeTiS.model._sklearn_model.SklearnModel*

Implementation of a class for Lasso.

See *BaseModel* for more information on the attributes.

Parameters

- **optuna_trial** (*optuna.trial.Trial*) –
- **datasets** (*list*) –
- **featureset_name** (*str*) –
- **pca_transform** (*bool*) –
- **target_column** (*str*) –
- **optimize_featureset** (*bool*) –

define_model()

Definition of the actual prediction model.

See *BaseModel* for more information.

Return type
sklearn.linear_model.Lasso

define_hyperparams_to_tune()

See *BaseModel* for more information on the format.

Return type
dict

ForeTiS.model.lstm

Module Contents

Classes

LSTM	Implementation of a class for a Long Short-Term Memory (LSTM) network.

class ForeTiS.model.lstm.**LSTM**(*optuna_trial*, *datasets*, *featureset_name*, *optimize_featureset*, *pca_transform=None*, *current_model_name=None*, *batch_size=None*, *n_epochs=None*, *target_column=None*)

Bases: *ForeTiS.model._torch_model.TorchModel*

Implementation of a class for a Long Short-Term Memory (LSTM) network.

See *BaseModel* and *TorchModel* for more information on the attributes.

Parameters

- **optuna_trial** (*optuna.trial.Trial*) –
- **datasets** (*list*) –
- **featureset_name** (*str*) –
- **optimize_featureset** (*bool*) –
- **pca_transform** (*bool*) –
- **current_model_name** (*str*) –
- **batch_size** (*int*) –
- **n_epochs** (*int*) –
- **target_column** (*str*) –

define_model()

Definition of a LSTM network.

Architecture:

- LSTM, Dropout, Linear
- Linear output layer

Number of output channels of the first layer, dropout rate, frequency of a doubling of the output channels and number of units in the first linear layer. may be fixed or optimized.

Return type
torch.nn.Sequential

define_hyperparams_to_tune()

See *BaseModel* for more information on the format.

See *TorchModel* for more information on hyperparameters common for all torch models.

Return type
dict

train_val_loader(*train*, *val*)

Get the Dataloader with training and validation data

Poram train
training data

Parameters

- **val** (*pandas.DataFrame*) – validation data
- **train** (*pandas.DataFrame*) –

Returns
train_loader, val_loader, val

predict(*X_in*)

Implementation of a prediction based on input features for PyTorch models. See *BaseModel* for more information

Parameters
X_in (*pandas.DataFrame*) –

Return type
numpy.array

get_dataloader(*X*, *y=None*, *only_transform=None*, *predict=False*, *shuffle=False*)

Get a Pytorch DataLoader using the specified data and batch size

Parameters

- **X** (*numpy.array*) – feature matrix to use
- **y** (*numpy.array*) – optional target vector to use
- **only_transform** (*bool*) – whether to only transform or not
- **predict** (*bool*) – weather to use the data for predictions or not
- **shuffle** (*bool*) – shuffle parameter for DataLoader

Returns
Pytorch DataLoader

Return type
torch.utils.data.DataLoader

create_sequences(*X*, *y*)

Create sequenced data according to self.seq_length

Returns
sequenced data and labels

Parameters

- **X** (*numpy.array*) –
- **y** (*numpy.array*) –

Return type
tuple

ForeTiS.model.lstmbayes

Module Contents

Classes

LSTMbayes	Implementation of a class for a bayesian Long Short-Term Memory (LSTM) network.

class ForeTiS.model.lstmbayes.**LSTMbayes**(*optuna_trial*, *datasets*, *featureset_name*, *optimize_featureset*, *pca_transform=None*, *current_model_name=None*, *batch_size=None*, *n_epochs=None*, *target_column=None*)

Bases: *ForeTiS.model._torch_model.TorchModel*

Implementation of a class for a bayesian Long Short-Term Memory (LSTM) network.

See *BaseModel* and *TorchModel* for more information on the attributes.

Parameters

- **optuna_trial** (*optuna.trial.Trial*) –
- **datasets** (*list*) –
- **featureset_name** (*str*) –
- **optimize_featureset** (*bool*) –
- **pca_transform** (*bool*) –
- **current_model_name** (*str*) –
- **batch_size** (*int*) –
- **n_epochs** (*int*) –
- **target_column** (*str*) –

define_model()

Definition of a bayesian LSTM network.

Architecture:

- Bayesian LSTM, Dropout, Linear
- Bayesian Linear output layer

Number of output channels of the first layer, dropout rate, frequency of a doubling of the output channels and number of units in the first linear layer. may be fixed or optimized.

Return type
torch.nn.Sequential

define_hyperparams_to_tune()

See *BaseModel* for more information on the format.

See *TorchModel* for more information on hyperparameters common for all torch models.

Return type
dict

train_val_loader(*train*, *val*)

Get the Dataloader with training and validation data

Poram train
training data

Parameters

- **val** (*pandas.DataFrame*) – validation data
- **train** (*pandas.DataFrame*) –

Returns
train_loader, val_loader, val

predict(*X_in*)

Implementation of a prediction based on input features for the bayes lstm model. See *BaseModel* for more information

Parameters
X_in (*pandas.DataFrame*) –

Return type
numpy.array

get_dataloader(*X*, *y=None*, *only_transform=None*, *predict=False*, *shuffle=False*)

Get a Pytorch DataLoader using the specified data and batch size

Parameters

- **X** (*numpy.array*) – feature matrix to use
- **y** (*numpy.array*) – optional target vector to use
- **only_transform** (*bool*) – whether to only transform or not
- **predict** (*bool*) – weather to use the data for predictions or not
- **shuffle** (*bool*) – shuffle parameter for DataLoader

Returns
Pytorch DataLoader

Return type
torch.utils.data.DataLoader

create_sequences(*X*, *y*)

Create sequenced data according to self.seq_length

Returns
sequenced data and labels

Parameters

- **X** (*numpy.array*) –
- **y** (*numpy.array*) –

Return type
tuple

ForeTiS.model.mlp

Module Contents

Classes

Mlp	Implementation of a class for a feedforward Multilayer Perceptron (MLP).

class ForeTiS.model.mlp.**Mlp**(*optuna_trial*, *datasets*, *featureset_name*, *optimize_featureset*, *pca_transform=None*, *current_model_name=None*, *batch_size=None*, *n_epochs=None*, *target_column=None*)

Bases: ForeTiS.model._torch_model.TorchModel

Implementation of a class for a feedforward Multilayer Perceptron (MLP).

See BaseModel and TorchModel for more information on the attributes.

Parameters

- **optuna_trial** (*optuna.trial.Trial*) –
- **datasets** (*list*) –
- **featureset_name** (*str*) –
- **optimize_featureset** (*bool*) –
- **pca_transform** (*bool*) –
- **current_model_name** (*str*) –
- **batch_size** (*int*) –
- **n_epochs** (*int*) –
- **target_column** (*str*) –

define_model()

Definition of an MLP network.

Architecture:

- N_LAYERS of (Linear (+ ActivationFunction) (+ BatchNorm) + Dropout)
- Linear output layer
- Dropout layer

Number of units in the first linear layer and percentage decrease after each may be fixed or optimized.

Return type
torch.nn.Sequential

define_hyperparams_to_tune()

See BaseModel for more information on the format.

See TorchModel for more information on hyperparameters common for all torch models.

Return type
dict

ForeTiS.model.mlpbayes

Module Contents

Classes

Mlpbayes	Implementation of a class for a bayesian feedforward Multilayer Perceptron (MLP).

class ForeTiS.model.mlpbayes.**Mlpbayes**(*optuna_trial*, *datasets*, *featureset_name*, *optimize_featureset*, *pca_transform=None*, *current_model_name=None*, *batch_size=None*, *n_epochs=None*, *target_column=None*)

Bases: *ForeTiS.model._torch_model.TorchModel*

Implementation of a class for a bayesian feedforward Multilayer Perceptron (MLP).

See *BaseModel* and *TorchModel* for more information on the attributes.

Parameters

- **optuna_trial** (*optuna.trial.Trial*) –
- **datasets** (*list*) –
- **featureset_name** (*str*) –
- **optimize_featureset** (*bool*) –
- **pca_transform** (*bool*) –
- **current_model_name** (*str*) –
- **batch_size** (*int*) –
- **n_epochs** (*int*) –
- **target_column** (*str*) –

define_model()

Definition of an MLP network.

Architecture:

- N_LAYERS of (bayesian Linear (+ ActivationFunction) (+ BatchNorm) + Dropout)
- Bayesian Linear output layer
- Dropout layer

Number of units in the first bayesian linear layer and percentage decrease after each may be fixed or optimized.

Return type
torch.nn.Sequential

define_hyperparams_to_tune()

See *BaseModel* for more information on the format.

See *TorchModel* for more information on hyperparameters common for all torch models.

Return type
dict

predict(*X_in*)

Implementation of a prediction based on input features for PyTorch models. See *BaseModel* for more information

Parameters
X_in (*pandas.DataFrame*) –

Return type
numpy.array

ForeTiS.model.ridge

Module Contents

Classes

Ridge	Implementation of a class for Ridge.

class ForeTiS.model.ridge.**Ridge**(*optuna_trial*, *datasets*, *featureset_name*, *pca_transform*, *target_column*, *optimize_featureset*)

Bases: *ForeTiS.model._sklearn_model.SklearnModel*

Implementation of a class for Ridge.

See *BaseModel* for more information on the attributes.

Parameters

- **optuna_trial** (*optuna.trial.Trial*) –
- **datasets** (*list*) –
- **featureset_name** (*str*) –
- **pca_transform** (*bool*) –
- **target_column** (*str*) –
- **optimize_featureset** (*bool*) –

define_model()

Definition of the actual prediction model.

See *BaseModel* for more information.

Return type
sklearn.linear_model.Ridge

define_hyperparams_to_tune()

See *BaseModel* for more information on the format.

Return type
dict

ForeTiS.model.sarima

Module Contents

Classes

Arima	Implementation of a class for a (Seasonal) Autoregressive Integrated Moving Average ((S)ARIMA) model.

class ForeTiS.model.sarima.**Arima**(*optuna_trial*, *datasets*, *featureset_name*, *optimize_featureset*, *current_model_name=None*, *target_column=None*, *pca_transform=None*)

Bases: *ForeTiS.model._stat_model.StatModel*

Implementation of a class for a (Seasonal) Autoregressive Integrated Moving Average ((S)ARIMA) model. See *BaseModel* for more information on the attributes.

Parameters

- **optuna_trial** (*optuna.trial.Trial*) –
- **datasets** (*list*) –
- **featureset_name** (*str*) –
- **optimize_featureset** (*bool*) –
- **current_model_name** (*str*) –
- **target_column** (*str*) –
- **pca_transform** (*bool*) –

define_model()

Definition of the actual prediction model.

See *BaseModel* for more information.

Return type
pmdarima.ARIMA

define_hyperparams_to_tune()

See *BaseModel* for more information on the format.

Return type
dict

ForeTiS.model.sarimax

Module Contents

Classes

Arima	Implementation of a class for a (Seasonal) Autoregressive Integrated Moving Average with eXogenous factors ((S)ARIMAX) model.

class ForeTiS.model.sarimax.**Arima**(*optuna_trial*, *datasets*, *featureset_name*, *optimize_featureset*, *current_model_name=None*, *target_column=None*, *pca_transform=None*)

Bases: *ForeTiS.model._stat_model.StatModel*

Implementation of a class for a (Seasonal) Autoregressive Integrated Moving Average with eXogenous factors ((S)ARIMAX) model. See *BaseModel* for more information on the attributes.

Parameters

- **optuna_trial** (*optuna.trial.Trial*) –
- **datasets** (*list*) –
- **featureset_name** (*str*) –
- **optimize_featureset** (*bool*) –
- **current_model_name** (*str*) –
- **target_column** (*str*) –
- **pca_transform** (*bool*) –

define_model()

Definition of the actual prediction model.

See *BaseModel* for more information.

Return type
pmdarima.ARIMA

define_hyperparams_to_tune()

See *BaseModel* for more information on the format.

Return type
dict

ForeTiS.model.xgboost

Module Contents

Classes

XgBoost	Implementation of a class for XGBoost.

class ForeTiS.model.xgboost.**XgBoost**(*optuna_trial*, *datasets*, *featureset_name*, *pca_transform*, *target_column*, *optimize_featureset*)

Bases: *ForeTiS.model._sklearn_model.SklearnModel*

Implementation of a class for XGBoost.

See *BaseModel* for more information on the attributes.

Parameters

- **optuna_trial** (*optuna.trial.Trial*) –
- **datasets** (*list*) –
- **featureset_name** (*str*) –
- **pca_transform** (*bool*) –
- **target_column** (*str*) –
- **optimize_featureset** (*bool*) –

define_model()

Definition of the actual prediction model.

See *BaseModel* for more information.

Return type
xgboost.XGBModel

define_hyperparams_to_tune()

See *BaseModel* for more information on the format.

Return type
dict

ForeTiS.optimization

Submodules

ForeTiS.optimization.optuna_optim

Module Contents

Classes

OptunaOptim	Class that contains all info for the whole optimization using optuna for one model and dataset.

class ForeTiS.optimization.optuna_optim.**OptunaOptim**(*save_dir*, *data*, *config_file_section*, *featureset_name*, *datasplit*, *test_set_size_percentage*, *val_set_size_percentage*, *n_trials*, *save_final_model*, *batch_size*, *n_epochs*, *current_model_name*, *datasets*, *periodical_refit_frequency*, *refit_drops*, *refit_window*, *intermediate_results_interval*, *pca_transform*, *config*, *optimize_featureset*, *scale_thr*, *scale_seasons*, *scale_window_factor*, *cf_r*, *cf_order*, *cf_smooth*, *cf_thr_perc*, *scale_window_minimum*, *max_samples_factor*, *valtest_seasons*, *seasonal_valtest*, *n_splits*, *config_model_featureset*)

Class that contains all info for the whole optimization using optuna for one model and dataset.

** Attributes **

- study (*optuna.study.Study*): optuna study for optimization run
- current_best_val_result (*float*): the best validation result so far
- early_stopping_point (*int*): point at which early stopping occured (relevant for some models)
- seasonal_periods (*int*): number of samples in one season of the used dataset
- target_column (*str*): target column for which predictions shall be made
- best_trials (*list*): list containing the numbers of the best trials
- user_input_params (*dict*): all params handed over to the constructor that are needed in the whole class
- base_path (*str*): base_path for save_path
- save_path (*str*): path for model and results storing

Parameters

- **save_dir** (*pathlib.Path*) – directory for saving the results
- **data** (*str*) – the dataset that you want to use
- **config_file_section** (*str*) – the section of the config file for the used dataset
- **featureset_name** (*str*) – name of the feature set used
- **datasplit** (*str*) – the used datasplit method, either 'timeseries-cv', 'train-val-test', 'cv'
- **test_set_size_percentage** (*int*) – size of the test set relevant for cv-test and train-val-test
- **val_set_size_percentage** (*int*) – size of the validation set relevant for train-val-test
- **n_trials** (*int*) – number of trials for optuna
- **save_final_model** (*bool*) – specify if the final model should be saved
- **batch_size** (*int*) – batch size for neural network models
- **n_epochs** (*int*) – number of epochs for neural network models
- **current_model_name** (*str*) – name of the current model according to naming of .py file in package model

- **datasets** (ForeTiS.preprocess.base_dataset.Dataset) – the Dataset class containing the feature sets
- **periodical_refit_frequency** (*list*) – if and for which intervals periodical refitting should be performed
- **refit_drops** (*int*) – after how many periods the model should get updated
- **refit_window** (*int*) – seasons get used for refitting
- **intermediate_results_interval** (*int*) – number of trials after which intermediate results will be saved
- **pca_transform** (*bool*) – whether pca dimensionality reduction will be optimized or not
- **config** (*configparser.RawConfigParser*) – the information from dataset_specific_config.ini
- **optimize_featureset** (*bool*) – whether feature set will be optimized or not output scale threshold
- **scale_thr** (*float*) – only relevant for evars-gpr: output scale threshold
- **scale_seasons** (*int*) – only relevant for evars-gpr: output scale seasons taken into account
- **scale_window_factor** (*float*) – only relevant for evars-gpr: scale window factor based on seasonal periods
- **cf_r** (*float*) – only relevant for evars-gpr: changefinders r param (decay factor older values)
- **cf_order** (*int*) – only relevant for evars-gpr: changefinders SDAR model order param
- **cf_smooth** (*int*) – only relevant for evars-gpr: changefinders smoothing param
- **cf_thr_perc** (*int*) – only relevant for evars-gpr: percentile of train set anomaly factors as threshold for cpd with changefinder
- **scale_window_minimum** (*int*) – only relevant for evars-gpr: scale window minimum
- **max_samples_factor** (*int*) – only relevant for evars-gpr: max samples factor of seasons to keep for gpr pipeline
- **valtest_seasons** (*int*) – define the number of seasons to be used when seasonal_valtest is True
- **seasonal_valtest** (*bool*) – whether validation and test sets should be a multiple of the season length
- **n_splits** (*int*) – splits to use for 'timeseries-cv' or 'cv'
- **config_model_featureset** (*configparser.RawConfigParser*) –

create_new_study()

Create a new optuna study.

Returns
a new optuna study instance

Return type
optuna.study.Study

objective(*trial*)

Objective function for optuna optimization that returns a score

Parameters
trial (*optuna.trial.Trial*) – trial of optuna for optimization

Returns
score of the current hyperparameter config

clean_up_after_exception(*trial_number*, *trial_params*, *reason*)

Clean up things after an exception: delete unfitted model if it exists and update runtime csv

Parameters

- **trial_number** (*int*) – number of the trial
- **trial_params** (*dict*) – parameters of the trial
- **reason** (*str*) – hint for the reason of the Exception

write_runtime_csv(*dict_runtime*)

Write runtime info to runtime csv file

Parameters
dict_runtime (*dict*) – dictionary with runtime information

calc_runtime_stats()

Calculate runtime stats for saved csv file.

Returns
dict with runtime info enhanced with runtime stats

Return type
dict

check_params_for_duplicate(*current_params*)

Check if params were already suggested which might happen by design of TPE sampler.

Parameters
current_params (*dict*) – dictionar with current parameters

Returns
bool reflecting if current params were already used in the same study

Return type
bool

pca_transform_train_test(*train*, *test*)

Deliver PCA transformed train and test set

Parameters

- **train** (*pandas.DataFrame*) – data for the training
- **test** (*pandas.DataFrame*) – data for the testing

Returns
tuple of transformed train and test dataset

Return type
tuple

load_retrain_model(*path*, *filename*, *retrain*, *early_stopping_point=None*, *test=None*)

Load and retrain persisted model :param path: path where the model is saved :param filename: filename of the model :param retrain: data for retraining :param test: data for testing :param early_stopping_point: optional early stopping point relevant for some models :return: model instance

Parameters

- **path** (*str*) –
- **filename** (*str*) –
- **retrain** (*pandas.DataFrame*) –
- **early_stopping_point** (*int*) –
- **test** (*pandas.DataFrame*) –

Return type
tuple

`generate_results_on_test()`

Generate the results on the testing data

Returns
evaluation metrics dictionary

Return type
dict

`get_feature_importance(model, period)`

Get feature importances for models that possess such a feature, e.g. XGBoost

Parameters

- **model** (ForeTiS.model._base_model.BaseModel) – model to analyze
- **period** (*int*) – refitting period

Returns
DataFrame with feature importance information

Return type
pandas.DataFrame

`plot_results(final_results)`

Parameters
final_results (*pandas.DataFrame*) –

ForeTiS.postprocess

Submodules

ForeTiS.postprocess.model_reuse

Module Contents

Functions

apply_final_model(results_directory_model, ...[, ...])	Apply a final model on a new dataset. It will be applied to the whole dataset.

ForeTiS.postprocess.model_reuse.**apply_final_model**(*results_directory_model*, *old_data_dir*, *old_data*, *new_data_dir*, *new_data*, *save_dir=None*, *config_file_path=None*, *retrain_model=True*)

Apply a final model on a new dataset. It will be applied to the whole dataset. So the main purpose of this function is, if you get new samples you want to predict on. If the final model was saved, this will be used for inference on the new dataset. Otherwise, it will be retrained on the initial dataset and then used for inference on the new dataset.

The new dataset will be filtered for the SNP ids that the model was initially trained on.

CAUTION: the SNPs of the old and the new dataset have to be the same!

Parameters

- **results_directory_model** (*str*) – directory that contains the model results that you want to use
- **new_data_dir** (*str*) – directory that contains the new data
- **old_data_dir** (*str*) – directory that contains the old data
- **save_dir** (*str*) – directory to store the results
- **old_data** (*str*) – the old dataset that you used
- **new_data** (*str*) – the new dataset that you want to use
- **config_file_path** (*pathlib.Path*) – the path to the config file you want to use
- **retrain_model** (*bool*) – whether to retrain the model with the whole old dataset

ForeTiS.postprocess.results_analysis

Module Contents

Functions

result_string_to_dictionary(result_string)	Convert result string saved in a .csv file to a dictionary

ForeTiS.postprocess.results_analysis.**result_string_to_dictionary**(*result_string*)

Convert result string saved in a .csv file to a dictionary

Parameters
result_string (*str*) – string from .csv file

Returns
dictionary with info from .csv file

Return type
dict

ForeTiS.preprocess

Submodules

ForeTiS.preprocess.DateCalenderFeatures

Module Contents

Functions

add_date_based_features(df)	Function adding date based features to dataset
add_counters(df, columns_for_counter, resample_weekly, ...)	Function adding counters for upcoming or past public holidays (according to event_lags)

ForeTiS.preprocess.DateCalenderFeatures.**add_date_based_features**(*df*)

Function adding date based features to dataset

Parameters
df (*pandas.DataFrame*) – dataset for adding features

ForeTiS.preprocess.DateCalenderFeatures.**add_counters**(*df*, *columns_for_counter*, *resample_weekly*, *event_lags*, *values_for_counter*)

Function adding counters for upcoming or past public holidays (according to event_lags) with own counters for those specified in special_days

Parameters

- **df** (*pandas.DataFrame*) – dataset for adding features
- **resample_weekly** (*bool*) – whether to resample weekly or not
- **columns_for_counter** (*list*) –
- **event_lags** (*list*) –
- **values_for_counter** (*list*) –

ForeTiS.preprocess.FeatureAdder

Module Contents

Functions

add_cal_features(df, columns_for_counter, ...)	Function adding all calendar-based features
add_statistical_features(seasonal_periods, ...)	Function adding all statistical features

ForeTiS.preprocess.FeatureAdder.**add_cal_features**(*df*, *columns_for_counter*, *resample_weekly*, *event_lags*, *values_for_counter*)

Function adding all calendar-based features

Parameters

- **df** (*pandas.DataFrame*) – dataset used for adding features
- **resample_weekly** (*bool*) – whether to resample weekly or not
- **columns_for_counter** (*list*) –
- **event_lags** (*list*) –
- **values_for_counter** (*list*) –

ForeTiS.preprocess.FeatureAdder.**add_statistical_features**(*seasonal_periods*, *windowsize_current_statistics*, *columns_for_lags*, *columns_for_lags_rolling_mean*, *columns_for_rolling_mean*, *windowsize_lagged_statistics*, *seasonal_lags*, *df*)

Function adding all statistical features

Parameters

- **seasonal_periods** (*int*) – seasonality used for seasonal-based features
- **windowsize_current_statistics** (*int*) – size of window used for feature statistics
- **windowsize_lagged_statistics** (*int*) – size of window used for sales statistics
- **seasonal_lags** (*int*) – seasonal lags to add of the features specified
- **df** (*pandas.DataFrame*) – dataset used for adding features
- **columns_for_lags** (*list*) –
- **columns_for_lags_rolling_mean** (*list*) –
- **columns_for_rolling_mean** (*list*) –

ForeTiS.preprocess.StatisticalFeatures

Module Contents

Functions

add_lagged_statistics(seasonal_periods, ...)	Function adding lagged and seasonal-lagged features to dataset
add_current_statistics(seasonal_periods, ...)	Function adding rolling seasonal statistics

ForeTiS.preprocess.StatisticalFeatures.**add_lagged_statistics**(*seasonal_periods*, *windowsize_lagged_statistics*, *seasonal_lags*, *df*, *columns_for_lags_rolling_mean*)

Function adding lagged and seasonal-lagged features to dataset

Parameters

- **seasonal_periods** (*int*) – seasonal_period used for seasonal-lagged features
- **windowsize_lagged_statistics** (*int*) – size of window used for sales statistics

- **seasonal_lags** (*int*) – seasonal lags to add of the features specified
- **df** (*pandas.DataFrame*) – dataset for adding features
- **columns_for_lags_rolling_mean** (*list*) – the columns where seasonal lagged rolling mean should be applied

ForeTiS.preprocess.StatisticalFeatures.**add_current_statistics**(*seasonal_periods*, *windowsize_current_statistics*, *df*, *columns_for_rolling_mean*, *columns_for_lags*)

Function adding rolling seasonal statistics

Parameters

- **seasonal_periods** (*int*) – seasonal_period used for seasonal rolling statistics
- **windowsize_current_statistics** (*int*) – size of window used for feature statistics
- **df** (*pandas.DataFrame*) – dataset for adding features
- **columns_for_rolling_mean** (*list*) – the columns where the rolling mean should be applied
- **columns_for_lags** (*list*) – the columns that should be lagged by one sample

ForeTiS.preprocess.base_dataset

Module Contents

Classes

Dataset	Class containing datasets ready for optimization.

class ForeTiS.preprocess.base_dataset.**Dataset**(*data_dir*, *data*, *config_file_section*, *test_set_size_percentage*, *windowsize_current_statistics*, *windowsize_lagged_statistics*, *imputation_method='None'*, *config=None*, *event_lags=None*, *valtest_seasons=None*, *seasonal_valtest=None*)

Class containing datasets ready for optimization.

Attributes

- user_input_params (*mixed*): the arguments passed by the user or default values from run.py respectively
- values_for_counter (*list*): the values that should trigger the counter adder
- columns_for_counter (*list*): the columns where the counter adder should be applied
- columns_for_lags (*list*): the columns that should be lagged by one sample
- columns_for_rolling_mean (*list*): the columns where the rolling mean should be applied
- columns_for_lags_rolling_mean (*list*): the columns where seasonal lagged rolling mean should be applied
- string_columns (*list*): columns containing strings
- float_columns (*list*): columns containing floats

- time_column (*str*): columns containing the time information
- seasonal_periods (*int*): how many datapoints one season has
- featuresets_regex (*list*): regular expression with which the feature sets should be filtered
- imputation (*bool*): whether to perfrom imputation or not
- resample_weekly (*bool*): whether to resample weekly or not
- time_format (*str*): the time format, either "W", "D", or "H"
- features (*list*): the features of the dataset
- categorical_columns (*list*): the categorical columns of the dataset
- max_seasonal_lags (*int*): maximal number of seasonal lags to be applied
- target_column (*str*): the target column for the prediction
- featuresets (*list*): list containing all featuresets that get created in this class

Parameters

- **data_dir** (*pathlib.Path*) – data directory where the data is stored
- **data** (*str*) – the dataset that you want to use
- **config_file_section** (*str*) – the section of the config file for the used dataset
- **test_set_size_percentage** (*int*) – size of the test set relevant for cv-test and train-val-test
- **windowsize_current_statistics** (*int*) – the windowsize for the feature engineering of the current statistic
- **windowsize_lagged_statistics** (*int*) – the windowsize for the feature engineering of the lagged statistics
- **imputation_method** (*str*) – the imputation method to use. Options are: 'mean' , 'knn' , 'iterative'
- **config** (*configparser.RawConfigParser*) – the information from dataset_specific_config.ini
- **event_lags** (*int*) – the event lags for the counters
- **valtest_seasons** (*int*) – the number of seasons to be used for validation and testing when seasonal_valtest is True
- **seasonal_valtest** (*bool*) –

Param

seasonal_valtest: whether validation and test sets should be a multiple of the season length or a percentage of the dataset

load_raw_data(*data_dir*, *data*)

Load raw datasets

Parameters

- **data_dir** (*str*) – directory where the data is stored
- **data** (*str*) – which dataset should be loaded

Returns

list of datasets to use for optimization

Return type
pandas.DataFrame

`drop_non_target_useless_columns`(*df*)

Drop the possible target columns that where not chosen as target column

Parameters
df (*pandas.DataFrame*) – DataFrame to use for dropping

Returns
DataFrame with only the target column and features left

`set_dtypes`(*df*)

Function setting dtypes of dataset. cols_to_str are converted to string, rest except date to float.

Parameters
df (*pandas.DataFrame*) – DataFrame whose columns data types should be set

`impute_dataset_train_test`(*df=None*, *test_set_size_percentage=20*, *imputation_method=None*)

Get imputed dataset as well as train and test set (fitted to train set)

Parameters

- **df** (*pandas.DataFrame*) – dataset to impute
- **test_set_size_percentage** (*int*) – the size of the test set in percentage
- **imputation_method** (*str*) – specify the used method if imputation is applied

Returns
imputed dataset, train and test set

Return type
pandas.DataFrame

`featureadding_and_resampling`(*df*)

Function preparing train and test sets based on raw dataset.

Parameters
df (*pandas.DataFrame*) – dataset with raw samples

Returns
Data with added features and resampling

Return type
list

ForeTiS.preprocess.raw_data_functions

Module Contents

Functions

drop_columns(df, columns)	Function dropping all columns specified
drop_rows_by_dates(df, start, end)	Function dropping rows within specified dates
custom_resampler(arraylike, target_column)	Custom resampling function when resampling frequency of dataset
get_one_hot_encoded_df(df, columns_to_encode)	Function delivering dataframe with specified columns one hot encoded
get_simple_imputer(df[, strategy])	Get simple imputer for each column according to specified strategy
get_iter_imputer(df[, sample_posterior, max_iter, ...])	Multivariate, iterative imputer fitted to df with specified parameters
get_knn_imputer(df[, n_neighbors])	Imputer of missing values according to k-nearest neighbors in feature space
encode_cyclical_features(df, columns)	Function that encodes the cyclic features to sinus and cosinus distribution

ForeTiS.preprocess.raw_data_functions.**drop_columns**(*df*, *columns*)

Function dropping all columns specified

Parameters

- **df** (*pandas.DataFrame*) – dataset used for dropping
- **columns** (*list*) – columns which should be dropped

ForeTiS.preprocess.raw_data_functions.**drop_rows_by_dates**(*df*, *start*, *end*)

Function dropping rows within specified dates

Parameters

- **df** (*pandas.DataFrame*) – dataset used for dropping
- **start** (*datetime.date*) – start date for dropped period
- **end** (*datetime.date*) – end date for dropped period

ForeTiS.preprocess.raw_data_functions.**custom_resampler**(*arraylike*, *target_column*)

Custom resampling function when resampling frequency of dataset

Parameters

- **arraylike** (*pandas.Series*) – Series to use for calculation
- **target_column** (*str*) – choosen target column

Returns

sum or mean of arraylike or 1

ForeTiS.preprocess.raw_data_functions.**get_one_hot_encoded_df**(*df*, *columns_to_encode*)

Function delivering dataframe with specified columns one hot encoded

Parameters

- **df** (*pandas.DataFrame*) – dataset to use for encoding
- **columns_to_encode** (*list*) – columns to encode

Returns

dataset with encoded columns

Return type
pandas.DataFrame

ForeTiS.preprocess.raw_data_functions.**get_simple_imputer**(*df*, *strategy='mean'*)

Get simple imputer for each column according to specified strategy

Parameters

- **df** (*pandas.DataFrame*) – DataFrame to impute
- **strategy** (*str*) – strategy to use, e.g. 'mean' or 'median'

Returns
imputer

Return type
sklearn.impute.SimpleImputer

ForeTiS.preprocess.raw_data_functions.**get_iter_imputer**(*df*, *sample_posterior=True*, *max_iter=100*, *min_value=0*, *max_value=None*)

Multivariate, iterative imputer fitted to df with specified parameters

Parameters

- **df** (*pandas.DataFrame*) – DataFrame to fit for imputation
- **sample_posterior** (*bool*) – sample from predictive posterior of fitted estimator (standard: BayesianRidge())
- **max_iter** (*int*) – maximum number of iterations for imputation
- **min_value** (*int*) – min value for imputation
- **max_value** (*int*) – max value for imputation

Returns
imputer

Return type
sklearn.impute.IterativeImputer

ForeTiS.preprocess.raw_data_functions.**get_knn_imputer**(*df*, *n_neighbors=10*)

Imputer of missing values according to k-nearest neighbors in feature space

Parameters

- **df** (*pandas.DataFrame*) – DataFrame to use for imputation
- **n_neighbors** (*int*) – number of neighbors to use for imputation

Returns
imputer

Return type
sklearn.impute.KNNImputer

ForeTiS.preprocess.raw_data_functions.**encode_cyclical_features**(*df*, *columns*)

Function that encodes the cyclic features to sinus and cosinus distribution

Parameters

- **df** (*pandas.DataFrame*) – DataFrame to use for imputation
- **columns** (*list*) – columns that should be encoded

ForeTiS.utils

Submodules

ForeTiS.utils.check_functions

Module Contents

Functions

check_all_specified_arguments(arguments)	Check all specified arguments for plausibility

ForeTiS.utils.check_functions.**check_all_specified_arguments**(*arguments*)

Check all specified arguments for plausibility

Parameters

arguments (*dict*) – all arguments provided by the user

ForeTiS.utils.helper_functions

Module Contents

Functions

get_list_of_implemented_models()	Create a list of all implemented models based on files existing in 'model' subdirectory of the repository.
get_mapping_name_to_class()	Get a mapping from model name (naming in package model without .py) to class name.
set_all_seeds([seed])	Set all seeds of libs with a specific function for reproducibility of results
get_indexes(df, datasplit, folds, seasonal_valtest, ...)	Get the indexes for cv
get_datasplit_config_info_for_resultfolder(r	Get all datasplit info for a result folder

ForeTiS.utils.helper_functions.**get_list_of_implemented_models**()

Create a list of all implemented models based on files existing in ‘model’ subdirectory of the repository.

Return type

list

ForeTiS.utils.helper_functions.**get_mapping_name_to_class**()

Get a mapping from model name (naming in package model without .py) to class name.

Returns

dictionary with mapping model name to class name

Return type

dict

ForeTiS.utils.helper_functions.**set_all_seeds**(*seed=42*)

Set all seeds of libs with a specific function for reproducibility of results

Parameters
seed (*int*) – seed to use

ForeTiS.utils.helper_functions.**get_indexes**(*df*, *datasplit*, *folds*, *seasonal_valtest*, *valtest_seasons*, *seasonal_periods*, *val_set_size_percentage*)

Get the indexes for cv

Parameters

- **df** (*pandas.DataFrame*) – data that should be splited
- **datasplit** (*str*) – splitting method
- **folds** (*int*) – number of folds of the hyperparameter optimization
- **valtest_seasons** (*int*) – the number of seasons to be used for validation and testing when seasonal_valtest is True
- **seasonal_periods** (*int*) – how many datapoints one season has
- **test_set_size_percentage** – size of the test set relevant for cv-test and train-val-test
- **val_set_size_percentage** (*int*) – size of the validation set relevant for train-val-test

Returns
train and test indexes

ForeTiS.utils.helper_functions.**get_datasplit_config_info_for_resultfolder**(*resultfolder*)

Get all datasplit info for a result folder

Parameters
resultfolder (*str*) – path to retrieve info

Returns
datasplit info with datasplit, n_outerfolds, n_innerfolds, val_set_size_percentage, test_set_size_percentage, maf_percentage

Return type
tuple

2.6.2 Submodules

ForeTiS.optim_pipeline

Module Contents

Functions

run(data_dir, save_dir[, datasplit, ...])	Run the whole optimization pipeline

ForeTiS.optim_pipeline.**run**(*data_dir*, *save_dir*, *datasplit='timeseries-cv'*, *test_set_size_percentage=20*, *val_set_size_percentage=20*, *n_splits=3*, *imputation_method=None*, *windowsize_current_statistics=3*, *windowsize_lagged_statistics=3*, *models=None*, *n_trials=200*, *pca_transform=False*, *save_final_model=False*, *periodical_refit_frequency=None*, *refit_drops=0*, *data=None*, *config_file_path=None*, *config_file_section=None*, *refit_window=5*, *intermediate_results_interval=None*, *batch_size=32*, *n_epochs=100000*, *event_lags=None*, *optimize_featureset=False*, *scale_thr=0.1*, *scale_seasons=2*, *cf_thr_perc=70*, *scale_window_factor=0.1*, *cf_r=0.4*, *cf_order=1*, *cf_smooth=4*, *scale_window_minimum=2*, *max_samples_factor=10*, *valtest_seasons=1*, *seasonal_valtest=True*)

Run the whole optimization pipeline

Parameters

- **data_dir** (*str*) – data directory where the phenotype and genotype matrix are stored
- **save_dir** (*str*) – directory for saving the results. Default is None, so same directory as data_dir
- **datasplit** (*str*) – datasplit to use. Options are: nested-cv, cv-test, train-val-test
- **test_set_size_percentage** (*int*) – size of the test set relevant for cv-test and train-val-test
- **val_set_size_percentage** (*int*) – size of the validation set relevant for train-val-test
- **n_splits** (*int*) – splits to use for 'timeseries-cv' or 'cv'
- **imputation_method** (*str*) – the imputation method to use. Options are: 'mean' , 'knn' , 'iterative'
- **windowsize_current_statistics** (*int*) – the windowsize for the feature engineering of the current statistic
- **windowsize_lagged_statistics** (*int*) – the windowsize for the feature engineering of the lagged statistics
- **models** (*list*) – list of models that should be optimized
- **n_trials** (*int*) – number of trials for optuna
- **pca_transform** (*bool*) – whether pca dimensionality reduction will be optimized or not
- **save_final_model** (*bool*) – specify if the final model should be saved
- **periodical_refit_frequency** (*list*) – if and for which intervals periodical refitting should be performed
- **refit_drops** (*int*) – after how many periods the model should get updated
- **data** (*str*) – the dataset that you want to use
- **config_file_path** (*str*) – the path of the config file
- **config_file_section** (*str*) – the section of the config file for the used dataset
- **refit_window** (*int*) – seasons get used for refitting
- **intermediate_results_interval** (*int*) – number of trials after which intermediate results will be saved
- **batch_size** (*int*) – batch size for neural network models
- **n_epochs** (*int*) – number of epochs for neural network models

- **event_lags** (*int*) – the event lags for the counters
- **optimize_featureset** (*bool*) – whether feature set will be optimized or not output scale threshold
- **scale_thr** (*float*) – only relevant for evars-gpr: output scale threshold
- **scale_seasons** (*int*) – only relevant for evars-gpr: output scale seasons taken into account
- **cf_thr_perc** (*int*) – only relevant for evars-gpr: percentile of train set anomaly factors as threshold for cpd with changefinder
- **scale_window_factor** (*float*) – only relevant for evars-gpr: scale window factor based on seasonal periods
- **cf_r** (*float*) – only relevant for evars-gpr: changefinders r param (decay factor older values)
- **cf_order** (*int*) – only relevant for evars-gpr: changefinders SDAR model order param
- **cf_smooth** (*int*) – only relevant for evars-gpr: changefinders smoothing param
- **scale_window_minimum** (*int*) – only relevant for evars-gpr: scale window minimum
- **max_samples_factor** (*int*) – only relevant for evars-gpr: max samples factor of seasons to keep for gpr pipeline
- **valtest_seasons** (*int*) – define the number of seasons to be used when seasonal_valtest is True
- **seasonal_valtest** (*bool*) – whether validation and test sets should be a multiple of the season length

2.6.3 Package Contents

ForeTiS.**__version__ = 0.0.1**

ForeTiS.**__author__ = Josef Eiglsperger, Florian Haselbeck, Dominik G. Grimm**

ForeTiS.**__credits__ = GrimmLab @ TUM Campus Straubing (https://bit.cs.tum.de/)**

PYTHON MODULE INDEX

f

ForeTiS, 43
ForeTiS.evaluation, 43
ForeTiS.evaluation.eval_metrics, 43
ForeTiS.model, 45
ForeTiS.model._additionalmodels, 45
ForeTiS.model._additionalmodels.gpr_sklearn, 45
ForeTiS.model._additionalmodels.lstmbayes_intel, 46
ForeTiS.model._additionalmodels.mlpbayes_intel, 48
ForeTiS.model._base_model, 49
ForeTiS.model._baseline_model, 52
ForeTiS.model._model_classes, 53
ForeTiS.model._model_functions, 57
ForeTiS.model._sklearn_model, 57
ForeTiS.model._stat_model, 59
ForeTiS.model._template_sklearn_model, 61
ForeTiS.model._template_stat_model, 62
ForeTiS.model._template_tensorflow_model, 63
ForeTiS.model._template_torch_model, 64
ForeTiS.model._tensorflow_model, 65
ForeTiS.model._torch_model, 67
ForeTiS.model.ard, 70
ForeTiS.model.averagehistorical, 71
ForeTiS.model.averagemoving, 71
ForeTiS.model.averageseasonal, 72
ForeTiS.model.averageseasonallag, 73
ForeTiS.model.bayesridge, 74
ForeTiS.model.elasticnet, 75
ForeTiS.model.es, 75
ForeTiS.model.gprtf, 76
ForeTiS.model.lasso, 77
ForeTiS.model.lstm, 78
ForeTiS.model.lstmbayes, 80
ForeTiS.model.mlp, 82
ForeTiS.model.mlpbayes, 83
ForeTiS.model.ridge, 84
ForeTiS.model.sarima, 85
ForeTiS.model.sarimax, 85
ForeTiS.model.xgboost, 86
ForeTiS.optim_pipeline, 101
ForeTiS.optimization, 87
ForeTiS.optimization.optuna_optim, 87
ForeTiS.postprocess, 91
ForeTiS.postprocess.model_reuse, 91
ForeTiS.postprocess.results_analysis, 92
ForeTiS.preprocess, 93
ForeTiS.preprocess.base_dataset, 95
ForeTiS.preprocess.DateCalenderFeatures, 93
ForeTiS.preprocess.FeatureAdder, 93
ForeTiS.preprocess.raw_data_functions, 97
ForeTiS.preprocess.StatisticalFeatures, 94
ForeTiS.utils, 100
ForeTiS.utils.check_functions, 100
ForeTiS.utils.helper_functions, 100

INDEX

Symbols

__author__ (*in module ForeTiS*), 103
__credits__ (*in module ForeTiS*), 103
__version__ (*in module ForeTiS*), 103

A

add_cal_features() (*in module ForeTiS.preprocess.FeatureAdder*), 93
add_counters() (*in module ForeTiS.preprocess.DateCalenderFeatures*), 93
add_current_statistics() (*in module ForeTiS.preprocess.StatisticalFeatures*), 95
add_date_based_features() (*in module ForeTiS.preprocess.DateCalenderFeatures*), 93
add_lagged_statistics() (*in module ForeTiS.preprocess.StatisticalFeatures*), 94
add_statistical_features() (*in module ForeTiS.preprocess.FeatureAdder*), 94
apply_final_model() (*in module ForeTiS.postprocess.model_reuse*), 91
ARDRegression (*class in ForeTiS.model.ard*), 70
Arima (*class in ForeTiS.model.sarima*), 85
Arima (*class in ForeTiS.model.sarimax*), 86
AverageHistorical (*class in ForeTiS.model.averagehistorical*), 71
AverageMoving (*class in ForeTiS.model.averagemoving*), 71
AverageSeasonal (*class in ForeTiS.model.averageseasonal*), 72
AverageSeasonal (*class in ForeTiS.model.averageseasonallag*), 73

B

BaselineModel (*class in ForeTiS.model._baseline_model*), 52
BaseModel (*class in ForeTiS.model._base_model*), 49
BayesianRidge (*class in ForeTiS.model.bayesridge*), 74

C

calc_runtime_stats() (*ForeTiS.optimization.optuna_optim.OptunaOptim method*), 90
check_all_specified_arguments() (*in module ForeTiS.utils.check_functions*), 100
check_params_for_duplicate() (*ForeTiS.optimization.optuna_optim.OptunaOptim method*), 90
clean_up_after_exception() (*ForeTiS.optimization.optuna_optim.OptunaOptim method*), 90
common_hyperparams() (*ForeTiS.model._stat_model.StatModel static method*), 60
common_hyperparams() (*ForeTiS.model._tensorflow_model.TensorflowModel static method*), 67
common_hyperparams() (*ForeTiS.model._torch_model.TorchModel static method*), 69
create_new_study() (*ForeTiS.optimization.optuna_optim.OptunaOptim method*), 89
create_sequences() (*ForeTiS.model._additionalmodels.lstmbayes_intel.LSTM method*), 47
create_sequences() (*ForeTiS.model.lstm.LSTM method*), 79
create_sequences() (*ForeTiS.model.lstmbayes.LSTMbayes method*), 81
custom_resampler() (*in module ForeTiS.preprocess.raw_data_functions*), 98

D

Dataset (*class in ForeTiS.preprocess.base_dataset*), 95
define_hyperparams_to_tune() (*ForeTiS.model._additionalmodels.gpr_sklearn.Gpr method*), 45
define_hyperparams_to_tune() (*ForeTiS.model._additionalmodels.lstmbayes_intel.LSTM method*), 46
define_hyperparams_to_tune() (*ForeTiS.model._additionalmodels.mlpbayes_intel.Mlp method*), 48

define_hyperparams_to_tune() (*ForeTiS.model._base_model.BaseModel method*), 50
define_hyperparams_to_tune() (*ForeTiS.model._template_sklearn_model.TemplateSklearnModel method*), 61
define_hyperparams_to_tune() (*ForeTiS.model._template_stat_model.TemplateStatModel method*), 63
define_hyperparams_to_tune() (*ForeTiS.model._template_torch_model.TemplateTorchModel method*), 65
define_hyperparams_to_tune() (*ForeTiS.model.ard.ARDRegression method*), 70
define_hyperparams_to_tune() (*ForeTiS.model.averagehistorical.AverageHistorical method*), 71
define_hyperparams_to_tune() (*ForeTiS.model.averagemoving.AverageMoving method*), 72
define_hyperparams_to_tune() (*ForeTiS.model.averageseasonal.AverageSeasonal method*), 73
define_hyperparams_to_tune() (*ForeTiS.model.averageseasonallag.AverageSeasonal method*), 73
define_hyperparams_to_tune() (*ForeTiS.model.bayesridge.BayesianRidge method*), 74
define_hyperparams_to_tune() (*ForeTiS.model.elasticnet.ElasticNet method*), 75
define_hyperparams_to_tune() (*ForeTiS.model.es.Es method*), 76
define_hyperparams_to_tune() (*ForeTiS.model.lasso.Lasso method*), 77
define_hyperparams_to_tune() (*ForeTiS.model.lstm.LSTM method*), 78
define_hyperparams_to_tune() (*ForeTiS.model.lstmbayes.LSTMbayes method*), 80
define_hyperparams_to_tune() (*ForeTiS.model.mlp.Mlp method*), 82
define_hyperparams_to_tune() (*ForeTiS.model.mlpbayes.Mlpbayes method*), 83
define_hyperparams_to_tune() (*ForeTiS.model.ridge.Ridge method*), 84
define_hyperparams_to_tune() (*ForeTiS.model.sarima.Arima method*), 85
define_hyperparams_to_tune() (*ForeTiS.model.sarimax.Arima method*), 86
define_hyperparams_to_tune() (*ForeTiS.model.xgboost.XgBoost method*), 87
define_model() (*ForeTiS.model._additionalmodels.gpr_sklearn.Gpr method*), 45
define_model() (*ForeTiS.model._additionalmodels.lstmbayes_intel.LSTM method*), 46
define_model() (*ForeTiS.model._additionalmodels.mlpbayes_intel.Mlp method*), 48
define_model() (*ForeTiS.model._base_model.BaseModel method*), 50
define_model() (*ForeTiS.model._template_sklearn_model.TemplateSklearnModel method*), 61
define_model() (*ForeTiS.model._template_stat_model.TemplateStatModel method*), 63
define_model() (*ForeTiS.model._template_torch_model.TemplateTorchModel method*), 65
define_model() (*ForeTiS.model._tensorflow_model.TensorflowModel method*), 66
define_model() (*ForeTiS.model.ard.ARDRegression method*), 70
define_model() (*ForeTiS.model.averagehistorical.AverageHistorical method*), 71
define_model() (*ForeTiS.model.averagemoving.AverageMoving method*), 72
define_model() (*ForeTiS.model.averageseasonal.AverageSeasonal method*), 73
define_model() (*ForeTiS.model.averageseasonallag.AverageSeasonal method*), 73
define_model() (*ForeTiS.model.bayesridge.BayesianRidge method*), 74
define_model() (*ForeTiS.model.elasticnet.ElasticNet method*), 75
define_model() (*ForeTiS.model.es.Es method*), 76
define_model() (*ForeTiS.model.lasso.Lasso method*), 77
define_model() (*ForeTiS.model.lstm.LSTM method*), 78
define_model() (*ForeTiS.model.lstmbayes.LSTMbayes method*), 80
define_model() (*ForeTiS.model.mlp.Mlp method*), 82
define_model() (*ForeTiS.model.mlpbayes.Mlpbayes method*), 83

define_model() (*ForeTiS.model.ridge.Ridge method*), 84
define_model() (*ForeTiS.model.sarima.Arima method*), 85
define_model() (*ForeTiS.model.sarimax.Arima method*), 86
define_model() (*ForeTiS.model.xgboost.XgBoost method*), 87
drop_columns() (*in module ForeTiS.preprocess.raw_data_functions*), 98
drop_non_target_useless_columns() (*ForeTiS.preprocess.base_dataset.Dataset method*), 97
drop_rows_by_dates() (*in module ForeTiS.preprocess.raw_data_functions*), 98

E

ElasticNet (*class in ForeTiS.model.elasticnet*), 75
encode_cyclical_features() (*in module ForeTiS.preprocess.raw_data_functions*), 99
Es (*class in ForeTiS.model.es*), 75
euclid_dist() (*ForeTiS.model._model_classes.SafeMatern52 method*), 56
extend_kernel_combinations() (*ForeTiS.model._additionalmodels.gpr_sklearn.Gpr method*), 45
extend_kernel_combinations() (*ForeTiS.model._tensorflow_model.TensorflowModel method*), 67

F

featureadding_and_resampling() (*ForeTiS.preprocess.base_dataset.Dataset method*), 97
featureset_hyperparam() (*ForeTiS.model._base_model.BaseModel method*), 52
ForeTiS
    module, 43
ForeTiS.evaluation
    module, 43
ForeTiS.evaluation.eval_metrics
    module, 43
ForeTiS.model
    module, 45
ForeTiS.model._additionalmodels
    module, 45
ForeTiS.model._additionalmodels.gpr_sklearn
    module, 45
ForeTiS.model._additionalmodels.lstmbayes_intel
    module, 46
ForeTiS.model._additionalmodels.mlpbayes_intel
    module, 48
ForeTiS.model._base_model
    module, 49
ForeTiS.model._baseline_model
    module, 52
ForeTiS.model._model_classes
    module, 53
ForeTiS.model._model_functions
    module, 57
ForeTiS.model._sklearn_model
    module, 57
ForeTiS.model._stat_model
    module, 59
ForeTiS.model._template_sklearn_model
    module, 61
ForeTiS.model._template_stat_model
    module, 62
ForeTiS.model._template_tensorflow_model
    module, 63
ForeTiS.model._template_torch_model
    module, 64
ForeTiS.model._tensorflow_model
    module, 65
ForeTiS.model._torch_model
    module, 67
ForeTiS.model.ard
    module, 70
ForeTiS.model.averagehistorical
    module, 71
ForeTiS.model.averagemoving
    module, 71
ForeTiS.model.averageseasonal
    module, 72
ForeTiS.model.averageseasonallag
    module, 73
ForeTiS.model.bayesridge
    module, 74
ForeTiS.model.elasticnet
    module, 75
ForeTiS.model.es
    module, 75
ForeTiS.model.gprtf
    module, 76
ForeTiS.model.lasso
    module, 77
ForeTiS.model.lstm
    module, 78
ForeTiS.model.lstmbayes
    module, 80
ForeTiS.model.mlp
    module, 82
ForeTiS.model.mlpbayes
    module, 83
ForeTiS.model.ridge
    module, 84

ForeTiS.model.sarima
    module, 85
ForeTiS.model.sarimax
    module, 85
ForeTiS.model.xgboost
    module, 86
ForeTiS.optim_pipeline
    module, 101
ForeTiS.optimization
    module, 87
ForeTiS.optimization.optuna_optim
    module, 87
ForeTiS.postprocess
    module, 91
ForeTiS.postprocess.model_reuse
    module, 91
ForeTiS.postprocess.results_analysis
    module, 92
ForeTiS.preprocess
    module, 93
ForeTiS.preprocess.base_dataset
    module, 95
ForeTiS.preprocess.DateCalenderFeatures
    module, 93
ForeTiS.preprocess.FeatureAdder
    module, 93
ForeTiS.preprocess.raw_data_functions
    module, 97
ForeTiS.preprocess.StatisticalFeatures
    module, 94
ForeTiS.utils
    module, 100
ForeTiS.utils.check_functions
    module, 100
ForeTiS.utils.helper_functions
    module, 100
forward() (*ForeTiS.model._model_classes.GetOutputZero method*), 55
forward() (*ForeTiS.model._model_classes.PrepareForDropout method*), 56
forward() (*ForeTiS.model._model_classes.PrepareForlstm method*), 55
forward() (*ForeTiS.model._model_classes.PrintLayer method*), 54

G

generate_results_on_test() (*ForeTiS.optimization.optuna_optim.OptunaOptim method*), 91
get_dataloader() (*ForeTiS.model._additionalmodels.lstmbayes_intel.LSTM method*), 47
get_dataloader() (*ForeTiS.model._torch_model.TorchModel method*), 69
get_dataloader() (*ForeTiS.model.lstm.LSTM method*), 79
get_dataloader() (*ForeTiS.model.lstmbayes.LSTMbayes method*), 81
get_datasplit_config_info_for_resultfolder() (*in module ForeTiS.utils.helper_functions*), 101
get_evaluation_report() (*in module ForeTiS.evaluation.eval_metrics*), 44
get_feature_importance() (*ForeTiS.optimization.optuna_optim.OptunaOptim method*), 91
get_indexes() (*in module ForeTiS.utils.helper_functions*), 101
get_inverse_transformed_set() (*ForeTiS.model._stat_model.StatModel method*), 60
get_iter_imputer() (*in module ForeTiS.preprocess.raw_data_functions*), 99
get_knn_imputer() (*in module ForeTiS.preprocess.raw_data_functions*), 99
get_list_of_implemented_models() (*in module ForeTiS.utils.helper_functions*), 100
get_loss() (*ForeTiS.model._torch_model.TorchModel method*), 69
get_mapping_name_to_class() (*in module ForeTiS.utils.helper_functions*), 100
get_one_hot_encoded_df() (*in module ForeTiS.preprocess.raw_data_functions*), 98
get_simple_imputer() (*in module ForeTiS.preprocess.raw_data_functions*), 99
get_torch_object_for_string() (*ForeTiS.model._torch_model.TorchModel static method*), 69
get_transformed_set() (*ForeTiS.model._stat_model.StatModel method*), 60
GetOutputZero (*class in ForeTiS.model._model_classes*), 54
Gpr (*class in ForeTiS.model._additionalmodels.gpr_sklearn*), 45
Gpr (*class in ForeTiS.model.gprtf*), 76

I

impute_dataset_train_test() (*ForeTiS.preprocess.base_dataset.Dataset method*), 97

L

Lasso (*class in ForeTiS.model.lasso*), 77
load_model() (*in module ForeTiS.model._model_functions*), 57

load_raw_data() (*ForeTiS.preprocess.base_dataset.Dataset method*), 96
load_retrain_model() (*ForeTiS.optimization.optuna_optim.OptunaOptim method*), 90
LSTM (*class in ForeTiS.model._additionalmodels.lstmbayes_intel*), 46
LSTM (*class in ForeTiS.model.lstm*), 78
LSTMbayes (*class in ForeTiS.model.lstmbayes*), 80

M

mape() (*in module ForeTiS.evaluation.eval_metrics*), 44
Mlp (*class in ForeTiS.model._additionalmodels.mlpbayes_intel*), 48
Mlp (*class in ForeTiS.model.mlp*), 82
Mlpbayes (*class in ForeTiS.model.mlpbayes*), 83
module
ForeTiS, 43
ForeTiS.evaluation, 43
ForeTiS.evaluation.eval_metrics, 43
ForeTiS.model, 45
ForeTiS.model._additionalmodels, 45
ForeTiS.model._additionalmodels.gpr_sklearn, 45
ForeTiS.model._additionalmodels.lstmbayes_intel, 46
ForeTiS.model._additionalmodels.mlpbayes_intel, 48
ForeTiS.model._base_model, 49
ForeTiS.model._baseline_model, 52
ForeTiS.model._model_classes, 53
ForeTiS.model._model_functions, 57
ForeTiS.model._sklearn_model, 57
ForeTiS.model._stat_model, 59
ForeTiS.model._template_sklearn_model, 61
ForeTiS.model._template_stat_model, 62
ForeTiS.model._template_tensorflow_model, 63
ForeTiS.model._template_torch_model, 64
ForeTiS.model._tensorflow_model, 65
ForeTiS.model._torch_model, 67
ForeTiS.model.ard, 70
ForeTiS.model.averagehistorical, 71
ForeTiS.model.averagemoving, 71
ForeTiS.model.averageseasonal, 72
ForeTiS.model.averageseasonallag, 73
ForeTiS.model.bayesridge, 74
ForeTiS.model.elasticnet, 75
ForeTiS.model.es, 75
ForeTiS.model.gprtf, 76
ForeTiS.model.lasso, 77
ForeTiS.model.lstm, 78
ForeTiS.model.lstmbayes, 80
ForeTiS.model.mlp, 82
ForeTiS.model.mlpbayes, 83
ForeTiS.model.ridge, 84
ForeTiS.model.sarima, 85
ForeTiS.model.sarimax, 85
ForeTiS.model.xgboost, 86
ForeTiS.optim_pipeline, 101
ForeTiS.optimization, 87
ForeTiS.optimization.optuna_optim, 87
ForeTiS.postprocess, 91
ForeTiS.postprocess.model_reuse, 91
ForeTiS.postprocess.results_analysis, 92
ForeTiS.preprocess, 93
ForeTiS.preprocess.base_dataset, 95
ForeTiS.preprocess.DateCalenderFeatures, 93
ForeTiS.preprocess.FeatureAdder, 93
ForeTiS.preprocess.raw_data_functions, 97
ForeTiS.preprocess.StatisticalFeatures, 94
ForeTiS.utils, 100
ForeTiS.utils.check_functions, 100
ForeTiS.utils.helper_functions, 100

O

objective() (*ForeTiS.optimization.optuna_optim.OptunaOptim method*), 89
OptunaOptim (*class in ForeTiS.optimization.optuna_optim*), 87

P

pca_transform() (*ForeTiS.model._base_model.BaseModel method*), 52
pca_transform_train_test() (*ForeTiS.optimization.optuna_optim.OptunaOptim method*), 90
plot_results() (*ForeTiS.optimization.optuna_optim.OptunaOptim method*), 91
predict() (*ForeTiS.model._additionalmodels.lstmbayes_intel.LSTM method*), 47
predict() (*ForeTiS.model._additionalmodels.mlpbayes_intel.Mlp method*), 49
predict() (*ForeTiS.model._base_model.BaseModel method*), 51
predict() (*ForeTiS.model._baseline_model.BaselineModel method*), 53
predict() (*ForeTiS.model._sklearn_model.SklearnModel method*), 58
predict() (*ForeTiS.model._stat_model.StatModel method*), 59
predict() (*ForeTiS.model._tensorflow_model.TensorflowModel method*), 66

predict() (*ForeTiS.model._torch_model.TorchModel method*), 69
predict() (*ForeTiS.model.lstm.LSTM method*), 79
predict() (*ForeTiS.model.lstmbayes.LSTMbayes method*), 81
predict() (*ForeTiS.model.mlpbayes.Mlpbayes method*), 83
PrepareForDropout (*class in ForeTiS.model._model_classes*), 55
PrepareForlstm (*class in ForeTiS.model._model_classes*), 55
PrintLayer (*class in ForeTiS.model._model_classes*), 53

R

result_string_to_dictionary() (*in module ForeTiS.postprocess.results_analysis*), 92
retrain() (*ForeTiS.model._base_model.BaseModel method*), 51
retrain() (*ForeTiS.model._baseline_model.BaselineModel method*), 53
retrain() (*ForeTiS.model._sklearn_model.SklearnModel method*), 58
retrain() (*ForeTiS.model._stat_model.StatModel method*), 59
retrain() (*ForeTiS.model._tensorflow_model.TensorflowModel method*), 66
retrain() (*ForeTiS.model._torch_model.TorchModel method*), 68
retrain() (*ForeTiS.model.averageseasonallag.AverageSeasonal method*), 73
retrain_model_with_results_file() (*in module ForeTiS.model._model_functions*), 57
Ridge (*class in ForeTiS.model.ridge*), 84
run() (*in module ForeTiS.optim_pipeline*), 101

S

SafeMatern52 (*class in ForeTiS.model._model_classes*), 56
save_model() (*ForeTiS.model._base_model.BaseModel method*), 52
set_all_seeds() (*in module ForeTiS.utils.helper_functions*), 100
set_dtypes() (*ForeTiS.preprocess.base_dataset.Dataset method*), 97
SklearnModel (*class in ForeTiS.model._sklearn_model*), 57
smape() (*in module ForeTiS.evaluation.eval_metrics*), 44
StatModel (*class in ForeTiS.model._stat_model*), 59
suggest_all_hyperparams_to_optuna() (*ForeTiS.model._base_model.BaseModel method*), 52
suggest_hyperparam_to_optuna() (*ForeTiS.model._base_model.BaseModel method*), 51

T

TemplateSklearnModel (*class in ForeTiS.model._template_sklearn_model*), 61
TemplateStatModel (*class in ForeTiS.model._template_stat_model*), 62
TemplateTensorflowModel (*class in ForeTiS.model._template_tensorflow_model*), 63
TemplateTorchModel (*class in ForeTiS.model._template_torch_model*), 64
TensorflowModel (*class in ForeTiS.model._tensorflow_model*), 65
TorchModel (*class in ForeTiS.model._torch_model*), 67
train_one_epoch() (*ForeTiS.model._torch_model.TorchModel method*), 68
train_val_loader() (*ForeTiS.model._additionalmodels.lstmbayes_intel.LSTM method*), 47
train_val_loader() (*ForeTiS.model._torch_model.TorchModel method*), 68
train_val_loader() (*ForeTiS.model.lstm.LSTM method*), 78
train_val_loader() (*ForeTiS.model.lstmbayes.LSTMbayes method*), 80
train_val_loop() (*ForeTiS.model._base_model.BaseModel method*), 51
train_val_loop() (*ForeTiS.model._baseline_model.BaselineModel method*), 53
train_val_loop() (*ForeTiS.model._sklearn_model.SklearnModel method*), 58
train_val_loop() (*ForeTiS.model._stat_model.StatModel method*), 59
train_val_loop() (*ForeTiS.model._tensorflow_model.TensorflowModel method*), 66
train_val_loop() (*ForeTiS.model._torch_model.TorchModel method*), 68

U

update() (*ForeTiS.model._base_model.BaseModel method*), 51
update() (*ForeTiS.model._baseline_model.BaselineModel method*), 53

update() (*ForeTiS.model._sklearn_model.SklearnModel method*), 58
update() (*ForeTiS.model._stat_model.StatModel method*), 59
update() (*ForeTiS.model._tensorflow_model.TensorflowModel method*), 66
update() (*ForeTiS.model._torch_model.TorchModel method*), 69
update() (*ForeTiS.model.averageseasonallag.AverageSeasonal method*), 74

V

validate_one_epoch() (*ForeTiS.model._torch_model.TorchModel method*), 68

W

write_runtime_csv() (*ForeTiS.optimization.optuna_optim.OptunaOptim method*), 90

X

XgBoost (*class in ForeTiS.model.xgboost*), 86