foresee

Release 0.1.0a8

Jun 21, 2020

Contents:

Welcome to *foresee* documentation!

foresee is a python package. Provided a time series and its parameters, foresee can generate forecasts using several time series forecasting models in python, can tune hyper parameters of these models, and can compare their forecast results using out of sample forecast accuracy. This library can process more than one time series if a time series id is provided. To get started, install *foresee* using pip

```
$ pip install foresee
```

and try one of these examples.

- *Example 1: one time series as a single column dataframe*
- *Example 2: multiple time series as a dataframe with a time series id column*
- *Example 3: run forecasts with UI app*

or try it out at: https://easy-forecast.herokuapp.com/

Note: Code and documentation for this library are still under development and will change frequently.

CHAPTER 1

Introduction

There are several open source python packages with models for time series forecasting. The goal of this project is to generate forecasts using some of these models, compare their results in a holdout period, and report the outcome. There is also functionality for model hyper-parameter tuning across pre-selected parameters and space using hyperopt library. Forecasting and tuning process can run in parallel using dask library, if needed, to speed up the operation.

This library has a basic web application created using plotly-dash which can accept csv file, for input data, and some parameters using drop downs and check lists. Forecast results is then displayed as a table and can be downloaded.

Currently there are five different forecasting models available. These will generate forecasts using their default parameters if tuning is not selected but with tuning a pre-selected set of their parameters will be tuned over a pre-defind space by comparing forecast accuracy over a holdout period.

1) EWM: Exponentially Weighted Mean
2) FFT: Fast Fourier Transformation
3) Holt-Winters: Holt Winters exponential smoothing model from statsmodels library
4) Prophet: Prophet model from fbprophet library
5) SARIMAX: Sarimax model from statsmodels library

1.1 TODO:

- add new models
- design user control over parameters and parameter space
- include other loss functions like *mse*
- . . .

CHAPTER 2

Quick Start

2.1 Install foresee

foresee is hosted on PyPI and can be installed with pip.

```
$ pip install foresee
```

2.2 Example 1: one time series as a single column dataframe

```
import warnings
warnings.filterwarnings("ignore")


import pandas as pd
import numpy as np
from io import StringIO
import importlib_resources

# import collect_result for handling the process
from foresee.scripts.main import collect_result

# 'basic_time_series_data.csv' file has only one column containing time series values
basic_time_series_data_txt = importlib_resources.files('foresee.data').joinpath(
↪'basic_time_series_data.csv').read_text()

ts_df = pd.read_csv(StringIO(basic_time_series_data_txt))
ts_df.head()

# present data here

# user defind parameters
```

(continues on next page)

(continued from previous page)

```
# if input dataframe has more than one column, provide column name containing time␣
↪series data

endog_colname = None

if len(ts_df.columns) > 1 and endog_colname is None:
        raise ValueError('time series column name is required!!!')

# if uploading your own sample data, update the following parameters if needed

freq = 5
fcst_length = 10
model_list = ['ewm_model', 'fft', 'holt_winters', 'prophet', 'sarimax']

'''
avilable run types:  'all_models', 'best_model', 'all_best'

all_models: no holdout, no tuning, no model competition. return results for all models

best_model: compare models forecast accuracy and return the result of the best model

all_best: compute forecast accuracy for all models and return the result for all␣
↪models

'''

run_type = 'all_models'

# if comparing models results, holdout length is required

if run_type == 'all_models':
        holdout_length = None
else:
        holdout_length = 20


# we are working with one time series and no date-time column so time series id and␣
↪date-time column name are set to None.
gbkey = None
ds_column = None
tune = False


# we are fitting one time series in this example so no need to parallelize.

fit_execution_method = 'non_parallel'


'''
result:  dataframe containing fitted values and future forecasts
fit_results_list:  list of dictionaries containing fitted values, forecasts, and␣
↪errors (useful for debuging)
'''

result, fit_result_list = collect_result(
        ts_df.copy(),
```

(continues on next page)

(continued from previous page)

```
        endog_colname,
        gbkey,
        ds_column,
        freq,
        fcst_length,
        run_type,
        holdout_length,
        model_list,
        fit_execution_method,
        tune
)

result.head()
# present data here
```

2.3 Example 2: multiple time series as a dataframe with a time series id column

```
import warnings
warnings.filterwarnings("ignore")

import pandas as pd
import numpy as np
from io import StringIO
import importlib_resources

# import main from foresee.scripts
from foresee.scripts import main
# upload sample time-series dataframe with columns(id, date_stamp, y)

test_data_light_txt = importlib_resources.files('foresee.data').joinpath('test_data_
↪light.csv').read_text()

ts_df = pd.read_csv(StringIO(test_data_light_txt))


ts_df['date_stamp'] = pd.to_datetime(ts_df['date_stamp'])
ts_df.head()

# user defind parameters

# time series values column name: required if input dataframe has more than one column

endog_colname = 'y'

if len(ts_df.columns) > 1 and endog_colname is None:
        raise ValueError('time series column name is required!!!')

# time series frequency
freq = 5

# out of sample forecast length
fcst_length = 10
```

(continues on next page)

(continued from previous page)

```
# available forecasting models
model_list = ['ewm_model', 'fft', 'holt_winters', 'prophet', 'sarimax']

# avilable run types: 'best_model', 'all_best', 'all_models'
run_type = 'all_best'

# if comparing models (run_type in 'best_model' or 'all_best') then holdout length is␣
↪required

if run_type == 'all_models':
        holdout_length = None
else:
        holdout_length = 20

# fit-forecast computations can be done in parallel for each time series. requires␣
↪dask library!!!
# for sequential processing set fit_execution_method to 'non_parallel'

fit_execution_method = 'parallel'


# since we have two time series in this dataset, time series id column name and date-
↪time column name are required.
gbkey = 'id'
ds_column = 'date_stamp'
tune = True

'''
result:  dataframe containing fitted values and future forecasts
fit_results_list:  list of dictionaries containing fitted values, forecasts, and␣
↪errors (useful for debuging)
'''

result, fit_result_list = main.collect_result(
                ts_df.copy(),
                endog_colname,
                gbkey,
                ds_column,
                freq,
                fcst_length,
                run_type,
                holdout_length,
                model_list,
                fit_execution_method,
                tune
)

result.head()
```

2.4 Example 3: run forecasts with UI app

This simple UI accepts *csv* file for input data and has check lists to set neccessary parameters. Application runs at this url: http://localhost:8050/dash

Excecute the following block of code then navigate to above URL, fill out time series information, and drop your file to be processed. Results will be returned as a table and can be downloaded.

```python
import flask
import dash

server = flask.Flask(__name__)

@server.route('/')
def index():
        return 'Flask root.'

from foresee.webapp.dash_app import app

if __name__ == '__main__':
        app.run_server()
```

CHAPTER 3

Modules Reference

3.1 compose

3.2 fitter

3.3 main

3.4 utils

Local utility functions

`foresee.scripts.utils.`**`read_csv`**`(`*file_name*`)`

[summary]

Parameters **`file_name`** (*`[type]`*) – [description]

Returns [description]

Return type [type]

`foresee.scripts.utils.`**`read_json`**`(`*file_name*`)`

[summary]

Parameters **`file_name`** (*`[type]`*) – [description]

Returns [description]

Return type [type]

CHAPTER 4

Models Reference

4.1 EWM

4.2 FFT

4.3 Holt Winters

4.4 Prophet

4.5 SARIMAX

CHAPTER 5

Authors

- Hamid Mohammadi (hmohammadi6545@gmail.com)

CHAPTER 6

License

Note: MIT License

CHAPTER 7

How to contribute

Contributions in any form are welcome, including:

- Documentation improvements
- Additional tests
- New models
- New features to existing models
- UI design

Discussions take place at foresee channel on slack

join us on slack

CHAPTER 8

Indices and tables

- genindex
- modindex
- search

Python Module Index

Index