forecasting

version 0.1

May 2, 2024

1 Class Overview

The "forecasting" class is designed to simplify out of sample forecasting using time series or panel data. This write up outlines how to utilize the forecasting class in any empirical setting. The class allows for using a wide variety of specification choices and can be applied simply to any data in only a few lines of code.

1.1 Creating Out-of-Sample Predictions

To create a forecast, you must first initiate your specification by calling the forecasting() class, and indicate the various specification choices you plan to use in your forecasting model (see forecasting class parameters below). For example:

```
specification=forecasting(data=dataframe,
                          target='return',
                          forecast_dates='date',
                          actual_date='date_realized',
                          identifiers=['permno'])
```

Here, the data must contain a few key variables:

1. The variable you want to forecast (aka the target)
2. The dates where you are making the forecasts
3. The dates the forecasted values are realized
4. The predictor variables
5. The cross sectional identifiers (only required if doing panel data)

If there are additional variables in your dataset that don't fit into the above categories, they will be put in as predictors by default. You can drop variables from the dataset using the drop_variables=[] argument, or explicitly state the variables you want to use as predictors using the predictors=[] argument.

After initiating your specification, you can then begin training your model hyperparamters (if necessary) by calling the .get_hyper_parameters() method. You may need to re-run this method several times depending on your specification to generate several sets of hyperparameters by inputting different values for the array argument. You can find out how many arrays you will need to run by calling the .n_param_arrays attribute:

```
for array in np.arange(specification.n_param_arrays):
    specification.get_hyper_parameters(array=array,
                                       overwrite=False,
                                       save=True)
```

Now that you have created all of the model hyperparameters, you are ready to begin generating predictions. Again, you may need to iterate over several sets of predictions by inputting different arrays into the .get_predictions() method, and can find out how many arrays you need by calling the .n_pred_arrays attribute:

```
for array in np.arange(specification.n_pred_arrays):
    specification.get_predictions(array=array)
```

1.2 Additional Tips and Tricks

After generating the model hyperparameters. It is often best to clean up the saved hyperparamters by combining them into one easy to access file. You can do this by running the .combine_parameters() method:

```
specification.combine_parameters()
```

This will output the paramters as one csv and delete the individual ones which will help prevent a large excess of files if you are running many specifications. Similarly, you can combine the predictions into one convenient file using the .combine_predictions() method:

```
specification.combine_predictions()
```

Finally, if something happens and you need to re-generate only a few missing files, you can run generate only the missing files by calling .missing_pred_arrays or .missing_param_arrays to find out what arrays are missing. Note you must run the combine_predictions (or combine_parameters) method first:

```
specification.combine_predictions()
for i in specification.missing_pred_arrays:
    specification.get_predictions(array=int(i))
```

2 Detailed Documentation

This section give detail on the arguments used in the key functions outlined in section 1.

2.1 forecasting

The forecasting class must be called to initialize the specification that will be used to create the out of sample forecasts. Specifications are set to recommended values by default, but can be altered using their corresponding arguments. The key specification choices are:

1. (algorithm) The algorithm used in creating the forecasts
2. (objective) The objective function used in generating forecasts
3. (cv_scheme) The cross validation scheme used to create hyperparameters. Use n_folds to change the number of folds
4. (refit_frequency) The frequency at which the model is re-trained
5. (parameter_frequency) The frequency at which the hyper-parameters are re-trained
6. (training_window) How many rolling periods to use in the training dataset
7. (trim_y_var) Whether or not to trim outlier observations in the training dataset

In order to set up the specification, you must input a pandas dataframe and indicate which variables to use for the following:

1. (target) The variable you want to forecast (aka the target)
2. (forecast_dates) The dates where you are making the forecasts
3. (actual_dates) The dates the forecasted values are realized
4. (predictors) [optional] The predictor variables. By default uses all variables not used as target, forecast_dates, actual_dates, or identifiers. Alternatively you can You can drop variables from the dataset using the drop_variables argument.
5. (identifiers) [only required if doing panel data] The cross sectional identifiers

Upon running the predictions and hyperparameters, the class will output the results as csv files into the working directory by default. You can change where these files will go however using the following arguments:

1. (parameter_directory) The directory where you want to save parameters
2. (prediction_directory) The directory where you want to save predictions
3. (result_directory) The directory where you want to save final results

Parameters
data = None: Pandas DataFrame

Dataset containing yvar, xvars ... etc

algorithm = 'EN': str

Which algorithm to use for making predictions. Current options are:

1. 'OLS' : Ordinary Least Squares (or least absolute deviations if the objective is set to mae)
2. 'EN' : Elastic Net
3. 'Ridge': Ridge
4. 'Lasso': Lasso
5. 'RF' : Random forest
6. 'GBRT' : Gradient Boosted Regression Trees

objective = 'mse': str

Which objective function to use in training the model. All algorithms are compatible with all objectives. The options are:

1. 'mse' : least squares
2. 'mae' : least absolute deviations
3. 'huber': huber objective with cutoffs at 1% each tail.

cv_scheme = 'timeseries': str

Which cross validation scheme to use when training hyperparameters. Options are:

1. 'timeseries' : separates the training and validation set temporally while maintaining ordinality.
2. 'kfold' : a standard kfold cross validation.

refit_frequency = 1: int

How often to retrain the model when making predictions. 1 retrains every time forecast_dates changes. -1 only fits the model once at the start and never again. 12 would retrain once a year if you have monthly data.

parameter_frequency = -1: int

How often to retrain the model hyperparameters. 1 retrains every time forecast_dates changes. -1 only trains parameters once at the start and never again. 12 would retrain once a year if you have monthly data.

training_window = -1: int

How much data to use in training the model. -1 uses all available data (expanding window), and any integer n uses the data realized within n periods of the forecasting date

trim_y_var = False:

Whether or not to trim the y-variable in the training dataset at 1on each tail.

forecast_dates = 'tm': float, int, datetime, etc.

The variable that identifies the different periods.

first_prediction_period = 365: float, int, datetime, etc.

The period where you want to begin making forecasts. Must be the same data type as forecast_dates.

actual_dates = 'tm_ann': float, int, datetime, etc.

The variable that identifies the different periods when the forecasted data becomes realized. Must be the same data type as forecast_dates.

identifiers = []: list

A list of the variables in data that are identifiers and should not be used when making predictions. For example ['permno','gvkey','cusip']

target = 'y_var': str

The variable being forecasted.

predictors = None: list

A list of predictor variables to use in forecasting. If left as None, then all variables besides target, forecast_dates, actual_dates, and identifiers are used.

n_folds = 5: int

The number of cross validation folds to use if using the kfold. cv_scheme. If using the timeseries cv_scheme then the most recent 1/n_folds of the data is used as the validation set and the rest is the training set.

parameter_directory = '': str

The path where you want to save parameters. Default is the working directory.

prediction_directory = '': str

The path where you want to save predictions. Default is the working directory.

result_directory = '': str

The path where you want to save final results. Default is the working directory.

drop_variables=[], list

Variables to drop from the predictor set.

verbose=True: boolean

Whether or not to print information about which arrays are being run.

Returns

None

Attributes Added Upon Initialization

n_param_arrays:

How many parameter arrays need to be run in order to generate all hyperparameters.

n_pred_arrays:

How many prediction arrays need to be run in order to generate all predictions.

param_arrays:

Which forecast dates will be used for generating hyperparameters

pred_arrays:

Which forecast dates will be used for generating predictions

param_file_name:

The name of the files where the hyperparameters will be saved excluding the array and .csv at the end of the file names

pred_file_name:

The name of the files where the predictions will be saved excluding the array and .csv at the end of the file names

2.2 get_parameters (forecasting.get_parameters)

The get_parameters method generates the hyperparameters used by the machine learning model in training through a bayesian search. This step is required when using all algorithms except for OLS. Depending on the specification used, you may need to generate multiple sets of hyperparameters (see parameter_frequency argument in forecasting). To find out how many sets of hyperparameters you need, you can check the .n_param_arrays attribute. You can then run each array by inputting the array as an argument in the .get_parameters method. If you have already generated your hyperparameters, then you can have the get_parameters method automatically load the parameters from the parameter_directory by setting overwrite=False. Otherwise the function will re-train the hyperparameters.

Parameters

array = 0: int

Which set of parameters to generate. If Overwrite=False and the parameters exist in the parameter_directory, then it will load the parameters instead of generating them.

overwrite=True: boolean

If set to True, then this forces the creation of new hyperparameters which will overwrite ones already created instead of loading the old ones.

save=True: boolean

If set to True, then the hyperparameters are saved as a csv in the parameter_directory.

Returns
None
Key Attributes Added
best_params:

The optimized hyperparameters (either loaded or created)

hyper_runtime:

How long the hyper-parameter optimization took for the most recently run array in seconds.

forecast_date_param:

The date where the hyperparameters from the most recent array were trained

X_train:

The predictor variables used in training

Y_train:

The target variables used in training

cv_indexes:

The cross validation indexes used in parameter training. Given as a a list of [training set, validation set] lists.

2.3 get_predictions (forecasting.get_predictions)

The get_predictions method generates the out of sample forecasts. Depending on the specification used, you may need to generate multiple sets of hyperparameters (see refit_frequency argument in forecasting). To find out how many sets of predictions you need, you can check the .n_pred_arrays attribute. You can then run each array by inputting the array as an argument in the .get_predictions method.
Parameters
array = 0: int

Which set of predictions to generate.

save=True: boolean

If set to True, then the hyperparameters are saved as a csv in the prediction_directory.

Returns
None
Key Attributes Added
param_use:

The index of the hyperparameters used from the best_params dataframe

prediction_runtime:

How long the prediction generation took for the most recently run array in seconds.

forecast_date:

The date where the model from the most recent predictions was trained.

X_train:

The predictor variables used in training

Y_train:

The target variables used in training

X_test:

The predictor variables used to generate out of sample predictions

Y_test:

The dataframe containing the out of sample predictions, along with the forecast_dates, actual_dates, target, and identifiers.

2.4 combine_parameters (forecasting.combine_parameters)

The combine_parameters method reads in previously generated hyperparameters for the current specification and then combines them into one csv file, before outputting them again in the parameter_directory. Old hyperparameter files are then deleted.

Parameters

None

Returns

None

Key Attributes Added

missing_params:

If some hyperparameter files are missing for the current specification, then the corresponding file names are added as an attribute.

missing_param_arrays:

If some hyperparameter files are missing for the current specification, then the arrays needed to generate those missing files are added as an attribute.

2.5 combine_predictions (forecasting.combine_predictions)

The combine_predictions method reads in previously generated predictions for the current specification and then combines them into one csv file, before outputting them again in the results_directory. Old prediction files are then deleted.

Parameters

None

Returns

None

Key Attributes Added

missing_preds:

If some prediction files are missing for the current specification, then the corresponding file names are added as an attribute.

missing_pred_arrays:

If some prediction files are missing for the current specification, then the arrays needed to generate those missing files are added as an attribute.