inCUBE

AI models for energy consumption forecasting

Davide Molinari

FBK-DSIP

Table of contents

1. inCUBE

1.1 Description

Time series multistep forecasting on consumptions of solar panels

1.2 Commands

The Makefile contains the central entry points for common tasks related to this project.

- **make test**: for running all tests
- **make train**: for training a model per every device
- **make predict**: for inference next values used model device trained
- **make lint**: for linting python files
- **make formatting**: for formatting python files
- **make create_dataset**: for creating dataset train and forecast
- **make create_environment**: for creating virtual environment and install all libraries

2. Code Documentation

2.1 `incube.main`

2.1.1 `get_inferenceobj(config, logger, device_folder)`

Creates and returns an inference object based on the specified target model in the configuration.

Parameters:

- `config` (`dict`) – A dictionary containing configuration parameters. Expected keys under the "predict" section include: - "target_model" (str): The name of the target model (e.g., "GB"). - "save_model_path" (str): Path to the directory where the model is saved. - "stats_folder" (str): Path to the folder for saving statistics. - "plot_folder" (str): Path to the folder for saving plots. - "context_length" (int): Length of the context window for predictions. - "horizon_length" (int): Length of the prediction horizon. - "step" (int): Step size for predictions. - "output_path" (str): Path to save the output predictions. - "timestamp_column" (str): Name of the timestamp column in the data. - "categorical_features" (list): List of categorical feature names. - "covariate_features" (list): List of covariate feature names.
- `logger` (`object`) – Logger instance for logging messages.
- `device_folder` (`str`) – Path to the folder containing device-specific data.

Returns:

- `PredictGB` –

  An instance of the PredictGB class if the target model is "GB".

Raises:

- `ValueError` –

  If the specified target model is not supported.

2.1.2 `get_trainobj(config, logger, device_folder)`

Creates and returns a training object based on the specified configuration.

Parameters:

- **`config`** (`dict`) – A dictionary containing the training configuration. Expected keys under `config["train"]` include: - "target_model" (str): The type of model to train. Currently supports "GB". - "cv" (int): Cross-validation folds. - "strategy" (str): Training strategy. - "early_stopping_rounds" (int): Number of rounds for early stopping. - "eval_metric" (str): Evaluation metric for the model. - "context_length" (int): Length of the context window. - "horizon_length" (int): Length of the prediction horizon. - "step" (int): Step size for predictions. - "covariate_features" (list): List of covariate feature names. - "categorical_features" (list): List of categorical feature names. - "train_size" (float): Proportion of data used for training. - "valid_size" (float): Proportion of data used for validation. - "test_size" (float): Proportion of data used for testing. - "save_model_path" (str): Path to save the trained model. - "extract_metrics" (bool): Whether to extract metrics during training. - "stats_folder" (str): Folder to save statistical outputs. - "plot_folder" (str): Folder to save plots. - "datetime_column" (str): Name of the datetime column in the dataset. - "trials" (int): Number of trials for hyperparameter optimization.
- **`logger`** (`Logger`) – Logger instance for logging messages.
- **`device_folder`** (`str`) – Path to the folder containing device-specific data.

Returns:

- **`TrainGB`** – An instance of the `TrainGB` class configured with the provided parameters.

Raises:

- `ValueError` – If the specified `target_model` is not supported.

2.1.3 `load_config(config_path)`

Loads a YAML configuration file from the specified path.

Parameters:

- `config_path` (`str`) –
  The file path to the YAML configuration file.

Returns:

- `dict` –
  The loaded configuration as a dictionary.

Raises:

- `SystemExit` –
  If the file is not found or cannot be loaded.

2.1.4 `main(args)`

The main entry point of the application. This function handles the execution of different modes (train or predict) based on the provided arguments.

Parameters:

- `args` (`Namespace`) – The command-line arguments containing the following: - file (str): Path to the configuration file. - mode (str): The mode of operation, either "train" or "predict".

Raises:

- `ValueError` – If the provided mode is not supported.
- `Exception` –
  If any other error occurs during execution, it is logged.

Note

Ensure that the configuration file exists and is properly formatted. The logger is initialized based on the provided arguments and configuration.

2.1.5 `predict(config, logger, folders)`

Perform prediction on processed dataset folders for each device.

Parameters:

- `config` (`dict`) – Configuration dictionary containing dataset paths and other settings. Expected to have a key "dataset" with a subkey "path_processed" pointing to the directory of processed datasets.
- `logger` (`Logger`) – Logger instance for logging debug information.

The function iterates through all subdirectories in the processed dataset path, treating each subdirectory as a device folder. For each device folder, it: 1. Extracts the device name from the folder name. 2. Logs the start of the prediction process for the device. 3. Creates an inference object using `get_inferenceobj`. 4. Calls the `predict` method of the inference object to perform predictions. 5. Logs the completion of the prediction process for the device.

2.1.6 `read_data(config, logger, dataset_name)`

Reads a processed dataset from a specified path and returns it as a DataFrame.

Parameters:

- **`config`** (`dict`) – Configuration dictionary containing the dataset path under the key "dataset" -> "path_processed".
- **`logger`** (`Logger`) – Logger instance for logging debug information.
- **`dataset_name`** (`str`) – Name of the dataset file to be read.

Returns:

- –
  pandas.DataFrame: The dataset loaded from the specified file.

Raises:

- `FileNotFoundError` – If the specified dataset file does not exist.
- `Exception` –
  For any other issues encountered while reading the file.

2.1.7 `set_logger(args, config)`

Configures and initializes a logger based on the provided arguments and configuration.

Parameters:

- **`args`** – An object containing runtime parameters. It must have an attribute `mode` that determines the logging mode.
- **`config`** (`dict`) – A dictionary containing logging configuration. It should include a "logging" key with a valid logging configuration and a key corresponding to `args.mode` with a "log_filename" entry.

Returns:

- –
  logging.Logger: A configured logger instance named "DatasetLogger".

2.1.8 `train(config, logger, folders)`

Trains models for each device folder found in the processed dataset path.

Parameters:

- `config` (`dict`) – Configuration dictionary containing the dataset path and other settings.
- `logger` (`Logger`) – Logger instance for logging debug information.

The function iterates through all subdirectories in the processed dataset path, treating each subdirectory as a device folder. For each device folder: 1. Logs the start of the training process for the device. 2. Creates a training object using `get_trainobj`. 3. Trains the model using the `train_model` method of the training object. 4. Logs the completion of the training process for the device.

2.2 `incube.modeling.train`

2.2.1 `TrainGB`

`__build_results_dataset(Y, preds, train_index, val_index, test_index)`

Builds a results dataset by combining actual and predicted values generated by time series model.

Parameters:

- `Y` (`DataFrame`) – The original DataFrame containing the target time series data.
- `preds` (`DataFrame`) – The DataFrame containing the predicted values for the time series.
- `train_index` (`Index`) – Index of the training dataset.
- `val_index` (`Index`) – Index of the validation dataset.
- `test_index` (`Index`) – Index of the test dataset.

Returns:

- – pd.DataFrame: A DataFrame containing the following columns: - 'timestamp_start': The original timestamp of the data point. - 'timestamp': The adjusted timestamp based on the lag. - 'lag': The lag value for the data point. - 'actual': The actual target value. - 'pred': The predicted value. - 'DATASET': The dataset type ('TRAIN', 'VALID', or 'TEST').

Notes

- The resulting DataFrame is sorted by 'timestamp_start' and 'lag'.

`__create_lag_features(df_device)`

Creates lag features for a given DataFrame based on the context length and horizon length.

This method generates lagged feature columns and target columns for time series data. It shifts the "ElectricWConsumed" column by positive and negative offsets to create lagged features for model input and target prediction, respectively. The resulting DataFrame is filtered to remove rows with NaN values in the lagged columns and is downsampled based on the specified stride.

Parameters:

- `df_device` (`DataFrame`) – The input DataFrame containing a "timestamp" column and an "ElectricWConsumed" column.

Returns:

- – pd.DataFrame: A DataFrame with lagged features and targets, downsampled by the stride.

Attributes:

- `lag_cols_feat` (`list`) – A list of column names for the lagged feature columns.
- `lag_cols_target` (`list`) – A list of column names for the lagged target columns.

Notes

- The DataFrame is sorted by the "timestamp" column before creating lag features.
- Rows with NaN values in any of the lagged columns are dropped.
- The stride parameter determines the downsampling rate of the resulting DataFrame.

`__extract_metrics(X_train, y_train, X_val, y_val, X_test, y_test)`

Extracts and computes evaluation metrics (RMSE, MAE, R2) for the model predictions on training, validation, and test datasets. Optionally saves the computed metrics to a CSV file if a stats folder is provided.

Parameters:
- `X_train` (`DataFrame`) – Features for the training dataset.
- `y_train` (`Series`) – Target values for the training dataset.
- `X_val` (`DataFrame`) – Features for the validation dataset.
- `y_val` (`Series`) – Target values for the validation dataset.
- `X_test` (`DataFrame`) – Features for the test dataset.
- `y_test` (`Series`) – Target values for the test dataset.

Returns:
- – pd.DataFrame: A DataFrame containing the results dataset with predictions and corresponding metrics for each lag and dataset type.

Notes
- The method computes metrics for specific lags (0, 12, 23).
- If `self.stats_folder` is not None, the metrics are saved to a CSV file named `stats_<device>.csv` in the specified folder.
- Logs a debug message if saving stats is skipped due to a None path.

`__get_model(params)`

Creates and returns an XGBoost model with the specified parameters.

Parameters:
- `params` (`dict`) – A dictionary containing the hyperparameters for the XGBoost model.

Returns:
- – xgb.Booster: An instance of the XGBoost model configured with the provided parameters.

`__init__(cv: int, strategy: str, early_stopping_rounds: int, eval_metric: str, context_length: int, horizon_length: int, stride: int, cov_cols: list, cat_cols: list, train_size: float, valid_size: float, test_size: float, device_folder: str, logger: object, path_save_model: str, extract_metrics: bool, stats_folder: str, plot_folder: str, timestamp_column: str, trials: int, return_results: bool, quantiles: list, top_n: int)`

Initialize the training configuration for the model.

Parameters:

- `cv` (`int`) – Number of cross-validation folds.
- `strategy` (`str`) – Training strategy to be used.
- `early_stopping_rounds` (`int`) – Number of rounds for early stopping.
- `eval_metric` (`str`) – Evaluation metric for model performance.
- `context_length` (`int`) – Length of the context window.
- `horizon_length` (`int`) – Length of the prediction horizon.
- `stride` (`int`) – Step size for sliding window.
- `cov_cols` (`list`) – List of covariate column names.
- `cat_cols` (`list`) – List of categorical column names.
- `train_size` (`float`) – Proportion of data to use for training.
- `valid_size` (`float`) – Proportion of data to use for validation.
- `test_size` (`float`) – Proportion of data to use for testing.
- `device_folder` (`str`) – Path to the device folder.
- `logger` (`object`) – Logger object for logging information.
- `path_save_model` (`str`) – Path to save the trained model.
- `extract_metrics` (`bool`) – Whether to extract metrics during training.
- `stats_folder` (`str`) – Path to save statistical outputs.
- `plot_folder` (`str`) – Path to save plots.
- `timestamp_column` (`str`) – Name of the timestamp column in the dataset.
- `trials` (`int`) – Number of trials for hyperparameter optimization.
- `return_results` (`bool`) – Whether to return results after training.

`__plot_boxplot_per_lag(plot_path, df_results)`

Generates and saves a boxplot visualizing the absolute error per lag and dataset.

This method calculates the absolute error between the actual and predicted values in the provided DataFrame, and creates a boxplot to display the distribution of errors for each lag and dataset type. The plot is saved as a PNG file.

Parameters:

- `plot_path` (`str`) – The directory path where the plot will be saved.
- `df_results` (`DataFrame`) – A DataFrame containing the following columns: - "actual": The actual values. - "pred": The predicted values. - "lag": The lag values. - "DATASET": The dataset type.

Saves

A PNG file of the boxplot at the specified `plot_path` with the filename formatted as "{self.device}_BOXPLOT_ERROR.png".

`__plot_features_importances(plot_path)`

Plots and saves the feature importances of the model.

This method generates a plot of the feature importances for the model and saves it as a PNG file in the specified directory.

Parameters:

- `plot_path` (`str`) – The directory path where the plot image will be saved.

Saves

A PNG file named "_FEAT_IMPORTANCES.png" in the specified `plot_path` directory, where is the value of the `self.device` attribute.

`__plot_lag0(plot_path, df_results)`

Generates and saves a line plot for lag 0 predictions and actual values.

This method creates a plot comparing the actual values and predicted values for lag 0, separated by dataset type (TRAIN, VALID, TEST). The plot is saved as a PNG file in the specified directory.

Parameters:

- `plot_path` (`str`) – The directory path where the plot image will be saved.
- `df_results` (`DataFrame`) – A DataFrame containing the results with the following columns: - "lag": The lag value (used to filter for lag 0). - "timestamp": The timestamp for each data point. - "actual": The actual observed values. - "pred": The predicted values. - "DATASET": The dataset type (e.g., TRAIN, VALID, TEST).

Saves

A PNG file named "{device}_TRAIN_LAG0.png" in the specified `plot_path` directory, where `device` is an attribute of the class instance.

`__plot_random_day(plot_path, df_results)`

Plots and saves a visualization of the actual vs predicted values for a random day from the provided results dataframe.

Parameters:

- `plot_path` (`str`) – The directory path where the plot image will be saved.
- `df_results` (`DataFrame`) – A dataframe containing the results with columns "timestamp_start", "timestamp", "actual", and "pred".

Saves

A PNG image of the plot in the specified `plot_path` directory with the filename format "{device}_TRAIN_RANDOM_DAY.png".

`__plots(df_results)`

Generates and saves various plots based on the provided results dataframe.

This method creates a directory for saving plots if it does not already exist. It then generates and saves the following types of plots: - Lag 0 plot - Random day plot - Boxplot per lag - Feature importances plot

Parameters:
- `df_results` (`DataFrame`) – The dataframe containing the results data used for generating the plots.

Returns:
- –
  None

`__read_data()`

Reads and processes training data from a CSV file.

This method performs the following steps: 1. Reads the training data CSV file located in the device folder. 2. Parses the specified timestamp column as datetime. 3. Creates lag features for the dataset. 4. Splits the dataset into training and testing subsets.

Returns:
- `tuple` – A tuple containing the training and testing datasets after processing.

`__save_model()`

Saves the current model to a specified path in JSON format.

This method checks if the `path_save_model` attribute is set. If it is `None`, the method logs a debug message and skips the save operation. Otherwise, it creates the necessary directory structure and saves the model to a JSON file named after the device.

The saved model file will be located at: `<path_save_model>/GB/<device>.json`

Note
- The method assumes that the `self.model` object has a `save_model` method that handles saving the model to the specified file path.

Returns:
- –
  None

`__search_best_parameters(X_train, y_train, X_val, y_val)`

Searches for the best hyperparameters for the model using Optuna.

This method performs hyperparameter optimization for an XGBoost model by defining a search space and evaluating the performance of different parameter combinations using a validation dataset. The best parameters are then used to create and return a model.

Parameters:

- `X_train` (`DataFrame or ndarray`) – Training feature set.
- `y_train` (`Series or ndarray`) – Training target values.
- `X_val` (`DataFrame or ndarray`) – Validation feature set.
- `y_val` (`Series or ndarray`) – Validation target values.

Returns:

- – xgb.Booster: An XGBoost model trained with the best hyperparameters.

Notes

- The method uses Optuna to perform the hyperparameter search.
- The objective function minimizes the root mean squared error (RMSE) on the validation dataset.
- The best hyperparameters and the corresponding score are logged.

Raises:

- `ValueError` – If the input data is not in the expected format.

`__split_dataset(df_device)`

Splits the dataset into training, validation, and test sets, and separates features (X) and target (y) for each set.

Parameters:

- `df_device` (`DataFrame`) – The input dataframe containing the dataset to be split. It must include columns for categorical features, covariate features, lag features, and the target.

Returns:

- `tuple` – A tuple containing the following: - X_train (pd.DataFrame): Features for the training set. - y_train (pd.DataFrame): Target for the training set. - X_val (pd.DataFrame): Features for the validation set. - y_val (pd.DataFrame): Target for the validation set. - X_test (pd.DataFrame): Features for the test set. - y_test (pd.DataFrame): Target for the test set.

Notes

- The dataframe is expected to have a "timestamp" column, which will be used as the index.
- The categorical columns specified in `self.cat_cols` are converted to the "category" dtype.
- The split sizes are determined by `self.train_size`, `self.valid_size`, and `self.test_size`, which represent proportions of the dataset.
- The `self.horizon_length` parameter is used to exclude the last portion of the dataset from the test set.
- The features are selected based on `self.cov_cols` and `self.lag_cols_feat`, while the target is selected based on `self.lag_cols_target`.

`train_model()`

Trains the model using the provided training, validation, and test datasets.

This method reads the data, searches for the best model parameters, trains the model, saves the trained model, and optionally extracts metrics and generates plots.

Returns:

- – pd.DataFrame or None: A DataFrame containing the extracted metrics if `self.return_results`
- – is True and `self.extract_metrics` is enabled, otherwise None.

Steps

1. Reads the training, validation, and test datasets.
2. Searches for the best model parameters using the validation set.
3. Trains the model on the training dataset.
4. Saves the trained model to disk.
5. Optionally extracts metrics and generates plots if `self.extract_metrics` is True.
6. Returns the metrics DataFrame if `self.return_results` is True.

Note

- The method assumes that the following helper methods are implemented:
- `__read_data`: Reads and splits the data into training, validation, and test sets.
- `__search_best_parameters`: Searches for the best hyperparameters for the model.
- `__save_model`: Saves the trained model to disk.
- `__extract_metrics`: Extracts performance metrics for the model.
- `__plots`: Generates plots for the extracted metrics.
- Logging is used to track the progress of the training process.

2.3 `incube.modeling.predict`

2.3.1 `PredictGB`

`__build_results_dataset(start_time, preds)`

Builds a results dataset containing lag values, timestamps, and predictions.

Parameters:

- **`start_time`** (`Timestamp`) – The starting timestamp for generating the time series.
- **`preds`** (`list or ndarray`) – A list or array of prediction values corresponding to the horizon length.

Returns:

- – pd.DataFrame: A DataFrame containing the following columns: - 'lag': The lag values ranging from 0 to horizon_length - 1. - 'timestamp': The timestamps generated by adding hourly offsets to the start_time. - 'pred': The prediction values forecasted by the model.

`__create_lag_features(df_device)`

Creates lag features for a given DataFrame to be used in time series modeling.

This method generates lagged features for both past (context) and future (horizon) values of the "ElectricWConsumed" column. It also removes rows with missing values resulting from the lagging process and applies a stride to downsample the data.

Parameters:

- `df_device` (`DataFrame`) – The input DataFrame containing a time-indexed "ElectricWConsumed" column.

Returns:

- – pd.DataFrame: A DataFrame with lagged features added, rows with missing values removed, and downsampled according to the stride.

Attributes:

- `self.lag_cols_feat` (`list`) – A list of column names for the past lagged features.
- `self.lag_cols_target` (`list`) – A list of column names for the future lagged features.

Notes

- The `context_length` attribute determines the number of past lags to create.
- The `horizon_length` attribute determines the number of future lags to create.
- The `stride` attribute determines the downsampling rate of the resulting DataFrame.

`__forecast(df_forecast)`

Generates a forecast using the provided dataframe and the trained model.

Parameters:

- `df_forecast` (`DataFrame`) – A dataframe containing the input features and lagged columns required for forecasting. The dataframe should include columns specified in `self.cov_cols` and `self.lag_cols_feat` for input features, and `self.lag_cols_target` for target lagged values.

Returns:

- – pd.DataFrame: A dataframe containing the forecast results, built using the predictions
- – and the index of the last row in the input dataframe.

`__init__(path_model, device_folder, logger, stats_folder, plot_folder, context_length, horizon_length, stride, output_path, return_results, timestamp_column, cat_cols, cov_cols)`

Initializes the prediction model with the specified parameters.

Parameters:

- `path_model` (`str`) – Path to the model file.
- `device_folder` (`str`) – Path to the folder containing device-specific data.
- `logger` (`Logger`) – Logger instance for logging messages.
- `stats_folder` (`str`) – Path to the folder for saving statistical outputs.
- `plot_folder` (`str`) – Path to the folder for saving plots.
- `context_length` (`int`) – Length of the context window for predictions.
- `horizon_length` (`int`) – Length of the prediction horizon.
- `stride` (`int`) – Step size for moving the context window.
- `output_path` (`str`) – Path to save the prediction results.
- `return_results` (`bool`) – Whether to return the prediction results.
- `timestamp_column` (`str`) – Name of the column containing timestamps.
- `cat_cols` (`list`) – List of categorical column names.
- `cov_cols` (`list`) – List of covariate column names.

`__load_model()`

Loads a machine learning model for the specified device.

This method constructs the path to the model file based on the device name. If the model file does not exist at the constructed path, a warning is logged. The method then initializes an XGBRegressor instance and loads the model from the specified file.

Attributes:

- `self.path_model` (`str`) – The base path where the model files are stored.
- `self.device` (`str`) – The name of the device for which the model is being loaded.
- `self.logger` (`Logger`) – Logger instance for logging warnings or errors.
- `self.model` (`XGBRegressor`) – The machine learning model instance.

Raises:

- `FileNotFoundError` – If the model file does not exist at the specified path.

`__plot(df_results)`

Generates and saves a plot of predictions over time.

This method creates a line plot using the provided DataFrame `df_results`, which contains timestamps and prediction values. The plot is saved as a PNG file in a specified folder structure based on the `plot_folder` and `device` attributes of the class. If `plot_folder` is None, the method skips the plotting process.

Parameters:

- **`df_results`** (`DataFrame`) – A DataFrame containing the following columns: - "timestamp": The timestamps for the x-axis. - "pred": The prediction values for the y-axis.

Behavior

- Creates a directory structure if it does not exist.
- Saves the plot as a PNG file named `<device>_INFERENCE.png` in the folder `<plot_folder>/<device>`.
- Configures the plot with appropriate titles, labels, and font sizes.

Notes

- The method uses `matplotlib` for plotting.
- The plot is closed after saving to free up resources.
- If `plot_folder` is None, a debug message is logged, and the method returns without generating a plot.

`__read_data()`

Reads and processes historical and forecast data for a device.

This method reads forecast CSV files, located in the `device_folder` directory. It ensures that the historical data matches the required context length and the forecast data matches the required horizon length. Categorical columns are converted to the "category" data type.

Returns:

- – pd.DataFrame: A DataFrame containing the concatenated historical and
- – forecast data, with categorical columns properly typed.

Raises:

- `Exception` – If the size of the historical data is less than the required
- `Exception` – If the size of the forecast data is less than the required

`__save_forecast(df_results)`

Saves the forecast results to a CSV file if an output path is specified.

Parameters:
- `df_results` (`DataFrame`) – The DataFrame containing the forecast results.

Returns:
- – None

Notes
- If `self.output_path` is None, the method logs a debug message and skips saving.
- The file is saved with the name of the device as `<device>.csv` in the specified output path.

`predict()`

Executes the prediction process by loading the model, reading input data, creating lag features, generating forecasts, saving the results, and plotting the forecasts. Optionally returns the forecast results.

Steps: 1. Load the prediction model. 2. Read the input data required for forecasting. 3. Create lag features from the input data. 4. Generate forecasts using the model. 5. Save the forecast results to a CSV file. 6. Plot the forecast results. 7. Optionally return the forecast results if `self.return_results` is True.

Returns:
- – pd.DataFrame or None: The forecast results as a DataFrame if
- – `self.return_results` is True, otherwise None.

2.4 `incube.modeling.model`

2.4.1 `XGBRegressorQuantileMultistep`

Bases: `XGBRegressor`

`fit(X, y, **kwargs)`

Fits multiple models for time series forecasting using XGBoost.

This method trains a series of models, one for each target in the prediction horizon, using quantile regression. The training and validation datasets are provided as inputs, along with additional parameters for model configuration.

Parameters:

- `X` (`DataFrame`) – The feature matrix for training.
- `y` (`DataFrame`) – The target matrix for training, containing lagged values.
- `**kwargs` – Additional keyword arguments, including: - eval_set (list): A list containing a tuple of validation features and targets (X_val, y_val). - verbose (bool): If True, prints progress information during model fitting.

Attributes:

- `self.models` (`dict`) – A dictionary storing the trained models for each target in the prediction horizon.
- `self.params` (`dict`) – Parameters for the XGBoost model.

Raises:

- `KeyError` – If 'eval_set' is not provided in kwargs.
- `ValueError` – If the validation set does not contain the expected lagged targets.

Example

model.fit(X_train, y_train, eval_set=[(X_val, y_val)], verbose=True)

`get_features_importance(importance_type='gain', top_n=15)`

Compute and return the aggregated feature importance across all trained models. This method calculates the importance of features based on the specified importance type and aggregates the values across all models in the ensemble. It then returns the top N features sorted by their importance. Parameters:

---

importance_type : str, optional The type of importance to retrieve from the models. Default is 'gain'. Common options include 'weight', 'gain', 'cover', etc., depending on the model's API. top_n : int, optional The number of top features to return based on their importance. Default is 15. Returns:

---

pd.Series A pandas Series containing the top N features sorted by their aggregated importance across all models. The index represents the feature names, and the values represent their importance scores. Raises:

ValueError If no models have been trained yet (i.e., `self.models` is empty).

`predict(X, y=None)`

Generate predictions for the given input data and optionally compare them with actual values.

Parameters:

- `X` (`DataFrame`) – Input features for prediction. The index of the DataFrame is expected to represent timestamps.
- `y` (`DataFrame`, default: `None`) – Actual values for comparison. If provided, it should contain columns named

Returns:

- – pd.DataFrame: A DataFrame containing the following columns:
- –
  - 'timestamp_start': The original timestamps from the input data, repeated for each prediction lag.
- –
  - 'timestamp': The predicted timestamps for each lag.
- –
  - 'lag': The lag index for each prediction.
- –
  - 'pred': The predicted values.
- –
  - 'lower_bound': The lower bound of the prediction interval.
- –
  - 'upper_bound': The upper bound of the prediction interval.
- –
  - 'actual' (optional): The actual values corresponding to each lag, if `y` is provided.

Notes

- The method assumes that `self.models` is a list of models, each corresponding to a specific lag.
- Each model in `self.models` should have an `inplace_predict` method that returns predictions in the form of a 2D array with columns representing lower bound, predicted value, and upper bound.
- The length of `self.models` should match `self.prediction_len`, which defines the number of lags to predict.