```
%load_ext autoreload
%autoreload 2

import os, sys
from google.colab import drive

drive.mount('/content/drive')
nb_path = '/content/drive/MyDrive/Colab Notebooks/site-packages'
my_pakcage = '/content/drive/MyDrive/Colab Notebooks/my-packages'
sys.path.insert(0, nb_path)  # or append(nb_path)
sys.path.insert(0, my_pakcage)  # or append(nb_path)
```

```
Mounted at /content/drive
```

DeepAR hyper-parameter tuner using Bayesian-optimization.

Environment: tested on Google colab's gpu runtime environment. Expected to also work on cpu environment

Used 3rd party packages: pandas, numpy, mxnet, gluonts

Used internal packages: typing, os, pathlib, warnings

Usage:

```
1. prepare a dataset with timestamp as index.

2. split the dataset into train and valid set.
   The types of two datasets are recommended as pandas DataFrame. May not work on other types.

3. set the parameter bounds as dictionary with tuples or single number.
    ex) {num_cells : (20, 40), epochs : 30, ... }
    The parameters used are defined in the class DeepAR.model method
```

내부 코드

```
import pandas as pd
import numpy as np
from typing import Dict, Tuple, List, Union, Optional
import mxnet as mx
import os
from pathlib import Path
import warnings
from solar_energy_forecast.utils.timestamper import Timestamper
from solar_energy_forecast.utils.utilities import *

from gluonts.model.predictor import Predictor
from gluonts.dataset.common import Dataset
from gluonts.model.deepar import DeepAREstimator
from gluonts.mx.trainer import Trainer
from gluonts.dataset.common import ListDataset
from gluonts.evaluation.backtest import make_evaluation_predictions

try:
    from bayes_opt import BayesianOptimization

except ImportError as e:
    print("Bayesian Optimization package cannot be imported. Check if it is installed."
          "If not installed use $ pip install bayesian-optimization")
    exit(-10)


# abstract class for Bayesian Tuner
class BayesianTuner:
    def __init__(self,
                 # input dataset format not determined
                 train_df,
                 valid_df,
                 pbounds: Dict[str, Union[Tuple[float, float]]]):  # {param_name: (lower, upper)
        self._best_loss = None
        self._predictor = None
        self._estimator = None
        self.train_df = train_df
        self.valid_df = valid_df
        self.pbounds = pbounds
        self._records = []

    # given forecast and true values, return the sum of all
    def quantile_loss(self, y_true: Union[np.ndarray, pd.Series, pd.DataFrame],
                      y_forecast: Union[np.ndarray, pd.DataFrame],
                      quantiles: Optional[List[float]] = None):
        # TODO: may modify the base calculation structure base to numpy instead of dataframe, fo
        # default quantiles
        if not quantiles:
            quantiles = [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]

        # check if the quantiles are same
        assert len(quantiles) == y_forecast.shape[
            1], "Number of quantiles from forecast and quantiles list is different"

        # cast forecasts to dataframe always
        if isinstance(y_forecast, np.ndarray):
            y_forecast = pd.DataFrame(y_forecast, columns=quantiles)

        elif isinstance(y_forecast, pd.DataFrame):
            y_forecast.columns = quantiles

        # cast y_true
        if isinstance(y_true, pd.DataFrame):
            assert y_true.shape[1] != 1, "y_true value must be shape of ( , 1)"
            y_true = y_true.values

        elif isinstance(y_true, pd.Series):
            y_true = y_true.values
            y_true = y_true.values

        # quantile loss = max(q*(y_pred - y_true), (1-q)*(y_pred, y_true))
        for quantile in y_forecast.columns:
            diff = y_forecast[quantile] - y_true
            diff = np.where(diff >= 0, diff * quantile, (quantile - 1) * diff)
            y_forecast[quantile] = diff

        return y_forecast.sum().sum()

    # abstract method
    def model(self, **kwargs):
        raise NotImplementedError

    def tune_model(self, **kwargs):
        raise NotImplementedError

    # exception handling
    def return_records(self) -> pd.DataFrame:
        if self._records is not None:
            return pd.concat(self._records)

        else:
            warnings.warn("You must first train the model to get trained estimator. Call tune_mo

    def return_estimator(self):
        if self._estimator is not None:
            return self._estimator

        else:
            warnings.warn("You must first train the model to get trained estimator. Call tune_mo

    def return_predictor(self):
        if self._predictor is not None:
            return self._predictor

        else:
            warnings.warn("You must first train the model to get trained predictor. Call tune_mo

    def return_best_loss(self):
        if self._best_loss != 0:
            return round(self._best_loss, 4)

        else:
            warnings.warn("You must first train the model to get best loss. Call tune_model firs

class DeepARTuner(BayesianTuner):
    def __init__(self,
                 train_df: pd.DataFrame,
                 valid_df: pd.DataFrame,
                 pbounds: Dict[str, Tuple[float, float]],
                 learning_rate: float,
                 use_feat_dynamic_real: bool = True,
                 prediction_window: Optional[int] = None,
                 batch_size: int = 64):
        super().__init__(train_df, valid_df, pbounds)
        super().__init__(train_df, valid_df, pbounds)
        # check available device
        if not mx.test_utils.list_gpus():
            self.ctx = mx.context.cpu()

        else:
            self.ctx = mx.context.gpu()

        # set prediction length
        if prediction_window:
            self.prediction_window = prediction_window

        else:
            self.prediction_window = 2 * 48  # two days as default

        self.learning_rate = learning_rate
        self.transform_to_ListData()
        self.use_feat_dynamic_real = use_feat_dynamic_real
        self.batch_size = batch_size
        self.internal_iter_num = 0
        self.current_saving_folder = None

    # method to convert given dataframe into ListData class of gluonts
    def transform_to_ListData(self):
        train_DHI = self.train_df.DHI.values[:-self.prediction_window]
        train_DNI = self.train_df.DNI.values[:-self.prediction_window]
        train_RH = self.train_df.RH.values[:-self.prediction_window]
        train_T = self.train_df['T'][:-self.prediction_window].values

        self.train_ds = ListDataset(
            [{"start": self.train_df.index[0],
              "target": np.array(self.train_df.TARGET.values[:-self.prediction_window]),
              "feat_dynamic_real": [train_DHI, train_DNI, train_RH, train_T]
              }],
            freq="30min",
            one_dim_target=True
        )

        # TODO: Is valid set configured correctly?
        valid_DHI = self.valid_df.DHI.values[:-self.prediction_window]
        valid_DNI = self.valid_df.DNI.values[:-self.prediction_window]
        valid_RH = self.valid_df.RH.values[:-self.prediction_window]
        valid_T = self.valid_df['T'][:-self.prediction_window].values

        self.valid_ds = ListDataset(
            [{"start": self.valid_df.index[0],
              "target": np.array(self.valid_df.TARGET.values[:-self.prediction_window]),
              "feat_dynamic_real": [valid_DHI, valid_DNI, valid_RH, valid_T]
              }],
            freq="30min",
            one_dim_target=True
        )

    # calculate the sum of quantile loss from given period
    # quantile forecast values and true values must be entered. Prediction is made in this metho
    def forecast_quantiles(self,
    def forecast_quantiles(self,
                           dataset: Dataset,
                           predictor: Predictor,
                           num_samples: int = 100,
                           prediction_window: int = 2*48) -> pd.DataFrame:
        quantile_forecasts = {}
        quantiles = [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]

        forecast_iter, ts_iter = make_evaluation_predictions(dataset, predictor=predictor, num_s

        forecasts = next(forecast_iter)  # forecasts: instance from SampleForecast
        tss = next(ts_iter)  # tss : instance from pd.DataFrame

        for quantile in quantiles:
            quantile_forecasts[quantile] = forecasts.quantile(quantile)

        return pd.DataFrame(quantile_forecasts, columns=quantiles, index=tss.index[-self.predict

    # deepAR model for bayesian optimization.
    # This returns the sum of quantile loss for given parameters selected by bayesian optimizer
    def model(self,
              epochs,
              context_length,
              num_cells,
              num_layers
              ) -> float:

        estimator_params = {
            'cell_type': 'lstm',
            'context_length': int(context_length),
            'num_cells': int(num_cells),
            'num_layers': int(num_layers),
            'use_feat_dynamic_real': self.use_feat_dynamic_real,
            'epochs': int(epochs)
        }

        trainer = Trainer(epochs=int(epochs),
                          batch_size=self.batch_size,
                          ctx=self.ctx,
                          learning_rate=self.learning_rate)

        estimator = DeepAREstimator(estimator_params, trainer=trainer, freq='30min',
                                    prediction_length=self.prediction_window)

        predictor = estimator.train(training_data=self.train_ds)

        # TODO: maybe backtest_metrics can be used to shorten the process below
        forecast_df = self.forecast_quantiles(self.valid_ds, predictor, 200)
        forecast_df = refine_forecasts(forecast_df)
        y_true = self.valid_df.TARGET[-self.prediction_window:]
        quantile_loss = self.quantile_loss(y_true, forecast_df)

        # record inserting
        iter_record = pd.DataFrame(estimator_params, index=[self.internal_iter_num])
        iter_record['epochs'] = int(epochs)
        iter_record['batch_size'] = self.batch_size
        iter_record['batch_size'] = self.batch_size
        iter_record['learning_rate'] = round(self.learning_rate, 4)
        iter_record['quantile_loss'] = round(quantile_loss)

        self.internal_iter_num += 1
        self._records.append(iter_record)

        # As bayesian optimizer tries to maximize the target value
        # to make the model work, we need to inverse the sign so that it minimizes the loss
        return -quantile_loss

    # call this method when you actually tune
    def tune_model(self,
                   verbose: int = 2,
                   init_points: int = 4,
                   n_iter: int = 20,
                   saving_folder: Optional[str] = None,
                   skip_tune: bool = False,
                   **kwargs):
        best_params = None

        # when you want to tune the model
        if not skip_tune:
            deepAR = BayesianOptimization(f=self.model, pbounds=self.pbounds, verbose=verbose)
            deepAR.maximize(init_points=init_points, n_iter=n_iter)
            print('best_target_value:', -deepAR.max['target'])
            self._best_loss = -deepAR.max['target']

            trainer = Trainer(epochs=int(deepAR.max['params']['epochs']),
                              batch_size=self.batch_size,
                              ctx=self.ctx,
                              learning_rate=self.learning_rate)

            estimator = DeepAREstimator(deepAR.max['params'], trainer=trainer,
                                        freq='30min', prediction_length=self.prediction_window,
                                        cell_type='lstm', use_feat_dynamic_real=self.use_feat_dy
            best_params = deepAR.max["params"]

            predictor = estimator.train(training_data=self.train_ds)

            self._estimator = estimator
            self._predictor = predictor

        # when you do not want to tune the model but train with give parameter(**kwargs)
        else:
            print('Only training without tuning process...')
            best_params = kwargs
            trainer = Trainer(epochs=int(kwargs['epochs']),
                              batch_size=self.batch_size, ctx=self.ctx,
                              learning_rate=self.learning_rate)

            kwargs.pop('epochs', None)
            estimator = DeepAREstimator(**kwargs, trainer=trainer,
                                        freq='30min', prediction_length=self.prediction_window,
                                        cell_type='lstm', use_feat_dynamic_real=self.use_feat_dy
            predictor = estimator.train(training_data=self.train_ds)

            self._estimator = estimator
            self._predictor = predictor

        print("Evaluating on valid set...")
        forecast_df = self.forecast_quantiles(self.valid_ds, predictor, 200)
        forecast_df = refine_forecasts(forecast_df)

        y_true = self.valid_df.TARGET[-self.prediction_window:]

        quantile_sum = self.quantile_loss(y_true, forecast_df)

        print(f'The lowest sum of quantile loss for validation set is {round(quantile_sum, 4)}'
              f' with parameters {best_params}')

        # Model saving process
        if not saving_folder:
            curpath = os.getcwd()
            saving_folder = curpath + '/saved_model/model_' + str(round(quantile_sum, 4))

        else:
            saving_folder = saving_folder + '/model_' + str(round(quantile_sum, 4))

        # model saving
        try:
            print("Saving the model under " + saving_folder + " with records.")
            Path(saving_folder).mkdir(parents=True)
            self.current_saving_folder = saving_folder
            predictor.serialize(Path(saving_folder))
            record = self.return_records()
            record.to_csv(saving_folder + '/optimizer_record.csv')
            print("Successfully saved model.")

        except FileExistsError:
            warnings.warn(f"File or directory already exists in {saving_folder}")

        except:
            warnings.warn("Saving file failed due to unknown reason. "
                          "High probability of collision in predictor serialization is assumed")

        # to loads it back,
        # from gluonts.model.predictor import Predictor
        # predictor_deserialized = Predictor.deserialize(Path("/tmp/"))


    def predict_on_test(self, test_path: str,
                        **kwargs) -> pd.DataFrame:
        """

        Parameters
        ----------
         test_path : str => path of directory or folder containing all the test files

        Inner work
        Inner work
        ----------
         reads each test csv and converts it to ListDataset.
         # TODO: is the test_data set is correctly configured?

        Returns
        -------
         DataFrame containing quantile forecasts

        """

        #test_df = make_features(test_path)
        # if timestamped csv files under timestamped folder are not available,
        # it creates the new one using the test csv given from Dacon
        test_stamper = Timestamper(test_path=test_path)
        timestamped_path = test_stamper.stamp()
        all_quantile_forecasts = []

        for file_num in range(0, 81):
            current_file = f'/{file_num}.csv'
            test_df = pd.read_csv(timestamped_path + current_file)
            test_df = make_features(test_df)
            test_df['Timestamp'] = pd.to_datetime(test_df['Timestamp'])
            test_df = test_df.set_index(test_df['Timestamp'])

            test_range = pd.date_range(test_df.index[0], periods=48 * 9, freq='30min')
            test_df.index = test_range


            test_ds = ListDataset(
                [{"start": test_df.index[0],
                  "target": test_df.TARGET.values,
                  "feat_dynamic_real": [test_df.DHI.values, test_df.DNI.values, test_df.RH.value
                                        test_df['T'].values]
                  }],
                freq="30min",
                one_dim_target=True
            )

            forecast_df = self.forecast_quantiles(test_ds, self._predictor, 200)
            forecast_df = refine_forecasts(forecast_df)
            forecast_df = pd.DataFrame(forecast_df.values, columns=forecast_df.columns)

            all_quantile_forecasts.append(forecast_df)

        final = pd.concat(all_quantile_forecasts, axis=0)
        final[final < 0] = 0

        self.make_submission(final, **kwargs)

        return final

    def make_submission(self, target_df: pd.DataFrame,
                        sample_submission_file_path: Optional[str] = None) -> None:
        cur_path = os.getcwd()
        if not sample_submission_file_path:
            sample_submission_file_path = cur_path + '/sample_submission.csv'

        try:
            sample_sub = pd.read_csv(sample_submission_file_path, index_col=0)
            target_df.index = sample_sub.index
            target_df.columns = sample_sub.columns

        except:
            warnings.warn(f"Could not read sample submission file from {sample_submission_file_p

        try:
            target_df.to_csv(self.current_saving_folder)
            print("Submission file is successfully save into " + self.current_saving_folder)

        except:
            warnings.warn(f"Could not save the submission file to {self.current_saving_folder}")
```

코드 끝

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

df = pd.read_csv("/content/drive/MyDrive/Colab Notebooks/태양열 발전량 예측/Timestamped.csv")
df['Timestamp'] = pd.to_datetime(df['Timestamp'])
df = df.set_index(df['Timestamp'])

cut_edge = pd.to_datetime('2018-11-30 23:30:00') # 2년치 학습
temp = df[['TARGET', 'DHI', 'DNI', 'RH', 'T']]

valid_start = pd.to_datetime('2018-12-03 00:00:00')
valid_end = pd.to_datetime('2018-12-12 23:30:00')

train = temp[:cut_edge]
valid = temp[valid_start:valid_end]
```

```
pbounds = {'epochs': (50, 51),
            'context_length': (48 * 2, 48 * 2 + 30),
            'num_cells': (20, 60),
            'num_layers': (2, 5)}
```

```
params = {'epochs': 40,
            'context_length': 48 * 2,
            'num_cells': 40,
            'num_layers': 2}
```

```
saving_folder_path = '/content/drive/MyDrive/Colab Notebooks/태양열 발전량 예측/saved_model'
sample_submission_file = '/content/drive/MyDrive/Colab Notebooks/태양열 발전량 예측/sample_submi
```

```
test_file_path = '/content/drive/MyDrive/Colab Notebooks/태양열 발전량 예측/test'

tuner = DeepARTuner(train, valid, pbounds=pbounds, learning_rate=0.001, batch_size=64, use_feat_

tuner.tune_model(n_iter=10, saving_folder=saving_folder_path, skip_tune= False)
```

```
|   iter    |  target   | contex... |  epochs   | num_cells | num_la... |
-------------------------------------------------------------------------
```

```
print("₩nTuner now tries to predict on test_set...")
submission = tuner.predict_on_test(test_path=test_file_path,
                                   sample_submission_file_path=sample_submission_file)
```

```
submission.to_csv('/content/drive/MyDrive/Colab Notebooks/태양열 발전량 예측/submission.csv')
```