hgb_expanding_window

July 28, 2023

1 Customisable Prediction Accuracy

In this notebook, any specified model which adheres to the sklearn API can be fitted across year-long training windows starting in 2021, with predictions made and plotted for the final week of data. Models are persisted to disk for quickly repeatable runs.

[ ]:
```
import os
from joblib import dump, load
import numpy as np
import pandas as pd
import seaborn as sns
from tscv import GapRollForward
from tqdm.notebook import tqdm

# --- notebook parameters: import, choose model, set hyperparameters
from sklearn.ensemble import (RandomForestRegressor, GradientBoostingRegressor,
    AdaBoostRegressor, HistGradientBoostingRegressor)
from sklearn.neighbors import KNeighborsRegressor

MODEL_SELECTION = 'hgb'
MODELS_DEFINITION = {
    'rf': {
        'class': RandomForestRegressor,
        'kwargs': {
            'n_jobs': 8
        }
    },
    'gb': {
        'class': GradientBoostingRegressor,
        'kwargs': {}
    },
    'hgb': {
        'class': HistGradientBoostingRegressor,
        'kwargs': {} # capable of quantile loss, l2reg
    },
    'ada': {
        'class': AdaBoostRegressor,
        'kwargs': {}
```

```
    },
    'knn': {
        'class': KNeighborsRegressor,
        'kwargs': {}
    }
}
# --- end notebook parameters

MODEL = MODELS_DEFINITION[MODEL_SELECTION]['class']
MODEL_KWARGS = MODELS_DEFINITION[MODEL_SELECTION]['kwargs']
model_dir = f'../models/sa/{MODEL_SELECTION}'
if not os.path.isdir(model_dir):
    os.makedirs(model_dir)

# import and preprocess SA data
df = pd.read_csv(os.path.relpath('../data/merged_interpolated.csv'))
df.datetime = df.datetime.astype('datetime64')
dt = df['datetime'].dt
df['year'] = dt.year
df['month'] = dt.month
df['day'] = dt.day
df['hour'] = dt.hour
df['minute'] = dt.minute
df['day_of_week'] = dt.day_of_week
df['week'] = dt.isocalendar().week
X_inds = list(range(1, 8)) + list(range(11, 18))
y_ind = 9

df_2021 = df[df['year'] >= 2021]

# specify rolling training window strategy
obs_year = 48*365
obs_week = 48*7
tscv = GapRollForward(min_train_size=obs_week, #max_train_size=obs_year,
                      min_test_size=obs_week, max_test_size=obs_week,
                      roll_size=obs_week)
print(sum(1 for i in tscv.split(df_2021)), 'models to be loaded/trained')

# load persisted models if they exist, otherwise train and persist new models
models, training_weeks = [], []

for i, (train_ind, test_ind) in tqdm(enumerate(tscv.split(df_2021))):
    X_train, X_test = df_2021.iloc[train_ind, X_inds], df_2021.iloc[test_ind,␣
 ↪X_inds]
    y_train, y_test = df_2021.iloc[train_ind, y_ind], df_2021.iloc[test_ind,␣
 ↪y_ind]
```

```python
    # train or load
    begin, end = df_2021.iloc[[train_ind[0], train_ind[-1]], 0].dt.date
    argstring = '_'.join([f'{k}={v}' for k, v in MODEL_KWARGS.items()])
    model_filename = os.path.join(model_dir, f'{begin}_{end}_{argstring}.
 ↪joblib')
    try:
        model = load(model_filename)
    except FileNotFoundError:
        model = MODEL(**MODEL_KWARGS)
        model.fit(X_train, y_train)
        dump(model, model_filename)

    models.append(model)
    training_weeks.append((end - begin).days // 7)

# predict final week with each model
test_cutoff = df['datetime'].max() - pd.DateOffset(weeks=1)
final_week = df[df['datetime'] >= test_cutoff]

prdfs = []
for i, (model, weeks) in enumerate(zip(models, training_weeks)):
    prd = model.predict(final_week.iloc[:, X_inds])
    prdf = pd.DataFrame({'datetime': final_week['datetime'],
                        'model': i,
                        'train_end': end,
                        'training_weeks': weeks,
                        'predicted': prd,
                        'net_load': final_week['net_load']})
    prdfs.append(prdf)

predictions = pd.concat(prdfs)
predictions['residual'] = predictions['predicted'] - predictions['net_load']
predictions['pe'] = predictions['residual'] / predictions['net_load']
predictions['ape'] = predictions['pe'].abs()
prediction_summary = predictions.groupby('training_weeks').describe()

print(MODEL_SELECTION, 'MAPE:', predictions['ape'].mean())
```

```
112 models to be loaded/trained

0it [00:00, ?it/s]

hgb MAPE: 0.0898874219135926
```

[ ]:
```python
predictions.head()
```

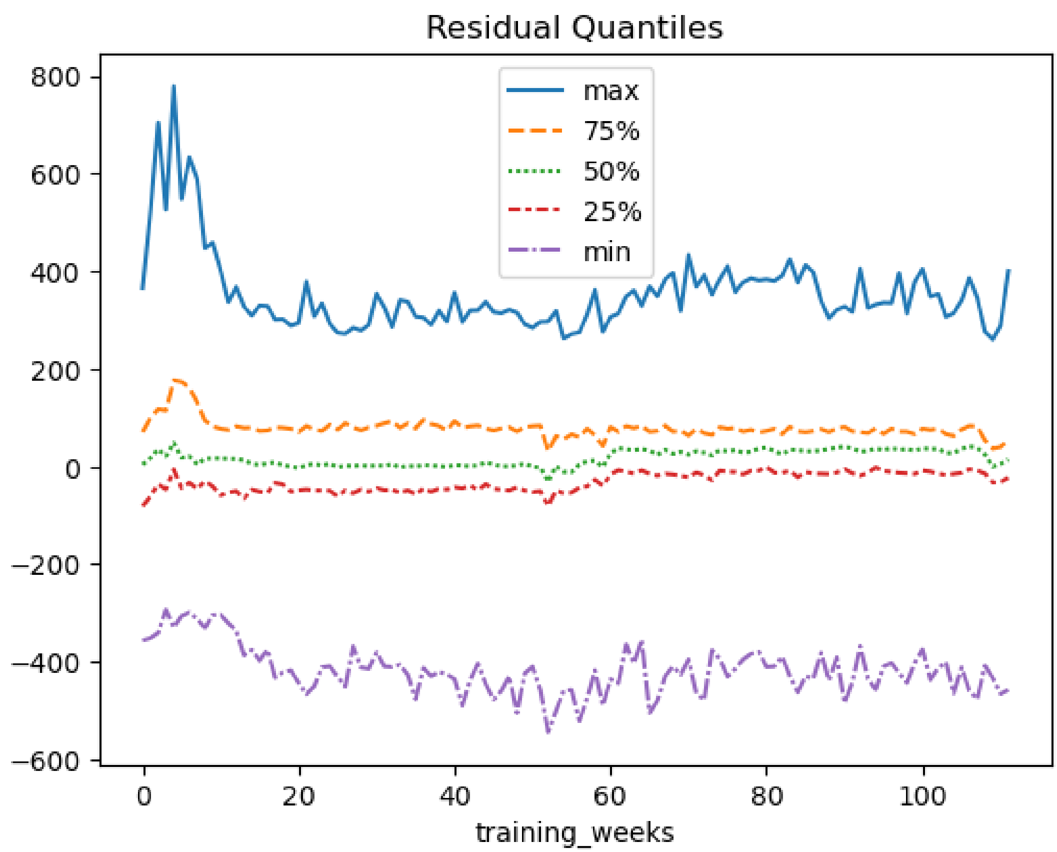
[ ]:
```
                 datetime  model   train_end  training_weeks    predicted  \
87389 2023-03-01 00:00:00      0  2023-02-23               0  1469.660225
```

```
87390 2023-03-01 00:30:00      0  2023-02-23                 0  1453.634305
87391 2023-03-01 01:00:00      0  2023-02-23                 0  1468.782400
87392 2023-03-01 01:30:00      0  2023-02-23                 0  1454.866741
87393 2023-03-01 02:00:00      0  2023-02-23                 0  1297.735285

       net_load   residual        pe       ape
87389      1402  67.660225  0.048260  0.048260
87390      1401  52.634305  0.037569  0.037569
87391      1412  56.782400  0.040214  0.040214
87392      1374  80.866741  0.058855  0.058855
87393      1315 -17.264715 -0.013129  0.013129
```

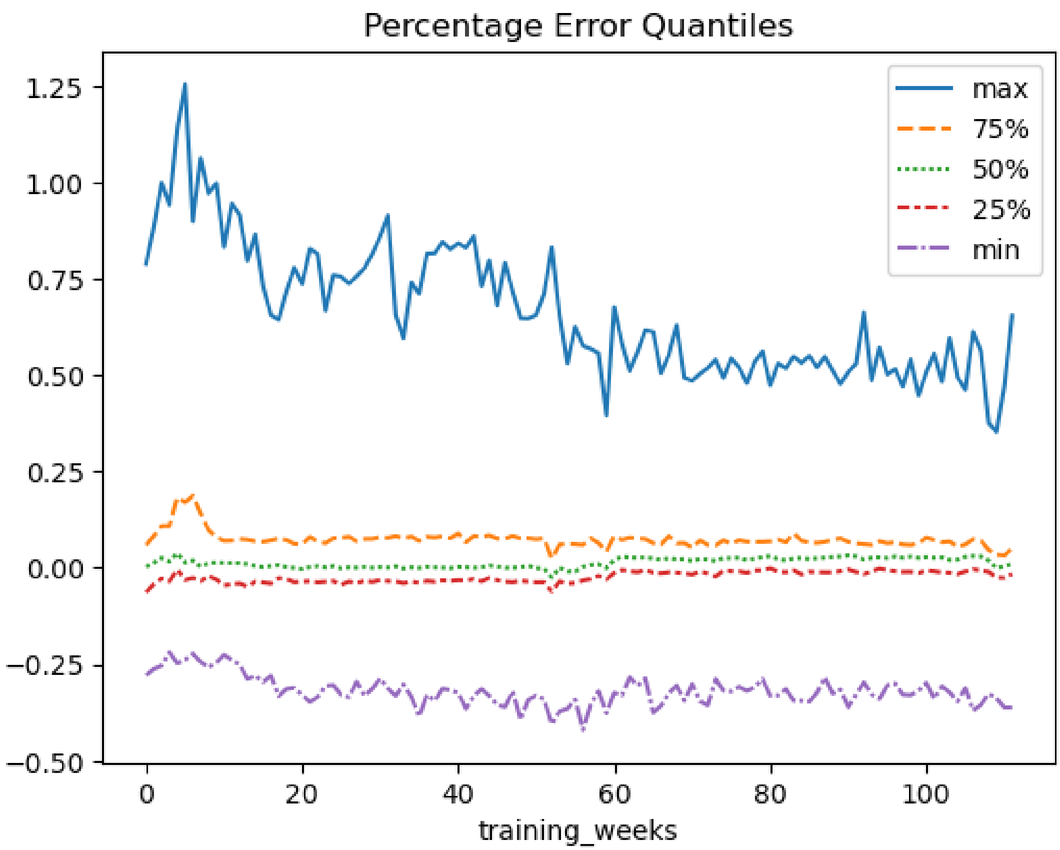
[ ]:
```
p = sns.lineplot(prediction_summary['residual'][['max', '75%', '50%', '25%',␣
 ↪'min']])
p.set(title='Residual Quantiles')
```

[ ]:
```
[Text(0.5, 1.0, 'Residual Quantiles')]
```



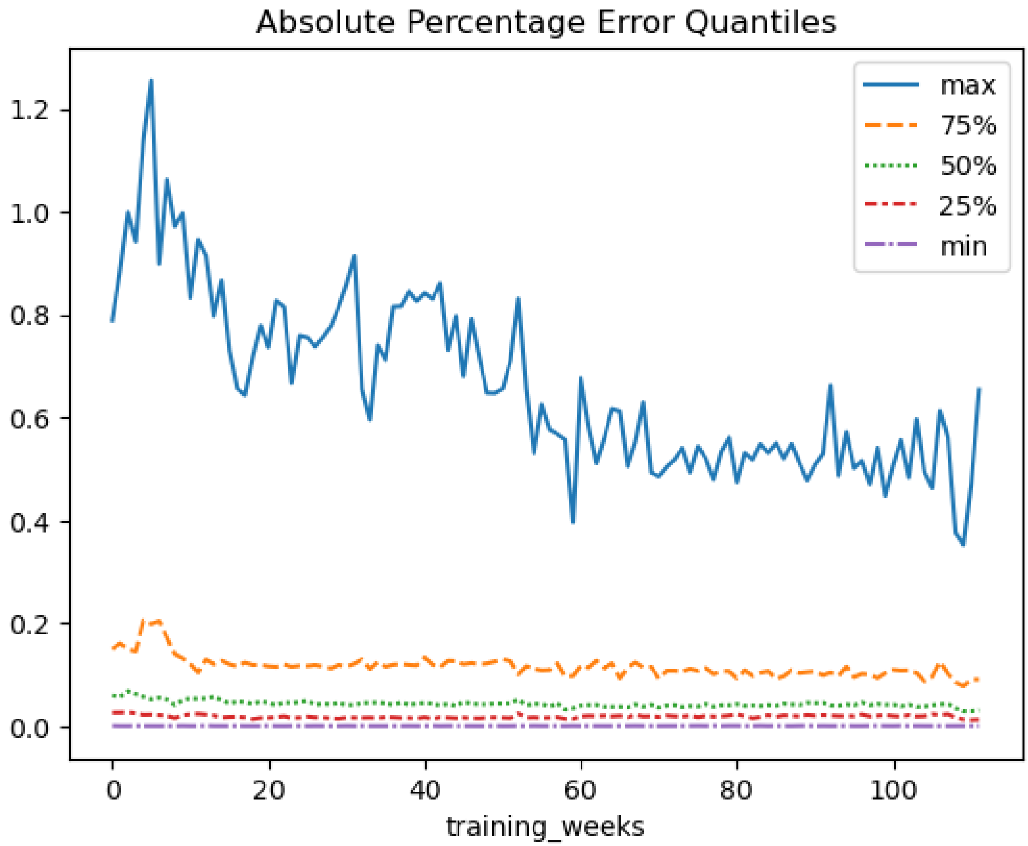
[ ]:
```
p2 = sns.lineplot(prediction_summary['pe'][['max', '75%', '50%', '25%', 'min']])
p2.set(title='Percentage Error Quantiles')
```

[ ]: [Text(0.5, 1.0, 'Percentage Error Quantiles')]



[ ]:
```
p2 = sns.lineplot(prediction_summary['ape'][['max', '75%', '50%', '25%',␣
    ↪'min']])
p2.set(title='Absolute Percentage Error Quantiles')
```

[ ]: [Text(0.5, 1.0, 'Absolute Percentage Error Quantiles')]


Absolute Percentage Error Quantiles
max
75%
50%
25%
min
1.2
1.0
0.8
0.6
0.4
0.2
0.0
0
20
40
60
80
100
training_weeks