WATER BAG TIME SERIE CLASSIFICATION - Preliminar Evaluation

Define Functions

Change project root directory

In [1]:

```
cd ../
```

```
C:\Users\luisr\Desktop\Repositories\Data Science Projects\Hackaton COR IV - Centro de Operações do
RJ\ACELERAÇÃO
```

Import modules and libraries

In [187]:

```
import os, json, pandas as pd, numpy as np, pickle
import matplotlib.pyplot as plt, seaborn as sns; sns.set()
from IPython.display import clear_output as co

#### Time serie features transformation pipeline & binary classification pipeline (Authoral)
from Modulos.timeserie_transform import TimeseriesTransformPipeline
from Modulos.imbalanced_selection import groupConsecutiveFlags, MinorityGroupSplitUndersample

#### Preprocessing & machine learning modules
from sklearn.preprocessing import MinMaxScaler as mms
from sklearn.model_selection import cross_validate, cross_val_predict
from imblearn.under_sampling import RandomUnderSampler
from imblearn.pipeline import Pipeline
from sklearn.utils import all_estimators
classifiers = dict(all_estimators('classifier'))
# from Modulos.cv_samplers import GroupUnderSampleSplit, print_cls_cnt

#### Metrics and scoring functions
from sklearn.metrics import (
    make_scorer, recall_score, precision_score,
    f1_score, precision_recall_fscore_support,
    classification_report as cr
)
recall_0 = make_scorer(recall_score, pos_label=0)
recall_1 = make_scorer(recall_score, pos_label=1)
precision_0 = make_scorer(precision_score, pos_label=0)
precision_1 = make_scorer(precision_score, pos_label=1)
f1_0 = make_scorer(f1_score, pos_label=0)
f1_1 = make_scorer(f1_score, pos_label=1)

scoring = {
    'accuracy': 'accuracy',
    'recall': 'recall', 'precision': 'precision',
    'recall-0': recall_0, 'recall-1': recall_1,
    'precision-0': precision_0, 'precision-1': precision_1,
    'f1-0': f1_0, 'f1-1': f1_1
}
```

Utility Funcitons

In [288]:

```
# Target selection and train/test split
def select_target(target_id, periods_ahead):
    print(f'Selected Target: {target_names[int(target_id)]} - id: {target_id}', '\n')

    # Select target
    Y = Yi[str(target_id)].loc[X.index].copy()
    display(Y.value_counts().to_frame('Target'))

    ### Target transformation
    if periods_ahead is not None:
        Y = (Y.rolling(periods_ahead, closed='left', min_periods=1).sum().shift(-periods_ahead + 1) > 0).astype(
'float')
        display(Y.value_counts().to_frame('Transformed Target'))

    # Group target positive class labels by being consecutive in time (group evaluation strategy)
    groups = groupConsecutiveFlags(ts=Y)

    return Y, groups

from sklearn.metrics import classification_report as cr, precision_recall_curve

# Classification report for test probabilities for given threshold
def clf_score(ye, yprob, threshold=0.5):
    yhat = (yprob > threshold).astype('int')
    scr = pd.DataFrame(cr(ye, yhat, digits=4, output_dict=True)).T
    return scr

# Precision-recall curve plot for test probabilities for given threshold
def precision_recall_plot(ye, yprob, thresh_lim=None, recall_lim=None):
    curve = pd.DataFrame(
        precision_recall_curve(ye, yprob, pos_label=1),
        index=['precision', 'recall', 'threshold']
    ).T.set_index('threshold').add_suffix(f' - 1')
    curve['f1 - 1'] = curve.mean(1)
    prec, rec = curve['precision - 1'], curve['recall - 1']
    curve['harmonic mean - 1'] = 2 * prec * rec / (prec + rec)
    fig, ax = plt.subplots(1, 2, figsize=(12, 3.5))
    curve.plot(ax=ax[0]); curve.reset_index().plot('recall - 1', ['precision - 1', 'f1 - 1', 'harmonic mean - 1'
], ax=ax[1])
    ax[0].set(title='Precision-Recall by Threshold', xlim=thresh_lim); ax[1].set(title='Precision-Recall Curve',
xlim=recall_lim)
    return ax
```

Load & Preprocess Data

In [4]:

```
from Modulos.waterbags import waterbag_project

project = waterbag_project(time_serie='clusters', freq='upsample', load_waterbags=True, time_features=True)

data = project.data.drop('index', axis=1)
Yi = project.time_serie
waterbags = project.waterbags

# Sample groups names per group label
target_names = waterbags.groupby(['sublabel', 'main_route']).first().index.to_frame().set_index('sublabel').to_d
ict()['main_route']
```

```
C:\Users\luisr\Desktop\Repositories\Data Science Projects\Hackaton COR IV - Centro de Operações do
RJ\ACELERAÇÃO\Modulos\waterbags.py:63: FutureWarning: pad is deprecated and will be removed in a fu
ture version. Use ffill instead.
  upsample = inmet.resample('15Min').pad()
```

1. Preprocessing & Data Transformation

Feature set

In [5]:

```
train_start, train_end = '2018-06', '2021-10'
eval_start, eval_end = '2021-11', '2022-04'
transform_args = dict(
    scale=True, interpolate='nearest', fillna='mean'
)

# Select feature set
X = TimeseriesTransformPipeline(
    data, train_start, cut=-1,
    drop_empty_cols=True,
    label_encode=data.columns[:11],
    **transform_args,
); X = X[: eval_end]

# Validation split
xt = X[:train_end]
xe = X[eval_start: eval_end]
```

```
Initial data: (437875, 241)
Time extraction: (142866, 241)
Drop empty columns:  (142865, 228)
```

Target variable

In [6]:

```
target_id = '1'
periods_ahead = 4
```

In [6]:

```
# Target validation split
yt = Y.loc[xt.index]
ye = Y.loc[xe.index]
groups_train = groups.loc[xt.index]
groups_eval = groups.loc[xe.index]
```

```
Selected Target: Rua do Catete - id: 1
```

	Target
0.0	136496
1.0	783

	Transformed Target
0.0	136282
1.0	997

2. Base line model

In [7]:

```
seed = 0
```

Fit and predict

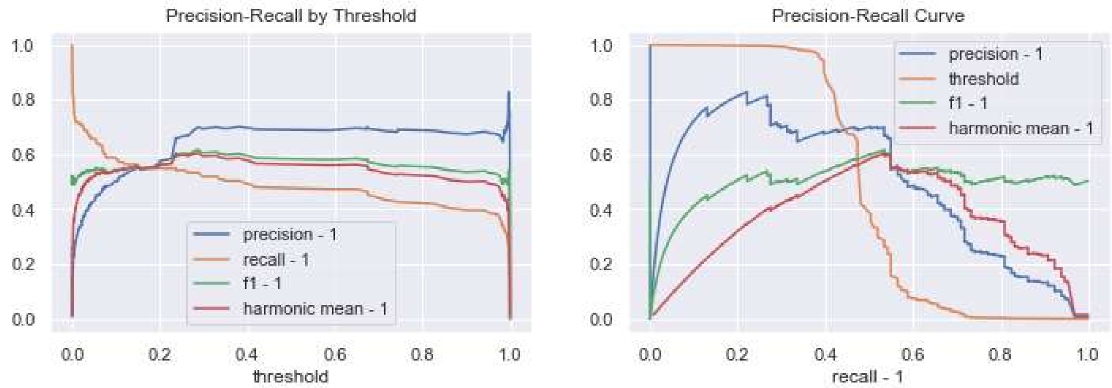
In [21]:

```
# Select specific classification model
gbc = classifiers['GradientBoostingClassifier']
model = gbc(n_estimators=100, random_state=0)

model.fit(xt, yt)
yprob = model.predict_proba(xe)[:, 1]
```

Evaluate predictions

In [36]:

```
ax = precision_recall_plot(ye, yprob)
scr = clf_score(ye, yprob, 0.50)
plt.show(); display(scr)
```



	precision	recall	f1-score	support
0.0	0.996008	0.998376	0.997191	17245.000000
1.0	0.688889	0.473282	0.561086	131.000000
accuracy	0.994418	0.994418	0.994418	0.994418
macro avg	0.842449	0.735829	0.779138	17376.000000
weighted avg	0.993693	0.994418	0.993903	17376.000000

3. Undersample pipeline

In [61]:

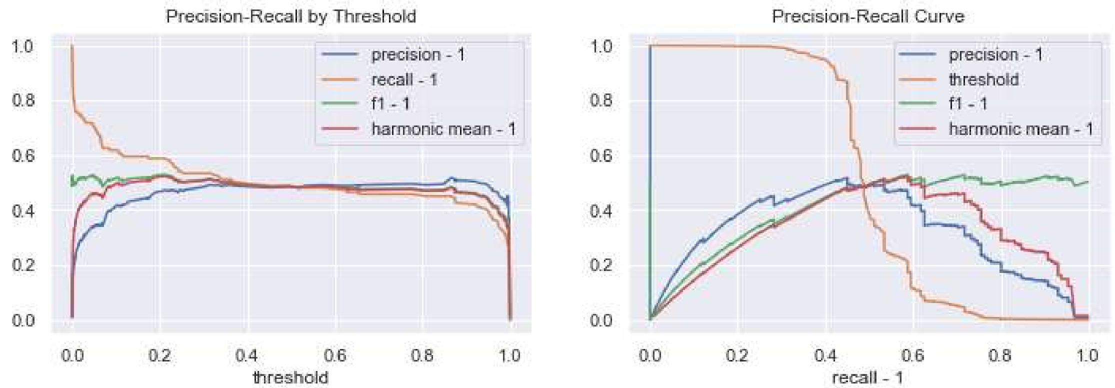
```
train_prct = 0.0075

rus = RandomUnderSampler(sampling_strategy=train_prct, random_state=seed)
model = gbc(n_estimators=100, random_state=0, verbose=1)

pipe = Pipeline([('under', rus), ('model', model)])

pipe.fit(xt, yt)
yprob = pipe.predict_proba(xe)[:, 1]
```

```
      Iter       Train Loss   Remaining Time
         1           0.0442            4.14m
         2           0.0421            4.11m
         3           0.0404            4.17m
         4           0.0357            4.24m
         5           0.0337            4.29m
         6           0.0326            4.22m
         7           0.0315            4.17m
         8           0.0305            4.14m
         9           0.0290            4.08m
        10           0.0279            4.03m
        20           0.0214            3.46m
        30           0.0173            3.01m
        40           0.0147            2.57m
        50           0.0129            2.14m
        60           0.0118            1.70m
        70           0.0108            1.27m
        80           0.0098           50.74s
        90           0.0092           25.35s
       100           0.0083            0.00s
```

Evaluate pipeline predictions

In [64]:

```
ax = precision_recall_plot(ye, yprob)
scr = clf_score(ye, yprob, 0.50)
plt.show(); display(scr)
```



	precision	recall	f1-score	support
0.0	0.996057	0.996057	0.996057	17245.000000
1.0	0.480916	0.480916	0.480916	131.000000
accuracy	0.992173	0.992173	0.992173	0.992173
macro avg	0.738486	0.738486	0.738486	17376.000000
weighted avg	0.992173	0.992173	0.992173	17376.000000

---

4. Minority group kfold cross validation

In [15]:

```
scr_cols = ['train_precision-1', 'train_recall-1', 'train_f1-1', 'test_precision-1', 'test_recall-1', 'test_f1-
1']
```

Undersample pipeline model

In [175]:

```
train_prct = 0.015
rus = RandomUnderSampler(sampling_strategy=train_prct, random_state=seed)

sgd = classifiers['SGDClassifier']
model = sgd(
    loss='hinge', penalty='l1',
    alpha=0.0001, l1_ratio=0.15,
    random_state=seed, verbose=0, n_jobs=-1,
)

pipe = Pipeline([('under', rus), ('model', model)])
```

Minority group split

In [176]:

```
# cross-validation split
splitter = MinorityGroupSplitUndersample(
    n_splits=10,
#     train_size=0.80, test_size=0.19, # Not used if split strategy is GroupKFold
    train_prct=None, test_prct='natural',
    random_state=seed,
); strategy='GroupKFold'

cv_group = splitter.split(xt, yt, groups_train, strategy)
```

Cross validation score

In [177]:

```
# evaluate splits
scr_group =  pd.DataFrame(cross_validate(
    pipe, xt, yt, groups=groups_train,
    scoring=scoring, cv=cv_group,
    n_jobs=-1, verbose=5,
    pre_dispatch='2*n_jobs',
    return_train_score=True,
))

scr_group[scr_cols].agg([np.mean, np.median, np.std])
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done   6 out of  10 | elapsed:   27.0s remaining:   18.0s
[Parallel(n_jobs=-1)]: Done  10 out of  10 | elapsed:   31.6s finished
```

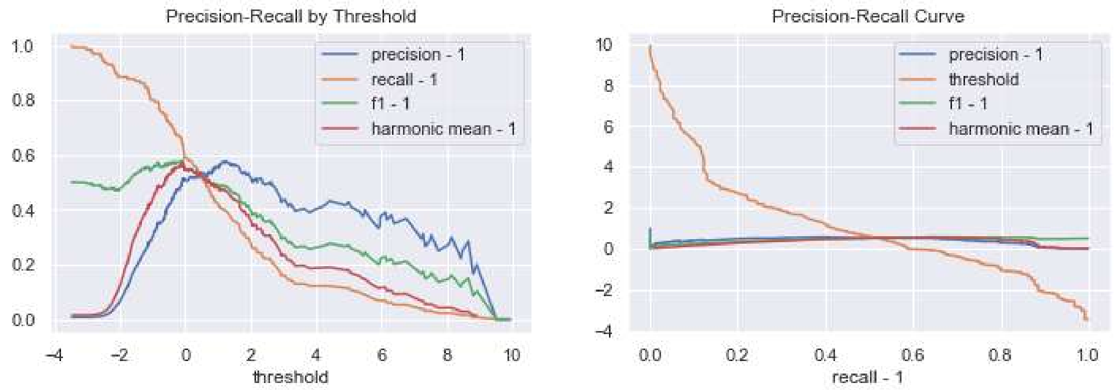
Out[177]:

	train_precision-1	train_recall-1	train_f1-1	test_precision-1	test_recall-1	test_f1-1
mean	0.725124	0.735854	0.728267	0.651341	0.644333	0.644053
median	0.729282	0.724359	0.726229	0.658396	0.635926	0.633295
std	0.042745	0.044770	0.012839	0.080212	0.127034	0.096337

Validation score

In [178]:

```
pipe.fit(xt, yt)
yprob = pipe.decision_function(xe)

ax = precision_recall_plot(ye, yprob)
scr = clf_score(ye, yprob, 0.50)
plt.show(); display(scr)
```



	precision	recall	f1-score	support
0.0	0.996347	0.996521	0.996434	17245.000000
1.0	0.531250	0.519084	0.525097	131.000000
accuracy	0.992921	0.992921	0.992921	0.992921
macro avg	0.763799	0.757802	0.760765	17376.000000
weighted avg	0.992841	0.992921	0.992881	17376.000000

Time based split cross-validation

In [449]:

```
from sklearn.model_selection import TimeSeriesSplit

period_mean = Y.resample('M').mean().to_frame('True Period Average - Normalized')
n_periods = period_mean.shape[0]

period_mean.iloc[:, 0] = mms().fit_transform(period_mean)
```

Undersample pipeline model

In [180]:

```
train_prct = 0.015
rus = RandomUnderSampler(sampling_strategy=train_prct, random_state=seed)

sgd = classifiers['SGDClassifier']
model = sgd(
    loss='hinge', penalty='l1',
    alpha=0.0001, l1_ratio=0.15,
    random_state=seed, verbose=0, n_jobs=-1,
)

pipe = Pipeline([('under', rus), ('model', model)])
```

Time series based split

In [443]:

```
n_splits = n_months
n_splits = 7

splitter = TimeSeriesSplit(n_splits=n_splits, test_size=None)

cv_time = list(splitter.split(X, Y))
```

Cross-validation score

In [444]:

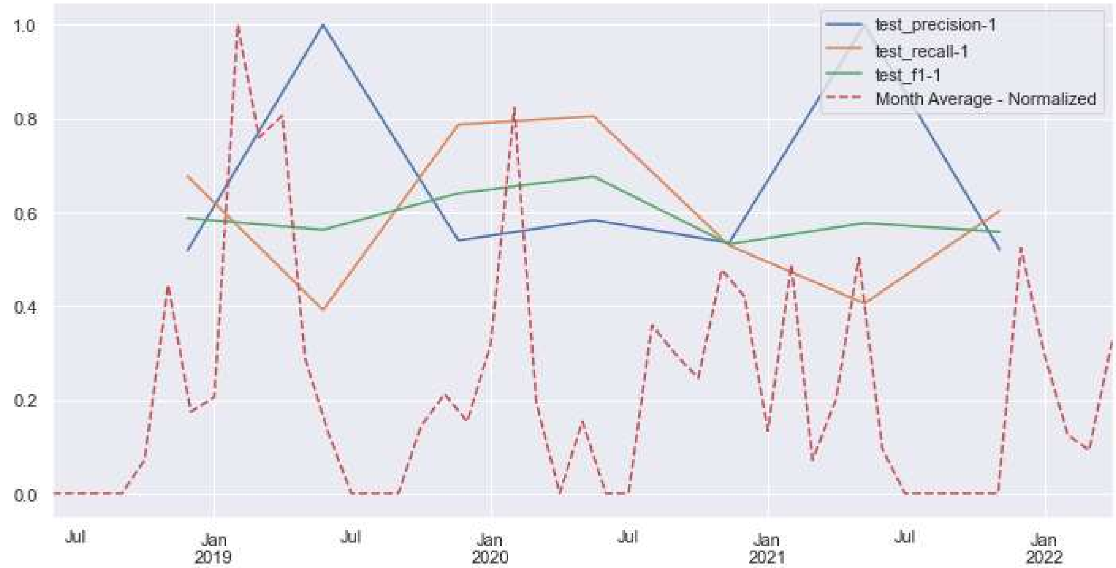
```
# evaluate splits
scr_time = pd.DataFrame(cross_validate(
    pipe, X.values, Y.values, # groups=groups_train,
    scoring=scoring, cv=cv_time,
    return_train_score=True,
    error_score=np.nan,
    n_jobs=-1, verbose=5,
    pre_dispatch='2*n_jobs',
))

# Reset score index by split test index start
test_start = []
for train, test in cv_time:
    test_start.append(X.iloc[test].index.min())
scr_time.index = test_start
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done   2 out of   7 | elapsed:   44.9s remaining:  1.9min
[Parallel(n_jobs=-1)]: Done   4 out of   7 | elapsed:   47.1s remaining:   35.3s
[Parallel(n_jobs=-1)]: Done   7 out of   7 | elapsed:   50.6s finished
```

Temporal performance visualization

In [445]:

```
fig, ax = plt.subplots(figsize=(12, 6))
scr_time[scr_cols[3:6]].plot(ax=ax)
ax = month_mean.plot(ax=ax, linestyle='--')
```



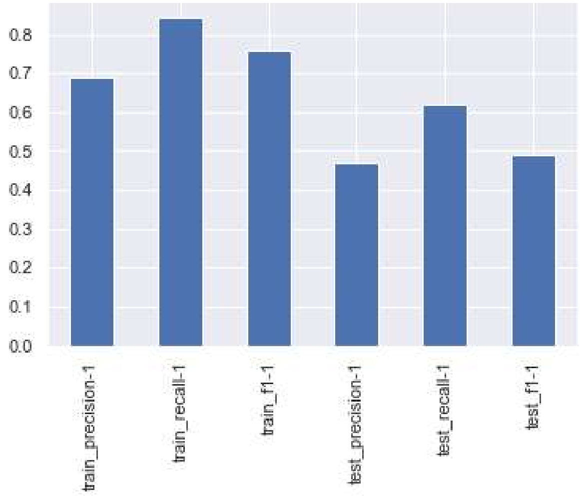
Average temporal performance

In [450]:

```
score[score['test_recall-1']!=0].mean()[scr_cols].plot.bar()
```

Out[450]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x218e1e60fd0>
```



Temporal split probability visualization

Time series based split

In [453]:

```
# n_splits = n_months

splitter = TimeSeriesSplit(n_splits=n_splits)
cv_time = list(splitter.split(X, Y))
```

Concatenated splits' probabilities

In [454]:

```
yprob_cv = []
for i, (train, test) in enumerate(cv_time):
    try:
        pipe.fit(X.iloc[train], Y.iloc[train])
        try: yprob = pipe.predict_proba(X.iloc[test])[:, 1]
        except: yprob = pipe.decision_function(X.iloc[test])
        yprob_cv.append(yprob)
    except Exception as e:
        yprob_cv.append(np.array([np.nan for i in range(len(test))]))
#          print('Error:', e)
    co(True); print(f'cv: {i+1}/{len(cv_time)}')

min_test_time = cv_time[0][1][0]
yprob_cv = pd.Series(np.concatenate(yprob_cv), index=X.index[min_test_time:]).dropna()
ye_cv = Y.loc[yprob_cv.index]
```

```
cv: 1/7
cv: 2/7
cv: 3/7
cv: 4/7
cv: 5/7
cv: 6/7
cv: 7/7
```

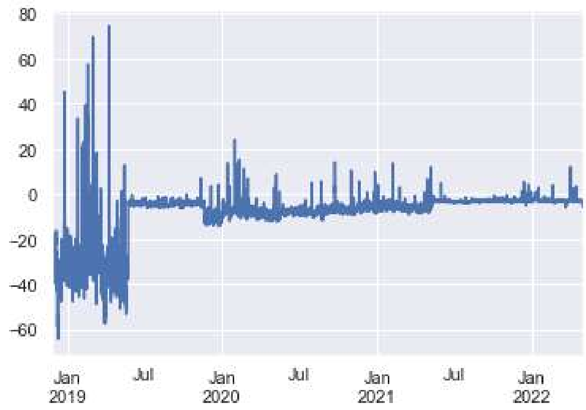
Probability time serie plot

In [455]:

```
yprob_cv.plot()
```

Out[455]:

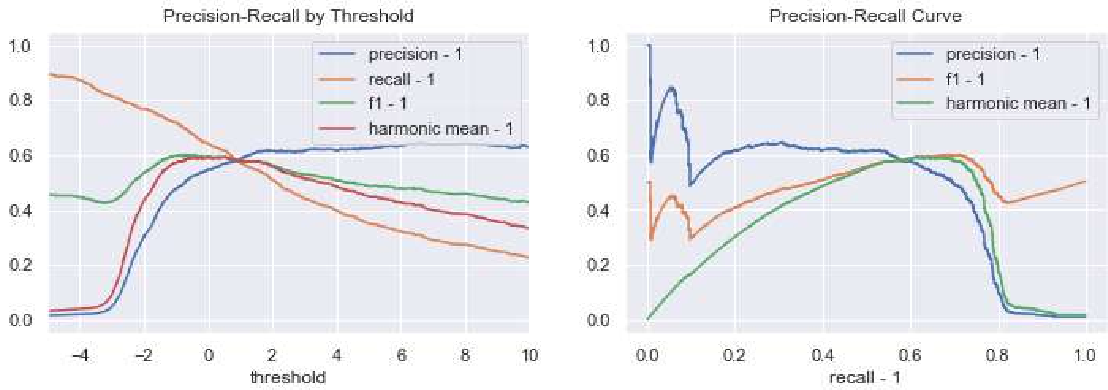
```
<matplotlib.axes._subplots.AxesSubplot at 0x218e363f3a0>
```



Threshold-moving evaluation

In [456]:

```
ax = precision_recall_plot(ye_cv, yprob_cv, thresh_lim=(-5, 10))
scr = clf_score(ye_cv, yprob_cv, 0.50)
plt.show(); display(scr)
```



	precision	recall	f1-score	support
0.0	0.996952	0.996291	0.996621	119165.000000
1.0	0.569620	0.617089	0.592405	948.000000
accuracy	0.993298	0.993298	0.993298	0.993298
macro avg	0.783286	0.806690	0.794513	120113.000000
weighted avg	0.993579	0.993298	0.993431	120113.000000

Temporal performance

Score per period - Single or cumulative

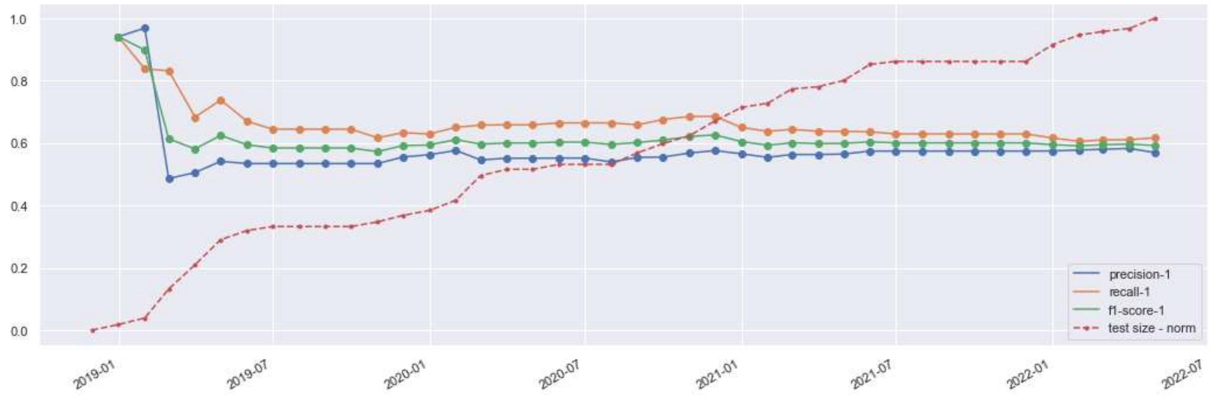
In [458]:

```
cumulative = True
threshold = 0.50
dates = pd.date_range(yprob_cv.index.min(), yprob_cv.index.max(), freq='M')

scr_names = ['precision-1', 'recall-1', 'f1-score-1', 'support-1']
labels = ['0.0', '1.0']

scrs_date = []
for date in dates.strftime('%Y-%m'):
    yprob_bin = yprob_cv[:date] if cumulative else yprob_cv.loc[date]
    ye_bin = ye_cv.loc[yprob_bin.index]
    scr = clf_score(ye_bin, yprob_bin, threshold)
    for label in labels:
        if label not in scr.index: scr.loc[label] = [np.nan, np.nan, np.nan, 0.0]
    scr_flat = pd.concat([scr.loc[label].add_suffix('-'+label[0]) for label in labels])
    scrs_date.append(scr_flat)
scrs_date = pd.DataFrame(scrs_date, index=dates)
```

Cumulative score per period

In [460]:

```
support_norm = pd.Series(mms().fit_transform(scrs_date[[scr_names[-1]]]).reshape(-1), index=scrs_date.index).to_
frame('test size - norm')

ax = scrs_date[scr_names[:-1]].iloc[:].plot(figsize=(18, 6), marker='o')
ax = support_norm.plot(ax=ax, ls='--', marker='.')
```



Probability windows

In [573]:

```
def groups_windows(groups, spread=24, freq=pd.Timedelta(15, 'min')):
    windows = []; wide = spread * freq
    for group in groups.unique():
        group_index = groups.index[groups==group]
        grp_min, grp_max = group_index.min(), group_index.max()
        windows.append((grp_min - wide, grp_max + wide))
    return windows

def window_proba(ye, yprob, time_lim, ax=None):
    yprob = pd.Series(mms().fit_transform(yprob.to_frame()).reshape(-1), index=yprob.index) # scale probability
 to 0-1 range
    msk = ye.index.to_series().between(*time_lim) # time window limits
    if ax is None: ax = plt.axes()
    yprob[msk].plot(ax=ax)
    ax = ye[msk].plot(ax=ax)
    return ax
```

Group based

In [ ]:

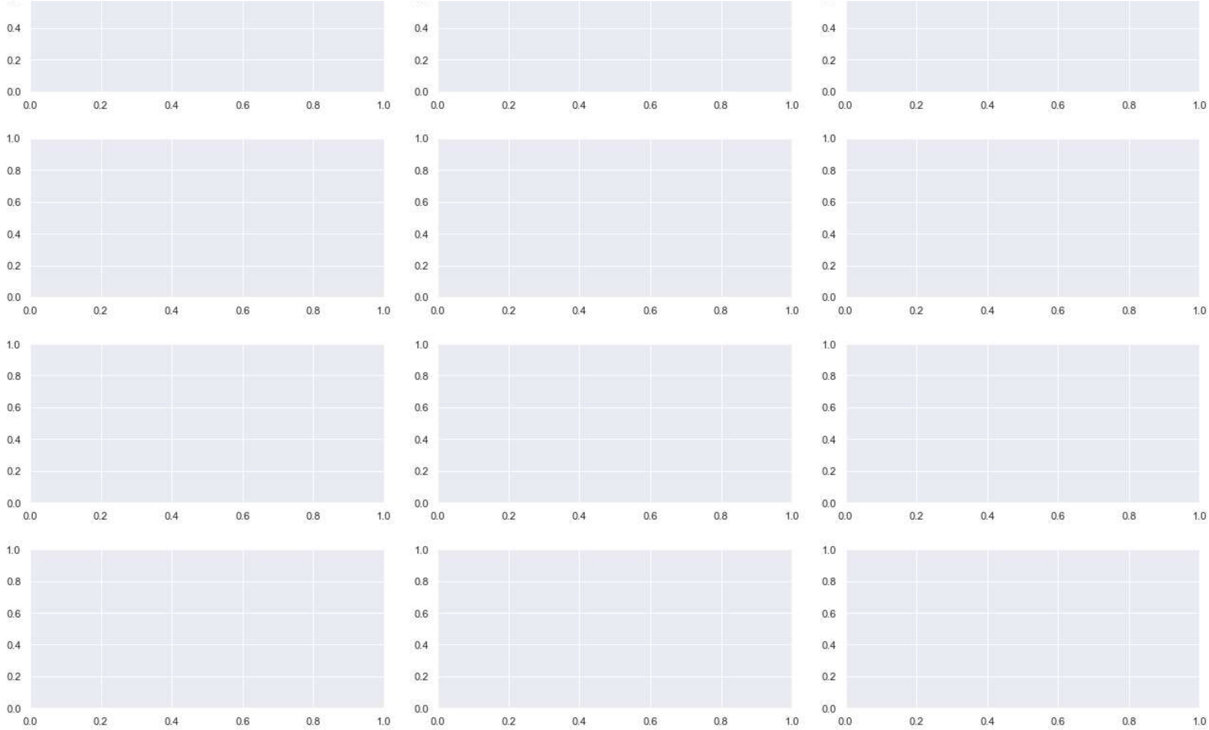
```

```

In [ ]:

```
windows
```

In [581]:

```
yprob = yprob_cv.copy()
yprob = pd.Series(mms().fit_transform(yprob.to_frame()).reshape(-1), index=yprob.index) # scale probability to 0
-1 range
ye = ye_cv.copy()
n_cols = 3

windows =  groups_windows(groups, spread=24, freq=pd.Timedelta(15, 'min'))

n_plots = len(windows)
n_rows = int(n_plots / n_cols if n_plots % n_cols == 0 else n_plots // n_cols + 1)
figsize = (6 * n_cols, 3 * n_rows)

fig, axs = plt.subplots(n_rows, n_cols, figsize=figsize, tight_layout=True)
axs = list(axs.reshape(-1))

for ax, time_lim in zip(axs, windows[:3]):
    msk = ye.index.to_series().between(time_lim[0], time_lim[1]) # time window limits
    yprob[msk].plot(ax=ax)
    ye[msk].plot(ax=ax)
#     window_proba(ye, yprob, time_lim, ax)

plt.show()
```



Learning curve

In [ ]: