4 - Program Performance

September 14, 2022

1 Rule Search Real Time Program Delivery Performance

Copyright (C) 2021 Geoffrey Guy Messier

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation, either version 3 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program. If not, see <https://www.gnu.org/licenses/>.

[1]:
```
%load_ext autoreload
%autoreload 1
```

[2]:
```
import numpy as np
import pandas as pd
import datetime, copy, imp
import time
import os
import re
import dask.dataframe as dd

from tqdm.auto import tqdm, trange
from tqdm.notebook import tqdm
tqdm.pandas()

from pandas.tseries.offsets import DateOffset

import sys
sys.path.insert(0, '../util/')

%aimport di_data
```

```
%aimport data_cache

from di_data import *
from data_cache import CacheResult
```

[3]:
```
#dataDirStr = '/hd2/data/di/plwh/'
#cacheDirStr = '/hd2/data/di/plwh/cache/'

dataDirStr = '/Users/gmessier/data/plwh/'
cacheDirStr = '/Users/gmessier/data/plwh/cache/'
```

1.1 Define Rules

```
WinSz: 30, ['A0 >= 28' 'A3 < 0.5']
WinSz: 60, ['A0 >= 54' 'A1 < 10.5']
WinSz: 90, ['A0 >= 78' 'A3 < 3.5']
WinSz: 120, ['A0 >= 99' 'A3 < 4.5']

0: EntrySleep
1: EntryConsl
2: EntryLog
3: EntryBar
4: Addiction
5: Bar
6: PhysicalHealth
7: MentalHealth
8: Violence
9: PoliceJustice
10: Conflict
11: EMS
12: Supports
13: Overdose
```

[4]:
```
rules = {
    30: [ { 'Ftr': 'EntrySleep', 'Thsh': 28, 'Op': '>=' }, { 'Ftr': 'EntryBar',␣
 ↪'Thsh': 0.5, 'Op': '<' } ],
    60: [ { 'Ftr': 'EntrySleep', 'Thsh': 54, 'Op': '>=' }, { 'Ftr':␣
 ↪'EntryConsl', 'Thsh': 10.5, 'Op': '<' } ],
    90: [ { 'Ftr': 'EntrySleep', 'Thsh': 78, 'Op': '>=' }, { 'Ftr': 'EntryBar',␣
 ↪'Thsh': 3.5, 'Op': '<' } ],
    120: [ { 'Ftr': 'EntrySleep', 'Thsh': 99, 'Op': '>=' }, { 'Ftr':␣
 ↪'EntryBar', 'Thsh': 4.5, 'Op': '<' } ]
}
```

[ ]:

1.2 Load Data

[6]:
```
#tbl = pd.read_hdf('/hd2/data/di/plwh/cache/DiRules-LoadRawData___.hd5')
tbl = pd.read_hdf('/Users/gmessier/data/plwh/cache/DiRules-LoadRawData___.hd5')
```

[7]:
```
descFeatures = [ 'ClientId', 'Date' ]
ruleFeatures = [ 'EntrySleep', 'EntryConsl', 'EntryBar' ]
tbl = tbl[descFeatures + ruleFeatures]
```

[8]:
```
#tteChr = pd.read_hdf('/hd2/data/di/plwh/cache/DiRules-CdnFedChronicTte___.hd5')
tteChr = pd.read_hdf('/Users/gmessier/data/plwh/cache/
 ↪DiRules-CdnFedChronicTte___.hd5')
```

[9]:
```
# Only consider clients that have a label.
tbl = tbl.loc[tbl.ClientId.isin(tteChr.index)]
```

[10]:
```
tbl
```

[10]:

	ClientId	Date	EntrySleep	EntryConsl	EntryBar
4801	34334	2008-11-09 00:00:00.000	0	0	1
14019	12902	2009-10-06 07:28:54.640	0	0	1
16463	31200	2012-11-20 00:00:00.000	0	0	1
18600	55304	2008-11-09 00:00:00.000	0	0	1
18787	30382	2008-07-01 00:00:00.000	0	0	1
…	…	…	…	…	…
5576428	2528038	2019-11-17 00:00:00.000	1	0	0
5576429	2532949	2019-11-17 00:00:00.000	1	0	0
5576430	2495818	2019-11-17 00:00:00.000	1	0	0
5576431	27906	2019-11-17 00:00:00.000	1	0	0
5576432	18559	2019-11-17 00:00:00.000	1	0	0

[5025192 rows x 5 columns]

[ ]:
```

```

1.3 Review Clients Monthly

[11]:
```
absentThrsh = 30 # days absent to be no longer active

def apply_rule(smry,rules):

    detected = True

    for iR in range(len(rules)):
        if rules[iR]['Op'] == '>=':
            detected &= smry[rules[iR]['Ftr']] >= rules[iR]['Thsh']
        else:
            detected &= smry[rules[iR]['Ftr']] < rules[iR]['Thsh']
```

```
    return detected


def review_client_monthly(tbl,clientId,winSzDays,rule,tte):
#def review_client_monthly(tbl,winSzDays,rule,tte):
#    clientId = tbl.iloc[0].ClientId

    # Determine the client's first review date.
    startDate = tbl.loc[tbl.ClientId == clientId,'Date'].min()
    fedDefDate = tte.loc[clientId].Date

    # The client is evaluated on the first meeting date where they have been in␣
↪shelter at
    # least winSzDays.
    evalDate = pd.to_datetime('{}{:02d}01'.format(startDate.year,startDate.
↪month)) + pd.DateOffset(months=1)
    while (evalDate-startDate).days < winSzDays:
        evalDate += pd.DateOffset(months=1)

    # Resample so that muliple data entries in each category that occur on a␣
↪particular day are counted only once.
    events = tbl.loc[tbl.ClientId == clientId].resample('D',on='Date').max()

    tti = (fedDefDate - startDate).days
    ruleDetect = False
    chronic = tte.loc[clientId].Flag == 'chr'

    while evalDate <= fedDefDate:

        # Add up all the events that have occurred in our window.
        smry = events.loc[ (events.Date <= evalDate) & (events.Date >=␣
↪evalDate-pd.DateOffset(days=winSzDays)) ][ruleFeatures].sum()

        # Ensure the client is active at this meeting.
        #print(( evalDate - events.loc[ events.Date <= evalDate ].Date.max() ).
↪days)
        isActive = ( evalDate - events.loc[ events.Date <= evalDate ].Date.
↪max() ).days <= absentThrsh

        if  isActive & apply_rule(smry,rule):

            tti = (evalDate - startDate).days
            ruleDetect = True
            break
```

```
        else:
            evalDate += pd.DateOffset(months=1)

    return { 'ClientId': clientId, 'RuleDetect': ruleDetect, 'Chronic':␣
↪chronic, 'TTI': tti }
```

[12]:
```
#review_client_monthly(tbl,9548,30,rules[30],tteChr)
#review_client_monthly(tbl,9544,30,rules[30],tteChr)
```

[ ]:
```
```

[13]:
```
from dask.distributed import Client

client = Client("tcp://127.0.0.1:51550")
client
```

[13]:
```
<Client: 'tcp://127.0.0.1:51550' processes=4 threads=8, memory=16.00 GiB>
```

[15]:
```
@CacheResult
def review_clients_monthly_parallel(tbl,winSzDays,rule,tte,client):

    tblFtr = client.scatter(tbl,broadcast=True)
    tteFtr = client.scatter(tte,broadcast=True)

    futures = []
    for clientId in tbl.ClientId.unique():
        futures += [ client.
↪submit(review_client_monthly,tblFtr,clientId,winSzDays,rule,tteFtr) ]



    res = pd.DataFrame(client.gather(futures))

    return res.set_index('ClientId')
```

[16]:
```
rdet30 =␣
↪review_clients_monthly_parallel(tbl,30,rules[30],tteChr,client,path=cacheDirStr,filename='D
↪hd5')
```

[17]:
```
rdet60 =␣
↪review_clients_monthly_parallel(tbl,60,rules[60],tteChr,client,path=cacheDirStr,filename='D
↪hd5')
```

[18]:
```
rdet90 =␣
↪review_clients_monthly_parallel(tbl,90,rules[90],tteChr,client,path=cacheDirStr,filename='D
↪hd5')
```

[19]:
```
rdet120 =␣
↪review_clients_monthly_parallel(tbl,120,rules[120],tteChr,client,path=cacheDirStr,filename=
↪hd5')
```

[20]:
```
rdet30
```

[20]:
```
           RuleDetect  Chronic   TTI
ClientId
34334           False     True   360
12902           False    False  2892
31200           False    False  3770
55304           False    False     0
30382            True     True   153
…                …       …    …
41377           False    False     0
2533500         False    False     0
58859           False    False     1
12792           False    False     0
2531483         False    False     0

[32346 rows x 3 columns]
```

[26]:
```
tteChr.loc[30382]
```

[26]:
```
Flag                    chr
Date    2009-03-10 00:00:00
Time                    252
Name: 30382, dtype: object

distributed.client - ERROR - Failed to reconnect to scheduler after 30.00
seconds, closing client
_GatheringFuture exception was never retrieved
future: <_GatheringFuture finished exception=CancelledError()>
asyncio.exceptions.CancelledError
```

[ ]:
```

```

[29]:
```
~rdet30.Chronic
```

[29]:
```
ClientId
34334      False
12902       True
31200       True
55304       True
30382      False
           …
41377       True
2533500     True
```

```
58859       True
12792       True
2531483     True
Name: Chronic, Length: 32346, dtype: bool
```

[33]:
```
def eval_rule_performance(rdet):
    nPos = rdet.Chronic.sum()
    recall = (rdet.Chronic & rdet.RuleDetect).sum()/nPos

    nTestPos = rdet.RuleDetect.sum()
    precision = (rdet.Chronic & rdet.RuleDetect).sum()/nTestPos

    ttiMed = rdet[rdet.Chronic].TTI.median()

    tPos = (rdet.Chronic & rdet.RuleDetect).sum()
    fNeg = (rdet.Chronic & ~rdet.RuleDetect).sum()
    fPos = (~rdet.Chronic & rdet.RuleDetect).sum()
    tNeg = (~rdet.Chronic & ~rdet.RuleDetect).sum()



    return (recall,precision,ttiMed,tPos,fNeg,fPos,tNeg)
```

[38]:
```
ws = { 30, 60, 90, 120 }
rdet = { 30: rdet30, 60: rdet60, 90: rdet90, 120: rdet120 }

for w in ws:
    (recall,precision,ttiMed,tPos,fNeg,fPos,tNeg) =␣
 ↪eval_rule_performance(rdet[w])

    print(f'WinSize: {w}')
    print(f' Recall: {recall:.2f}, Precision: {precision:.2f}, PosRate:␣
 ↪{(tPos+fPos)/(tPos+fPos+tNeg+fNeg):.2f}, TTI(median): {ttiMed:.2f}')
    print(f' True Pos: {tPos}/{tPos+fNeg}')
    print(f' False Neg: {fNeg}/{tPos+fNeg}')
    print(f' False Pos: {fPos}/{fPos+tNeg}')
    print(f' True Neg: {tNeg}/{fPos+tNeg}')
    print(f'{w} & {recall:.2f} & {precision:.2f} & {tPos} & {fNeg} & {fPos} &␣
 ↪{tNeg} & {ttiMed:.2f} \\\\')

    print('')
```

```
WinSize: 120
 Recall: 0.73, Precision: 0.83, PosRate: 0.09, TTI(median): 272.00
```

```
 True Pos: 2338/3191
 False Neg: 853/3191
 False Pos: 465/29155
 True Neg: 28690/29155
120 & 0.73 & 0.83 & 2338 & 853 & 465 & 28690 & 272.00 \\

WinSize: 90
 Recall: 0.75, Precision: 0.78, PosRate: 0.10, TTI(median): 254.00
 True Pos: 2409/3191
 False Neg: 782/3191
 False Pos: 680/29155
 True Neg: 28475/29155
90 & 0.75 & 0.78 & 2409 & 782 & 680 & 28475 & 254.00 \\

WinSize: 60
 Recall: 0.77, Precision: 0.71, PosRate: 0.11, TTI(median): 225.00
 True Pos: 2452/3191
 False Neg: 739/3191
 False Pos: 991/29155
 True Neg: 28164/29155
60 & 0.77 & 0.71 & 2452 & 739 & 991 & 28164 & 225.00 \\

WinSize: 30
 Recall: 0.85, Precision: 0.60, PosRate: 0.14, TTI(median): 162.00
 True Pos: 2726/3191
 False Neg: 465/3191
 False Pos: 1792/29155
 True Neg: 27363/29155
30 & 0.85 & 0.60 & 2726 & 465 & 1792 & 27363 & 162.00 \\
```

[23]:
```
tteChr.loc[tteChr.Flag == 'chr'].Time.median()
```

[23]:
```
297.0
```

[ ]:
```

```

[49]:
```
def summarize_shelter_access(acs):

    longFields = { 'TotalStays': 'Total Stays', 'TotalEpisodes': 'Total␣
 ↪Episodes', 'Tenure': 'Tenure (days)',
                   'UsagePct': 'Usage Percentage', 'AvgGapLen': 'Average Gap␣
 ↪Length (days)' }

    fields = [ 'TotalStays', 'TotalEpisodes', 'Tenure', 'UsagePct', 'AvgGapLen'␣
 ↪]
    for field in fields:
```

```
        print('%s:' % (field))
        nEntry = sum(~np.isnan(acs[field]))
        print(' Avg: {:.1f}, Med: {:.1f}, 10thPct: {:.1f}, 90thPct: {:.1f}'
              .format(acs[field].mean(),acs[field].median(),
                      acs[field].sort_values().iloc[int(nEntry*0.1)],
                      acs[field].sort_values().iloc[int(nEntry*0.9)]))


    print()
    for field in fields:
        nEntry = sum(~np.isnan(acs[field]))
        print('{} & {:.1f} & {:.1f} & {:.1f} & {:.1f}\\\\'
              .format(longFields[field],acs[field].mean(),acs[field].median(),
                      acs[field].sort_values().iloc[int(nEntry*0.1)],
                      acs[field].sort_values().iloc[int(nEntry*0.9)]))
```

[50]:
```
accs = pd.read_hdf('/Users/gmessier/data/plwh/cache/DiRules-AccessStatistics___.
↪hd5')
```

[51]:
```
fPosInds = rdet30.loc[rdet30.RuleDetect & ~rdet30.Chronic].index
```

[52]:
```
summarize_shelter_access(accs.loc[accs.index.isin(fPosInds)])
```

```
TotalStays:
 Avg: 186.3, Med: 153.0, 10thPct: 72.0, 90thPct: 350.0
TotalEpisodes:
 Avg: 6.1, Med: 4.0, 10thPct: 1.0, 90thPct: 14.0
Tenure:
 Avg: 1493.0, Med: 1210.0, 10thPct: 122.0, 90thPct: 3410.0
UsagePct:
 Avg: 30.0, Med: 16.1, 10thPct: 5.0, 90thPct: 86.0
AvgGapLen:
 Avg: 8.9, Med: 6.2, 10thPct: 1.1, 90thPct: 20.1

Total Stays & 186.3 & 153.0 & 72.0 & 350.0\\
Total Episodes & 6.1 & 4.0 & 1.0 & 14.0\\
Tenure (days) & 1493.0 & 1210.0 & 122.0 & 3410.0\\
Usage Percentage & 30.0 & 16.1 & 5.0 & 86.0\\
Average Gap Length (days) & 8.9 & 6.2 & 1.1 & 20.1\\
```

[ ]:

[ ]:

[ ]:

[ ]: