Client History Algorithm Comparison

May 31, 2021

1 Comparing Chronic Homelessness Prediction Algorithms using Client Histories

This notebook compares algorithms for predicting chronic homelessness using both classification metrics and an examination of the shelter access histories of the cohorts selected as chronic.

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

[6]:
```
import numpy as np
import pandas as pd
import datetime, copy, imp
import time
import matplotlib.pyplot as plt

from sksurv.nonparametric import kaplan_meier_estimator
from sksurv.linear_model import CoxPHSurvivalAnalysis
from sksurv.metrics import concordance_index_censored
from sklearn import metrics

from sklearn.neural_network import MLPClassifier
from sklearn.linear_model import LogisticRegression
```

```
from sklearn.model_selection import StratifiedKFold, RepeatedStratifiedKFold,␣
↪GridSearchCV

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

%aimport di_data
from di_data import *
```

1.0.1 Pre-Processing

[3]:
```
dirStr = '~/data/plwh/'
```

[4]:
```
def PreProcess():

    tblAll = pd.read_hdf(dirStr + 'UniversityExportAnonymized.hd5')

    tblAll = tblAll[tblAll.Date >= pd.to_datetime('2007-07-01')]

    print('Total Entries: {}'.format(len(tblAll.index)))
    print('Dates: ',min(tblAll.Date),' to ',max(tblAll.Date))

    tbl = copy.deepcopy(tblAll[ [ 'ClientId', 'Date', 'EntryType', 'Age' ] ])
    tbl['Police'] = (tblAll.PoliceLogFlag == 1) | (tblAll.CPS > 0)
    tbl['Ems'] = (tblAll.EmsLogFlag == 1) | (tblAll.EMS > 0)
    tbl['Health'] = (tblAll.Health > 0) | (tblAll.PhysicalHealth > 0) | (tblAll.
↪MentalHealth > 0) | (tblAll.Medication > 0)
    tbl['Violence'] = (tblAll.PhysicalViolence > 0) | (tblAll.Weapon > 0) |␣
↪(tblAll.Spray > 0) | (tblAll.Brawl > 0) | (tblAll.Gun > 0) | (tblAll.Knife >␣
↪0)
    tbl['Addiction'] = (tblAll.Addiction > 0) | (tblAll.Overdose > 0)

    # To address left censoring: Remove all clients with first sleep date␣
↪within a year of the 2008 data import.
    leftStart = tbl.Date.min()
    leftEnd = pd.to_datetime('2009-07-01')

    # To address right censoring: Remove all clients with first sleep date␣
↪within approximately 2 years of the end
    # of the data.  Reasoning: We want to allow a 2 year window to give the␣
↪clients a chance to become chronic.
    rightStart = pd.to_datetime('2018-01-20')
    rightEnd = tbl.Date.max()

    nClientsAll = len(tbl.ClientId.unique())
```

```
    tbl = RemoveByStartDate(tbl,leftStart,leftEnd,tbl.EntryType == 'Sleep')
    nLeftRemoved = nClientsAll - len(tbl.ClientId.unique())

    tbl = RemoveByStartDate(tbl,rightStart,rightEnd,tbl.EntryType == 'Sleep')
    nRightRemoved = nClientsAll - nLeftRemoved - len(tbl.ClientId.unique())


    # Discard all data before September 1, 2008 due to import errors from␣
↪previous database.
    tbl = tbl.loc[tbl.Date >= pd.to_datetime('2008-09-01')]

    nClients = len(tbl.ClientId.unique())

    print('Total Clients: {:d}/{:d} ({:.1f}%) ({:d} removed left, {:d} removed␣
↪right)'
          .format(nClients,nClientsAll,100.0*nClients/
↪nClientsAll,nLeftRemoved,nRightRemoved))

    return tbl
```

[7]:
```
tbl = PreProcess()
```

```
Total Entries: 5431521
Dates:  2007-07-01 00:00:00  to  2020-01-20 00:00:00
Total Clients: 18398/41935 (43.9%) (19967 removed left, 3570 removed right)
```

[ ]:
```

```

1.0.2 Identify Chronic Shelter Users

Generate a timeline of stays for each client in order to determine who satisfies the Canadian federal chronic shelter use definition.

[8]:
```
def GenerateStayTimelines():
    return tbl.loc[tbl.EntryType=='Sleep'].groupby('ClientId').
↪progress_apply(CalculateStaySequence)
```

[9]:
```
tlSty = GenerateStayTimelines()
```

```
  0%|          | 0/18398 [00:00<?, ?it/s]
```

[10]:
```
# Applies a time windowed threshold test to a count of stays.
def TimeToCdnFedChronic(tbl):

    # First Test: 180 days in past 1 year
    winSz = 365
```

```
    thresh = 180

    win = tbl.rolling('%dd' % winSz,on='Date').count().Ind
    registrationDate = tbl.Date.min()
    idDate1 = tbl[win >= thresh].Date.min()  # Will be equal to NaN if the␣
↪threshold isn't met.

    # Second Test: 546 days in past 3 years
    winSz = 365*3
    thresh = 546

    win = tbl.rolling('%dd' % winSz,on='Date').count().Ind
    registrationDate = tbl.Date.min()
    idDate2 = tbl[win >= thresh].Date.min()  # Will be equal to NaN if the␣
↪threshold isn't met.

    idDate = min([ idDate1, idDate2 ])

    if idDate == idDate:   # Satisfied if idDate is not NaN.
        return pd.Series({
            'Flag': 'chr',  # Flag indicating test was satisfied.
            'Date': idDate,   # Date client was identified.
            'Time': (idDate - registrationDate).days + 1 # Number of days it␣
↪took to identify client.
        })
    else:
        return pd.Series({   # Returned if the test is not satisfied.
            'Flag': 'tmp',
            'Date': tbl.Date.max(),
            'Time': (tbl.Date.max()-tbl.Date.min()).days + 1
        })
```

[11]:
```
def CdnFedChronicTte():
    return tlSty.groupby('ClientId').progress_apply(TimeToCdnFedChronic)
```

[12]:
```
tte = CdnFedChronicTte()
```

```
  0%|          | 0/18398 [00:00<?, ?it/s]
```

[13]:
```
nChron = sum(tte.Flag == 'chr')
nClients = len(tte.Flag)
print('Chronic clients: {}/{} ({:.1f}%)'.format(nChron,nClients,100.0*nChron/
 ↪nClients))
```

```
Chronic clients: 1549/18398 (8.4%)
```

1.0.3 Prediction Points

- The time points after first sleep date where the chronic predictions are made.

[14]:
```
predictionTimes = [ 90 ]
```

1.0.4 Generate Data Features

- These data features are the sum of the individual events in a client's data record (ie. sleeps, bars, record with a violence word, etc.) over a period starting at client's first sleep check-in.

[15]:
```
def SummarizeEntries(tbl,sleepDateDelta):

    endDate = tbl.loc[tbl.EntryType=='Sleep'].Date.min() + sleepDateDelta
    tblValid = tbl.loc[tbl.Date <= endDate]

    entryCountsAndAge = pd.Series({
        'Age': max([ 0, tblValid.Age.max() ]),
        'Bar': sum(tblValid.EntryType == 'Bar'),
        'Counsellor': sum(tblValid.EntryType ==␣
 ↪'CounsellorsNotes')+sum(tblValid.EntryType == 'ProgressDetails'),
        'Log': sum(tblValid.EntryType == 'Log'),
        'Sleep': sum(tblValid.EntryType == 'Sleep'),
        })

    return pd.concat([ entryCountsAndAge,␣
 ↪tblValid[['Police','Ems','Health','Violence','Addiction']].sum() ])
```

[16]:
```
def GenerateFeatureTable(tbl,assmtPnts):

    deltas = { '{}d'.format(pt): pd.Timedelta(pt,unit='d') for pt in assmtPnts }

    colIndNames = [
        deltas.keys(),
        ␣
 ↪['Age','Bar','Counsellor','Log','Sleep','Police','Ems','Health','Violence','Addiction']
    ]

    tbls = {}

    for winStr in deltas.keys():

        startDate = tbl[tbl.EntryType=='Sleep'].Date.min()
        endDate = startDate + deltas[winStr]

        tbls[winStr] = tbl.groupby('ClientId').
 ↪progress_apply(SummarizeEntries,sleepDateDelta=deltas[winStr])
```

```
    return pd.concat(tbls.values(),axis=1,keys=tbls.keys())
```

[17]:
```
datX = GenerateFeatureTable(tbl,assmtPnts = predictionTimes)
```

```
0%|          | 0/18398 [00:00<?, ?it/s]
```

1.0.5 Demographics

[18]:
```
def CalculateClientDemographics():
    return tbl.groupby('ClientId').progress_apply(ShelterGroupDemographics)
```

[19]:
```
demog = CalculateClientDemographics()
```

```
0%|          | 0/18398 [00:00<?, ?it/s]
```

1.0.6 Threshold Test

[20]:
```
# Applies a time windowed threshold test to a count of stays.
def SleepThreshChronicTest(tbl,threshPct,winSizes):

    results = []
    for winSz in winSizes:
        thresh = int(winSz*threshPct)
        win = tbl.rolling('%dd' % winSz,on='Date').count().Ind
        idDate = tbl[win >= thresh].Date.min()  # Will be equal to NaN if the␣
↪threshold isn't met.
        results.append(idDate == idDate)

    return pd.Series({'{}d'.format(winSizes[i]): results[i] for i in␣
↪range(len(winSizes)) })
```

[21]:
```
def SimpleThreshold():
    return tlSty.groupby('ClientId').
↪progress_apply(SleepThreshChronicTest,threshPct=0.
↪75,winSizes=predictionTimes)
```

[22]:
```
thshTest = SimpleThreshold()
```

```
0%|          | 0/18398 [00:00<?, ?it/s]
```

1.0.7 Initialize Machine Learning Algorithms

[23]:
```
mlpCls = { t: MLPClassifier(
                activation='relu',
                alpha=0.05,
                hidden_layer_sizes=(100,),
```

```
                learning_rate='adaptive',
                solver='sgd')
            for t in predictionTimes }
```

[24]:
```
lrCls = { t: LogisticRegression() for t in predictionTimes }
```

[25]:
```
datXNrm = copy.deepcopy(datX)
for colInd in datX.columns:
    datXNrm[colInd] = (datX[colInd]-datX[colInd].mean())/np.sqrt(datX[colInd].
 ↪var())
```

[26]:
```
datY = tte.Flag == 'chr'
```

1.0.8 K-Fold Evaluation

NOTE: Since the stratified K-fold routine randomly selects training and testing sets each time it's called, the results shown below may differ slightly from the results we show in our publication.

[27]:
```
nSplit = 10
nRepeat = 1
skf = RepeatedStratifiedKFold(n_splits=nSplit, n_repeats=nRepeat)
```

[28]:
```
nnPrf = { t: { 'TP': 0, 'FP': 0, 'TN': 0, 'CndP': 0, 'CndN': 0 } for t in␣
 ↪predictionTimes }
lrPrf = { t: { 'TP': 0, 'FP': 0, 'TN': 0, 'CndP': 0, 'CndN': 0 } for t in␣
 ↪predictionTimes }
thshPrf = { t: { 'TP': 0, 'FP': 0, 'TN': 0, 'CndP': 0, 'CndN': 0 } for t in␣
 ↪predictionTimes }
```

[29]:
```
nnCohort = { t: [] for t in predictionTimes }
lrCohort = { t: [] for t in predictionTimes }
thshCohort = { t: [] for t in predictionTimes }
```

[30]:
```
def ThreshEval(res,threshResults,dataY,testInd,trainInd):
    hat = threshResults.iloc[testInd]

    res['TP'] += sum((hat == True) & (dataY.iloc[testInd] == True))
    res['FP'] += sum((hat == True) & (dataY.iloc[testInd] == False))
    res['TN'] += sum((hat == False) & (dataY.iloc[testInd] == False))
    res['CndP'] += sum(dataY.iloc[testInd] == True)
    res['CndN'] += sum(dataY.iloc[testInd] == False)

    return threshResults.iloc[testInd].loc[hat].index
```

[31]:
```
def MLTrainAndEval(est,res,dataX,dataY,testInd,trainInd):
    est.fit(dataX.iloc[trainInd], dataY.iloc[trainInd])
    hat = est.predict(dataX.iloc[testInd])
```

```
        res['TP'] += sum((hat == True) & (dataY.iloc[testInd] == True).to_numpy())
        res['FP'] += sum((hat == True) & (dataY.iloc[testInd] == False).to_numpy())
        res['TN'] += sum((hat == False) & (dataY.iloc[testInd] == False).to_numpy())
        res['CndP'] += sum(dataY.iloc[testInd] == True)
        res['CndN'] += sum(dataY.iloc[testInd] == False)

        return dataX.iloc[testInd].loc[hat].index
```

[32]:

```
for tPred in predictionTimes:
    print('Prediction Time: {} days'.format(tPred))
    tStr = '{}d'.format(tPred)

    for trainInd, testInd in tqdm(skf.split(datXNrm,␣
↪datY),total=nSplit*nRepeat):

        nnCohort[tPred].extend(
            ␣
↪MLTrainAndEval(mlpCls[tPred],nnPrf[tPred],datXNrm[tStr],datY,testInd,trainInd))

        lrCohort[tPred].extend(
            ␣
↪MLTrainAndEval(lrCls[tPred],lrPrf[tPred],datXNrm[tStr][['Age','Sleep']],datY,testInd,trainI

        thshCohort[tPred].extend(
            ThreshEval(thshPrf[tPred],thshTest[tStr],datY,testInd,trainInd))
```

```
Prediction Time: 90 days

  0%|          | 0/10 [00:00<?, ?it/s]
```

[40]:

```
def PrintDemographyStats(demog,cohortInd,nRepeat):
    cohort = demog.loc[cohortInd]

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

    nPop = len(demog.index)
    nCohort = int(len(cohort.index)/nRepeat)
    print( 'Avg clients in cohort: %d/%d (%.1f%%)' % (nCohort,nPop,100*nCohort/
↪nPop))

    fields = [ 'TotalStays', 'TotalEpisodes', 'Tenure', 'UsagePct', 'AvgGapLen'␣
↪]
```

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

    print()
```

[41]:
```
def PrintEstimatorPerformance(r):
    print('TPR/Sens: %g (%d), FPR/FlsAlrm: %g (%d), Confidence: %g, Accuracy:␣
 ↪%g'
          % (r['TP']/r['CndP'], r['TP'], r['FP']/r['CndN'], r['FP'], r['TP']/
 ↪(r['TP']+r['FP']),
            (r['TP']+r['TN'])/(r['CndN']+r['CndP'])    ))
    print()
```

Performance: Logistic Regression

[42]:
```
for tPred in predictionTimes:
    print('---- Prediction at {} Days ----'.format(tPred))
    PrintEstimatorPerformance(lrPrf[tPred])
    PrintDemographyStats(demog,lrCohort[tPred],nRepeat)
    print()
```

```
---- Prediction at 90 Days ----
TPR/Sens: 0.316333 (490), FPR/FlsAlrm: 0.0162621 (274), Confidence: 0.641361,
Accuracy: 0.927546

Avg clients in cohort: 764/18398 (4.2%)
TotalStays:
 Avg: 671.8, Med: 409.5, 10thPct: 113.0, 90thPct: 1687.0
TotalEpisodes:
 Avg: 3.8, Med: 3.0, 10thPct: 1.0, 90thPct: 8.0
Tenure:
 Avg: 1273.6, Med: 1055.0, 10thPct: 201.0, 90thPct: 2662.0
UsagePct:
 Avg: 60.6, Med: 60.4, 10thPct: 13.7, 90thPct: 100.8
AvgGapLen:
 Avg: 3.1, Med: 1.5, 10thPct: 0.9, 90thPct: 7.3
```

Performance: Neural Network

[43]:
```
for tPred in predictionTimes:
    print('---- Prediction at {} Days ----'.format(tPred))
```

```
    PrintEstimatorPerformance(nnPrf[tPred])
    PrintDemographyStats(demog,nnCohort[tPred],nRepeat)
    print()
```

```
---- Prediction at 90 Days ----
TPR/Sens: 0.355713 (551), FPR/FlsAlrm: 0.0179833 (303), Confidence: 0.645199,
Accuracy: 0.929286

Avg clients in cohort: 854/18398 (4.6%)
TotalStays:
 Avg: 660.7, Med: 394.5, 10thPct: 106.0, 90thPct: 1681.0
TotalEpisodes:
 Avg: 3.7, Med: 3.0, 10thPct: 1.0, 90thPct: 8.0
Tenure:
 Avg: 1295.4, Med: 1091.5, 10thPct: 174.0, 90thPct: 2702.0
UsagePct:
 Avg: 59.1, Med: 59.9, 10thPct: 13.0, 90thPct: 99.9
AvgGapLen:
 Avg: 3.3, Med: 1.6, 10thPct: 1.0, 90thPct: 7.7
```

Performance: Threshold Test

```
[44]: for tPred in predictionTimes:
    print('---- Prediction at {} Days ----'.format(tPred))
    PrintEstimatorPerformance(thshPrf[tPred])
    PrintDemographyStats(demog,thshCohort[tPred],nRepeat)
    print()
```

```
---- Prediction at 90 Days ----
TPR/Sens: 0.985152 (1526), FPR/FlsAlrm: 0.0582824 (982), Confidence: 0.608453,
Accuracy: 0.945374

Avg clients in cohort: 2508/18398 (13.6%)
TotalStays:
 Avg: 563.3, Med: 362.0, 10thPct: 120.0, 90thPct: 1271.0
TotalEpisodes:
 Avg: 5.4, Med: 4.0, 10thPct: 1.0, 90thPct: 11.0
Tenure:
 Avg: 1526.9, Med: 1397.0, 10thPct: 288.0, 90thPct: 2961.0
UsagePct:
 Avg: 44.6, Med: 36.7, 10thPct: 11.1, 90thPct: 93.8
AvgGapLen:
 Avg: 4.2, Med: 2.7, 10thPct: 1.0, 90thPct: 9.0
```

[ ]:

```

```

[ ]:

```

```