import torch

import torchtuples as tt

import numpy as np

from sklearn.preprocessing import StandardScaler, LabelEncoder

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import roc\_auc\_score

import pandas as pd

from torchtuples.practical import MLPVanilla, accuracy\_binary

from pycox.models import CoxPH

from pycox.evaluation import EvalSurv

import matplotlib.pyplot as plt

raw = pd.read\_csv("num.csv",na\_values="Unknown")

raw.head()

raw.columns

train = raw[raw.Group == "Train"]

train.head()

test = raw[raw.Group == "Test"]

test.head()

train\_x = train.loc[:,'Age':'PSA']

test\_x = test.loc[:,'Age':'PSA']

num\_var = ["Age",'Size','Regional.nodes.positive', 'Regional.nodes.examined','PSA','Gleason.C.', 'Gleason.P.']

train\_x[num\_var].describe()

x\_train = train\_x.copy()

x\_train["Age"] =( x\_train["Age"] - 63.279221 ) / 6.836121

x\_train["Size"] =( x\_train["Size"] - 26.955889 ) / 31.334743

x\_train["Regional.nodes.positive"] =( x\_train["Regional.nodes.positive"] - 0.316614 ) / 1.054857

x\_train["Regional.nodes.examined"] =( x\_train["Regional.nodes.examined"] - 8.960591 ) / 7.596325

x\_train["PSA"] =( x\_train["PSA"] - 12.231348 ) / 12.782991

x\_train["Gleason.C."] =( x\_train["Gleason.C."] - 7.436632 ) / 0.889980

x\_train["Gleason.P."] =( x\_train["Gleason.P."] - 7.505374 ) / 0.856615

x\_test = test\_x.copy()

x\_test["Age"] =( x\_test["Age"] - 63.279221 ) / 6.836121

x\_test["Size"] =( x\_test["Size"] - 26.955889 ) / 31.334743

x\_test["Regional.nodes.positive"] =( x\_test["Regional.nodes.positive"] - 0.316614 ) / 1.054857

x\_test["Regional.nodes.examined"] =( x\_test["Regional.nodes.examined"] - 8.960591 ) / 7.596325

x\_test["PSA"] =( x\_test["PSA"] - 12.231348 ) / 12.782991

x\_test["Gleason.C."] =( x\_test["Gleason.C."] - 7.436632 ) / 0.889980

x\_test["Gleason.P."] =( x\_test["Gleason.P."] - 7.505374 ) / 0.856615

x\_train = x\_train.values.astype(np.float32)

x\_test = x\_test.values.astype(np.float32)

x\_train.shape[1]

Y = lambda df\_new: (df\_new['duration'].values, df\_new['event'].values)

y\_train = Y(train)

y\_test = Y(test)

in\_features = x\_train.shape[1]

num\_nodes = [16,16]

out\_features = 1

batch\_norm = True

dropout = 0.1

output\_bias = False

epochs = 100

callbacks = [tt.callbacks.EarlyStopping(patience=30)]

#callbacks = None

verbose = True

batch\_size = 512

net = tt.practical.MLPVanilla(in\_features, num\_nodes, out\_features, batch\_norm,

dropout, output\_bias=output\_bias)

model\_cox = CoxPH(net, tt.optim.Adam(lr=0.05)) #cox model on the neural network with Adam optimizer

log\_cox = model\_cox.fit(x\_train, y\_train, batch\_size, epochs, callbacks, verbose,

val\_data =(x\_test,y\_test), val\_batch\_size=batch\_size)

#Train

time\_test0, status\_test0 = y\_train[0], y\_train[1]

surv\_cox0 = model\_cox.predict\_surv\_df(x\_train) # survival of the train data

eval\_cox0 = EvalSurv(surv\_cox0, time\_test0, status\_test0, censor\_surv= 'km')

cox\_index0 = eval\_cox0.concordance\_td()

cox\_index0

# test

model\_cox.compute\_baseline\_hazards() #baseline hazard

surv\_cox = model\_cox.predict\_surv\_df(x\_test) # survival of the test data

# evaluate the c-index of the cox model

eval\_cox = EvalSurv(surv\_cox, y\_test[0], y\_test[1], censor\_surv= 'km')

cox\_index = eval\_cox.concordance\_td()

cox\_index # cox\_index at all

%matplotlib inline

%config InlineBackend.figure\_format = 'svg'

log\_cox.to\_pandas()[['train\_loss', 'val\_loss']].plot()

plt.xlabel('epoch')

plt.ylabel('loss')

plt.savefig('os.pdf', bbox\_inches='tight')

plt.show()

model\_cox.save\_model\_weights("os\_param.pt")

model\_cox.save\_net("os\_net.pt")

# train

# ATTENTION :x\_train, y\_train is numpy.ndarray

in\_tem = []

model\_cox.compute\_baseline\_hazards()

for i in range(1000):

idx = np.random.choice(len(x\_train),size = 500,replace = True)

x\_tem = x\_train[idx,]

surv\_cox\_tem = model\_cox.predict\_surv\_df(x\_tem) # survival of the test data

eval\_cox\_tem = EvalSurv(surv\_cox\_tem, y\_train[0][idx], y\_train[1][idx], censor\_surv= 'km')

c\_tem = eval\_cox\_tem.concordance\_td()

in\_tem.append(c\_tem)

in\_mean = np.mean(in\_tem)

in\_se = np.std(in\_tem) / np.power(1000,0.5)

print("Bootstrap Internal validation:")

print("The C-index:{}".format(in\_mean))

print("The C-index High 95% CI:{}".format(in\_mean + 1.96 \* in\_se))

print("The C-index Low 95% CI:{}".format(in\_mean - 1.96 \* in\_se))

# Test

ex\_tem = []

model\_cox.compute\_baseline\_hazards()

for i in range(1000):

idx = np.random.choice(len(x\_test),size = 500,replace = True)

x\_tem = x\_test[idx,]

surv\_cox\_tem = model\_cox.predict\_surv\_df(x\_tem) # survival of the test data

eval\_cox\_tem = EvalSurv(surv\_cox\_tem, y\_test[0][idx], y\_test[1][idx], censor\_surv= 'km')

c\_tem = eval\_cox\_tem.concordance\_td()

ex\_tem.append(c\_tem)

ex\_mean = np.mean(ex\_tem)

ex\_se = np.std(ex\_tem) / np.power(1000,0.5)

print("Bootstrap External validation:")

print("The C-index:{}".format(ex\_mean))

print("The C-index High 95% CI:{}".format(ex\_mean + 1.96 \* ex\_se))

print("The C-index Low 95% CI:{}".format(ex\_mean - 1.96 \* ex\_se))