Analysis code for the Deep Learning models: DeepSurv, DeepHit and Dynamic DeepHit

1 DeepSurv code

```
# -*- coding: utf-8 -*-
"""
Created on Mon Oct 28 2024

@author: Sarah Ogutu
"""
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn_pandas import DataFrameMapper

import torch
import torchtuples as tt

from pycox.models import CoxPH
#from pycox.evaluation.concordance import concordance_td
#from pycox.evaluation import ipcw, admin


from Util import EvalSurv #updated EvalSurv for pd error
from Model import linear_model
from IPython import get_ipython

np.random.seed(12346)
_ = torch.manual_seed(1235)


df_diff = pd.read_excel('D:/Downloads/Fred Hutch/diff_data.xlsx') # imputed
df_diff = df_diff.drop(df_diff.columns[0], axis=1)
#df_diff = pd.read_excel('D:/Downloads/Fred Hutch/dl_diff_cc.xlsx')#complete case
#df_diff = df_diff.drop(df_diff.columns[0], axis=1)

#Split data into Train, Test and Valid
df_train2 = df_diff
df_test2 = df_train2.sample(frac=0.2)
df_train2 = df_train2.drop(df_test2.index)
df_val2 = df_train2.sample(frac=0.2)
df_train2 = df_train2.drop(df_val2.index)

#Feature transforms
cols_standardize1 = ['age1', 'agedebu', 'p17v26_AGE_OLDEST_SEX_PART',
'BASIC_FGF', 'EOTAXIN', 'G_CSF', 'GM_CSF', 'IFN_G',
```

```
'IL_10', 'IL_12P70', 'IL_13', 'IL_15', 'IL_17A', 'IL_1B',
'IL_1RA', 'IL_2', 'IL_4', 'IL_5', 'IL_6', 'IL_7', 'IL_8',
'IL_9', 'IP_10', 'MCP_1', 'MIP_1A', 'MIP_1B', 'PDGF_BB',
'RANTES', 'TNF_A', 'VEGF', 'CTACK', 'GRO_A', 'HGF',
'IFN_A2', 'IL_12P40', 'IL_16', 'IL_18', 'IL_1A', 'IL_2RA',
'IL_3', 'LIF', 'M_CSF', 'MCP_3', 'MIF', 'MIG', 'SCF',
'SCGF_B', 'SDF_1A', 'TNF_B', 'TRAIL', 'B_NGF']
cols_leave1 = ['p5v9_LENGTH_IN_DBNVUL', 'p17v10_PARTNERS', 'p17v11_YEAR_STABLE',
'p17v12_YEAR_CASUAL', 'p17v13_30DAYS_STABLE',
'p17v14_30DAYS_CASUAL', 'p17v15_SEX_30DAYS', 'Treat_Placebo',
'Treat_Tenofovir', 'Site_eThekwini', 'Site_Vulindlela',
'p2v18_REG_PARTNER_LIVE_TOGETHER_No',
'p2v18_REG_PARTNER_LIVE_TOGETHER_Yes',
'p2v22_HIGHEST_EDUCATION_HSchool',
'p2v22_HIGHEST_EDUCATION_Primary',
'p2v22_HIGHEST_EDUCATION_Tertiary', 'p3v8_SELF_GEN_INCOME_No',
'p3v8_SELF_GEN_INCOME_Yes', 'p3v9_SALARY_No', 'p3v9_SALARY_Yes',
'p3v10_HUSBAND_No', 'p3v10_HUSBAND_Yes', 'p3v11_SOCIAL_GRANTS_No',
'p3v11_SOCIAL_GRANTS_Yes', 'p3v13_OTHER_INCOME_SOURCE_No',
'p3v13_OTHER_INCOME_SOURCE_Yes', 'p3v16_AMOUNT_INCOME_G1Kless5K',
'p3v16_AMOUNT_INCOME_less1K', 'marital_Casual', 'marital_Married',
'marital_StabCas', 'marital_Stable',
'p17v28_SEX_PART_HAVE_OTHER_Dontknow',
'p17v28_SEX_PART_HAVE_OTHER_No',
'p17v28_SEX_PART_HAVE_OTHER_Yes', 'p18v15_FREQ_CONDOM_USE_Always',
'p18v15_FREQ_CONDOM_USE_Occasionally',
'p19v14_ABNORMAL_DISCHARGE_No', 'p19v14_ABNORMAL_DISCHARGE_Yes']
standardize1 = [([col], StandardScaler()) for col in cols_standardize1]
leave1 = [(col, None) for col in cols_leave1]
x_mapper1 = DataFrameMapper(standardize1 + leave1)

x_train2 = x_mapper1.fit_transform(df_train2).astype('float32')
x_val2 = x_mapper1.transform(df_val2).astype('float32')
x_test2 = x_mapper1.transform(df_test2).astype('float32')


#Get Surv labels
get_target2 = lambda df: (df['months'].values, df['HIV'].values)
y_train2 = get_target2(df_train2)
y_val2 = get_target2(df_val2)
y_test2 = get_target2(df_test2)


################################################################################
# Pre-process dataset
################################################################################
in_features = x_train2.shape[1]
out_features  = 1
node_dims = [in_features,32,32,out_features]
```

```
layer_norm = True
batch_norm = True
dropout = 0.1
output_bias = False
activation_type = 'relu'
num_nodes = [32, 32]
batch_size = 128
epochs = 512

#Construct model
net2 = linear_model(node_dims, dropout, activation_type, layer_norm, batch_norm)
#Define optimizer
optimizer = tt.optim.Adam

#Define CoxPH model
model2 = CoxPH(net2, optimizer)

#Find best lr
lrfinder2 = model2.lr_finder(x_train2, y_train2, batch_size, tolerance=10)
#_ = lrfinder2.plot()
best_lr2 = lrfinder2.get_best_lr()

#Set best lr
model2.optimizer.set_lr(0.01)

#Set early stopping
callbacks = [tt.callbacks.EarlyStopping()]


##############################################################################
# Train Model
##############################################################################
get_ipython().run_line_magic('time', '')
log2 = model2.fit(x_train2, y_train2, batch_size, epochs, callbacks,
verbose = True,
val_data = (x_val2, y_val2), val_batch_size=batch_size)

#Plot loss
_ = log2.plot()



##############################################################################
# Test (Prediction)
##############################################################################
#compute baseline hazards at each time point
baseline_hazard2 = model2.compute_baseline_hazards()

#Predict
```

```
surv_pred2 = model2.predict_surv_df(x_test2) #rows-timepoints,cols-samples

#Plot survival for patients
num_pts = 10
surv_pred2.iloc[:, :num_pts].plot()
plt.ylabel('S(t | x)')
plt.title("Predicted Survival: Diff Model")
_ = plt.xlabel('Time')
plt.figure()
plt.close()


#############################################################################
#Evaluation
#############################################################################
ev2 = EvalSurv(surv_pred2, y_test2[0], y_test2[1], censor_surv='km')

#The time-dependent concordance index evaluated at the event times
cindex_td2 = ev2.concordance_td()
print("\nThe time-dep c-index is: ",round(cindex_td2,4))


#Get Time points in test sets
time_grid2 = np.linspace(y_test2[0].min(), y_test2[0].max(), 100)


#The integrated IPCW Brier score.
inte_ipcw_bs2 = ev2.integrated_brier_score(time_grid2)
print("The integrated IPCW Brier Score is:", round(inte_ipcw_bs2,4))

#The integrated IPCW (negative) binomial log-likelihood.
inte_ipcw_nbl2 = ev2.integrated_nbll(time_grid2)
print("The integrated IPCW (negative) binomial log-likelihood is: ",
round(inte_ipcw_nbl2,4))

#The IPCW Brier score (inverse probability of censoring weighted Brier score)
ipcw_bs2 = ev2.brier_score(time_grid2)
print("The  IPCW Brier score is: \n",ipcw_bs2)
plt.figure()
plt.title("IPCW Brier score")
_ = ev2.brier_score(time_grid2).plot()

#############################################################################
#combined plot
import matplotlib.pyplot as plt# side-by-side plots
fig, axes = plt.subplots(1, 2, figsize=(24, 9))

# Plot the survival predictions for the first model on the first axis
num_pts = 10
```

```
surv_pred.iloc[:, :num_pts].plot(ax=axes[0])
axes[0].set_ylabel('S(t | x)', fontsize = 30)
axes[0].set_title('(a)', fontsize = 30)
axes[0].set_xlabel('Time (Months)', fontsize = 30)
axes[0].set_ylim([0.75, 1])  # Set y-axis limit

# Adjust tick parameters for the first plot
axes[0].tick_params(axis='both', labelsize = 30)

# Increase the font size of the legend for the first plot
axes[0].legend(fontsize = 23)

# Plot the survival predictions for the second model on the second axis
surv_pred2.iloc[:, :num_pts].plot(ax=axes[1])
axes[1].set_ylabel('S(t | x)', fontsize = 30)
axes[1].set_title('(b)', fontsize = 30)
axes[1].set_xlabel('Time (Months)', fontsize = 30)
axes[1].set_ylim([0.75, 1])  # Set y-axis limit

# Adjust tick parameters for the second plot
axes[1].tick_params(axis='both', labelsize = 30 )

# Increase the font size of the legend for the first plot
axes[1].legend(fontsize = 23)

# Adjust the layout to prevent overlap and display the plot
plt.tight_layout()

# Save the combined plots as a high-resolution PDF
plt.savefig('D:/Downloads/Fred Hutch/DeepSurv.pdf', format='pdf', dpi=600)

plt.show()
```

2 DeepHit code

```
# -*- coding: utf-8 -*-
"""
Created on Mon Oct 28 2024

@author: Sarah Ogutu
"""

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

```
# For preprocessing
from sklearn.preprocessing import StandardScaler
from sklearn_pandas import DataFrameMapper

import torch # For building the networks
import torchtuples as tt # Some useful functions

from pycox.models import DeepHitSingle
from pycox.evaluation import EvalSurv

####### Data Set ######
df_diff = pd.read_excel('D:/Downloads/Fred Hutch/diff_data.xlsx') # Diff data
df_diff = df_diff.drop(df_diff.columns[0], axis=1)
#df_diff = pd.read_excel('D:/Downloads/Fred Hutch/dl_diff_cc.xlsx') # Complete case data
#df_diff = df_diff.drop(df_diff.columns[0], axis=1)

# Set seed
np.random.seed(811)
_ = torch.manual_seed(8111)

df_train4 = df_diff
df_test4 = df_train4.sample(frac=0.2)
df_train4 = df_train4.drop(df_test4.index)
df_val4 = df_train4.sample(frac=0.2)
df_train4 = df_train4.drop(df_val4.index)

######## Fearture transformation #############
cols_standardize1 = ['age1', 'agedebu', 'p17v26_AGE_OLDEST_SEX_PART', 'BASIC_FGF',
'EOTAXIN', 'G_CSF', 'GM_CSF', 'IFN_G', 'IL_10', 'IL_12P70',
'IL_13', 'IL_15', 'IL_17A', 'IL_1B', 'IL_1RA', 'IL_2', 'IL_4',
'IL_5', 'IL_6', 'IL_7', 'IL_8', 'IL_9', 'IP_10', 'MCP_1',
'MIP_1A', 'MIP_1B', 'PDGF_BB', 'RANTES', 'TNF_A', 'VEGF', 'CTACK',
'GRO_A', 'HGF', 'IFN_A2', 'IL_12P40', 'IL_16', 'IL_18',
'IL_1A', 'IL_2RA', 'IL_3', 'LIF', 'M_CSF', 'MCP_3', 'MIF',
'MIG', 'SCF', 'SCGF_B', 'SDF_1A', 'TNF_B', 'TRAIL', 'B_NGF']
cols_leave1 = ['p5v9_LENGTH_IN_DBNVUL', 'p17v10_PARTNERS', 'p17v11_YEAR_STABLE',
'p17v12_YEAR_CASUAL', 'p17v13_30DAYS_STABLE', 'p17v14_30DAYS_CASUAL',
'p17v15_SEX_30DAYS', 'Treat_Placebo', 'Treat_Tenofovir',
'Site_eThekwini', 'Site_Vulindlela', 'p2v18_REG_PARTNER_LIVE_TOGETHER_No',
'p2v18_REG_PARTNER_LIVE_TOGETHER_Yes', 'p2v22_HIGHEST_EDUCATION_HSchool',
'p2v22_HIGHEST_EDUCATION_Primary', 'p2v22_HIGHEST_EDUCATION_Tertiary',
'p3v8_SELF_GEN_INCOME_No', 'p3v8_SELF_GEN_INCOME_Yes', 'p3v9_SALARY_No',
'p3v9_SALARY_Yes', 'p3v10_HUSBAND_No', 'p3v10_HUSBAND_Yes',
'p3v11_SOCIAL_GRANTS_No', 'p3v11_SOCIAL_GRANTS_Yes',
'p3v13_OTHER_INCOME_SOURCE_No', 'p3v13_OTHER_INCOME_SOURCE_Yes',
'p3v16_AMOUNT_INCOME_G1Kless5K', 'p3v16_AMOUNT_INCOME_less1K',
'marital_Casual', 'marital_Married', 'marital_StabCas', 'marital_Stable',
'p17v28_SEX_PART_HAVE_OTHER_Dontknow', 'p17v28_SEX_PART_HAVE_OTHER_No',
'p17v28_SEX_PART_HAVE_OTHER_Yes', 'p18v15_FREQ_CONDOM_USE_Always',
```

```
'p18v15_FREQ_CONDOM_USE_Occasionally', 'p19v14_ABNORMAL_DISCHARGE_No',
'p19v14_ABNORMAL_DISCHARGE_Yes']
standardize1 = [([col], StandardScaler()) for col in cols_standardize1]
leave1 = [(col, None) for col in cols_leave1]
x_mapper = DataFrameMapper(standardize1 + leave1)

x_train4 = x_mapper.fit_transform(df_train4).astype('float32')
x_val4 = x_mapper.transform(df_val4).astype('float32')
x_test4 = x_mapper.transform(df_test4).astype('float32')

######## Label Transform ##################
num_durations = 10
labtrans = DeepHitSingle.label_transform(num_durations)
get_target4 = lambda df: (df['months'].values, df['HIV'].values)
y_train4 = labtrans.fit_transform(*get_target4(df_train4))
y_val4 = labtrans.transform(*get_target4(df_val4))

train = (x_train4, y_train4)
val = (x_val4, y_val4)

# We don't need to transform the test labels
durations_test, events_test = get_target4(df_test4)
y_test4 = get_target4(df_test4)

type(labtrans)

###### Neural Net ######################
in_features = x_train4.shape[1]
num_nodes = [32, 32]
out_features = labtrans.out_features
batch_norm = True
dropout = 0.1

net4 = tt.practical.MLPVanilla(in_features, num_nodes, out_features, batch_norm,
dropout)
############ Training the model #############
model4 = DeepHitSingle(net4, tt.optim.Adam, alpha=0.2, sigma=0.1,
duration_index=labtrans.cuts)
batch_size = 256
lr_finder4 = model4.lr_finder(x_train4, y_train4, batch_size, tolerance=3)
_ = lr_finder4.plot()

lr_finder4.get_best_lr()
model4.optimizer.set_lr(0.01)

epochs = 512
callbacks = [tt.callbacks.EarlyStopping()]
log4 = model4.fit(x_train4, y_train4, batch_size, epochs, callbacks,
val_data=val)
```

```
_ = log4.plot()

############## Prediction #############
surv4 = model4.predict_surv_df(x_test4)


surv4.iloc[:, :10].plot(drawstyle='steps-post')
plt.ylabel('S(t | x)')
_ = plt.xlabel('Time')

surv44 = model4.interpolate(10).predict_surv_df(x_test4)

surv44.iloc[:, :10].plot(drawstyle='steps-post')
plt.ylabel('S(t | x)')
_ = plt.xlabel('Time')

############ Evaluation ##########
# C-index
ev4 = EvalSurv(surv4, durations_test, events_test, censor_surv='km')
ev4.concordance_td()

# IPCW Brier score
time_grid4 = np.linspace(y_test4[0].min(), y_test4[0].max(), 100)
ev4.brier_score(time_grid4).plot()
plt.ylabel('Brier score')
_ = plt.xlabel('Time')

# Negative binomial likelihood
ev4.nbll(time_grid4).plot()
plt.ylabel('NBLL')
_ = plt.xlabel('Time')

# itegrated Brier Score
ev4.integrated_brier_score(time_grid4)
ev4.integrated_nbll(time_grid4)

#############################################################################
hazard4 = -np.diff(surv4.values, axis=0)  # Calculate the hazard function

# Convert the hazard function to a DataFrame and use the time index
hazard_df2 = pd.DataFrame(hazard4, index=surv4.index[1:], columns=surv4.columns)

# Plotting the hazard graph
plot2 = plt.figure(figsize=(12, 8))
for i in range(hazard_df2.shape[1]):
plt.plot(hazard_df2.index, hazard_df2.iloc[:, i], alpha=0.6)

plt.title("Hazard Graph Single Risk: Difference model")
plt.xlabel("Evaluation time (months)")
```

```
plt.ylabel("Probability of risk")
#plt.show()

##############################################################################

# combined plot
import matplotlib.pyplot as plt

# Create a figure with two subplots (side by side)
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(24, 9))  # Adjust figsize as needed

# Plotting the first hazard graph in ax1
for i in range(hazard_df.shape[1]):
ax1.plot(hazard_df.index, hazard_df.iloc[:, i], alpha=0.6)

ax1.set_title("(a)", fontsize = 24)
ax1.set_xlabel("Evaluation time (months)", fontsize = 30)
ax1.set_ylabel("Probability of risk", fontsize = 30)

# Plotting the second hazard graph in ax2 (replace with your own data for plot2)
for i in range(hazard_df2.shape[1]):  # Replace plot2_df with your actual DataFrame for plot2
ax2.plot(hazard_df2.index, hazard_df2.iloc[:, i], alpha=0.6)  # Adjust as needed

ax2.set_title("(b)", fontsize = 24)  # Set your title for plot2
ax2.set_xlabel("Evaluation time (months)", fontsize = 30)
ax2.set_ylabel("Probability of risk", fontsize = 30)

# Increase font size for axis tick labels
ax1.tick_params(axis='both', labelsize=30)
ax2.tick_params(axis='both', labelsize=30)

# Set the same y-axis limits for both subplots
y_min, y_max = 0, 1  # Set appropriate limits based on your data
ax1.set_ylim(y_min, y_max)
ax2.set_ylim(y_min, y_max)

# Adjust layout
plt.tight_layout()

# Save the combined plots as a high-resolution PDF
plt.savefig('D:/Downloads/Fred Hutch/DeepHit.pdf', format='pdf', dpi=600)

# Show the combined plots
plt.show()
```

3 Dynamic DeepHit code

3.1 Data preparation code

```
# -*- coding: utf-8 -*-
"""
Created on Mon NOV 4 2024

@author: Sarah Ogutu
"""

import pandas as pd
import numpy as np



##### USER-DEFINED FUNCTIONS
def f_get_Normalization(X, norm_mode):
num_Patient, num_Feature = np.shape(X)

if norm_mode == 'standard': #zero mean unit variance
for j in range(num_Feature):
if np.nanstd(X[:,j]) != 0:
X[:,j] = (X[:,j] - np.nanmean(X[:, j]))/np.nanstd(X[:,j])
else:
X[:,j] = (X[:,j] - np.nanmean(X[:, j]))
elif norm_mode == 'normal': #min-max normalization
for j in range(num_Feature):
X[:,j] = (X[:,j] - np.nanmin(X[:,j]))/(np.nanmax(X[:,j]) - np.nanmin(X[:,j]))
else:
print("INPUT MODE ERROR!")

return X


def f_get_fc_mask1(meas_time, num_Event, num_Category):
'''
mask3 is required to get the contional probability (to calculate the denominator part)
mask3 size is [N, num_Event, num_Category]. 1's until the last measurement time
'''
mask = np.zeros([np.shape(meas_time)[0], num_Event, num_Category]) # for denominator
for i in range(np.shape(meas_time)[0]):
mask[i, :, :int(meas_time[i, 0]+1)] = 1 # last measurement time

return mask


def f_get_fc_mask2(time, label, num_Event, num_Category):
'''
```

```
mask4 is required to get the log-likelihood loss
mask4 size is [N, num_Event, num_Category]
if not censored : one element = 1 (0 elsewhere)
if censored     : fill elements with 1 after the censoring time (for all events)
'''
mask = np.zeros([np.shape(time)[0], num_Event, num_Category]) # for the first loss function
for i in range(np.shape(time)[0]):
if label[i,0] != 0:  #not censored
mask[i,int(label[i,0]-1),int(time[i,0])] = 1
else: #label[i,2]==0: censored
mask[i,:,int(time[i,0]+1):] =  1 #fill 1 until from the censoring time (to get 1 - \sum F)
return mask


def f_get_fc_mask3(time, meas_time, num_Category):
'''
mask5 is required calculate the ranking loss (for pair-wise comparision)
mask5 size is [N, num_Category].
- For longitudinal measurements:
1's from the last measurement to the event time (exclusive and inclusive, respectively)
denom is not needed since comparing is done over the same denom
- For single measurement:
1's from start to the event time(inclusive)
'''
mask = np.zeros([np.shape(time)[0], num_Category]) # for the first loss function
if np.shape(meas_time):  #lonogitudinal measurements
for i in range(np.shape(time)[0]):
t1 = int(meas_time[i, 0]) # last measurement time
t2 = int(time[i, 0]) # censoring/event time
mask[i,(t1+1):(t2+1)] = 1  #this excludes the last measurement time and includes the event time
else:                     #single measurement
for i in range(np.shape(time)[0]):
t = int(time[i, 0]) # censoring/event time
mask[i,:(t+1)] = 1  #this excludes the last measurement time and includes the event time
return mask

import pdb

##### TRANSFORMING DATA
def f_construct_dataset(df, feat_list):
grouped  = df.groupby(['PID'])
id_list  = pd.unique(df['PID'])
max_meas = np.max(grouped.count())[0]

# Make sure len(feat_list) matches the expected number of features
data     = np.zeros([len(id_list), max_meas, len(feat_list)+1])  # Adjusted initialization
pat_info = np.zeros([len(id_list), 5])

for i, tmp_id in enumerate(id_list):
```

```
tmp = grouped.get_group(tmp_id).reset_index(drop=True)

pat_info[i,4] = tmp.shape[0]                                    # Number of measurements
pat_info[i,3] = np.max(tmp['Follow_up_Month'])                  # Last measurement time
pat_info[i,2] = tmp['HIV'][0]                                   # Cause
pat_info[i,1] = tmp['months'][0]                                # Time to event
pat_info[i,0] = tmp['PID'][0]                                   # Patient ID

data[i, :int(pat_info[i, 4]), 1:]  = tmp[feat_list]
data[i, :int(pat_info[i, 4]-1), 0] = np.diff(tmp['Follow_up_Month'])

return pat_info, data


def import_dataset(norm_mode = 'standard'):

df_                = pd.read_csv('./DynDeepHit_data.csv')
bin_list           = ['Treat', 'Site', 'p2v18_REG_PARTNER_LIVE_TOGETHER',
'p3v8_SELF_GEN_INCOME', 'p3v9_SALARY', 'p3v10_HUSBAND',
'p3v11_SOCIAL_GRANTS', 'p3v13_OTHER_INCOME_SOURCE',
'p3v16_AMOUNT_INCOME', 'p18v15_FREQ_CONDOM_USE',
'p19v14_ABNORMAL_DISCHARGE']
cont_list          = ['p5v9_LENGTH_IN_DBNVUL', 'age1', 'agedebu', 'p17v10_PARTNERS',
'p17v11_YEAR_STABLE', 'p17v12_YEAR_CASUAL', 'p17v13_30DAYS_STABLE',
'p17v14_30DAYS_CASUAL', 'p17v15_SEX_30DAYS',
'p17v26_AGE_OLDEST_SEX_PART','BASIC_FGF', 'EOTAXIN',
'G_CSF', 'GM_CSF', 'IFN_G', 'IL10', 'IL_12P70', 'IL13',
'IL15', 'IL_17A', 'IL_1B', 'IL_1RA', 'IL2', 'IL4',
'IL5', 'IL6', 'IL7', 'IL8', 'IL9', 'MCP1', 'MIP_1A',
'MIP_1B', 'PDGF_BB', 'RANTES', 'TNF_A', 'VEGF', 'CTACK',
'GRO_A', 'HGF', 'IFN_A2', 'IL_12P40', 'IL16', 'IL18',
'IL_2RA', 'IL3', 'LIF', 'M_CSF', 'MCP3', 'MIF', 'MIG',
'SCF', 'SCGF_B', 'SDF_1A', 'TNF_B', 'TRAIL', 'B_NGF']
feat_list          = cont_list + bin_list
df_                = df_[['PID', 'months', 'Follow_up_Month', 'HIV']+feat_list]
df_org_            = df_.copy(deep=True)

df_[cont_list]     = f_get_Normalization(np.asarray(df_[cont_list]).astype(float), norm_mode)

pat_info, data     = f_construct_dataset(df_, feat_list)
_, data_org        = f_construct_dataset(df_org_, feat_list)

data_mi                  = np.zeros(np.shape(data))
data_mi[np.isnan(data)]  = 1
data_org[np.isnan(data)] = 0
data[np.isnan(data)]     = 0

x_dim           = np.shape(data)[2] # 1 + x_dim_cont + x_dim_bin (including delta)
x_dim_cont      = len(cont_list)
```

```
x_dim_bin       = len(bin_list)

last_meas       = pat_info[:,[3]]  #pat_info[:, 3] contains age at the last measurement
label           = pat_info[:,[2]]  #two competing risks
time            = pat_info[:,[1]]  #age when event occurred

num_Category    = int(np.max(pat_info[:, 1]) * 1.2)
#or specifically define larger than the max tte
num_Event       = len(np.unique(label)) - 1

if num_Event == 1:
label[np.where(label!=0)] = 1 #make single risk

mask1           = f_get_fc_mask1(last_meas, num_Event, num_Category)
mask2           = f_get_fc_mask2(time, label, num_Event, num_Category)
mask3           = f_get_fc_mask3(time, -1, num_Category)

DIM             = (x_dim, x_dim_cont, x_dim_bin)
DATA            = (data, time, label)
# DATA            = (data, data_org, time, label)
MASK            = (mask1, mask2, mask3)

return DIM, DATA, MASK, data_mi
```

3.2 Main analysis

```
# -*- coding: utf-8 -*-
"""
Created on Mon NOV 4 2024

@author: Sarah Ogutu
"""

# environment name "myenv"
_EPSILON = 1e-08

import numpy as np
import pandas as pd
import tensorflow as tf
import random
import os

from sklearn.model_selection import train_test_split

import import_data as impt

from class_DeepLongitudinal import Model_Longitudinal_Attention
```

```
from utils_eval             import c_index, brier_score
from utils_log              import save_logging, load_logging
from utils_helper           import f_get_minibatch, f_get_boosted_trainset


# In[ ]:


def _f_get_pred(sess, model, data, data_mi, pred_horizon):
'''
predictions based on the prediction time.
create new_data and new_mask2 that are available previous or equal to
the prediction time (no future measurements are used)
'''
new_data    = np.zeros(np.shape(data))
new_data_mi = np.zeros(np.shape(data_mi))

meas_time = np.concatenate([np.zeros([np.shape(data)[0], 1]),
np.cumsum(data[:, :, 0], axis=1)[:, :-1]], axis=1)

for i in range(np.shape(data)[0]):
last_meas = np.sum(meas_time[i, :] <= pred_horizon)

new_data[i, :last_meas, :]    = data[i, :last_meas, :]
new_data_mi[i, :last_meas, :] = data_mi[i, :last_meas, :]

return model.predict(new_data, new_data_mi)


def f_get_risk_predictions(sess, model, data_, data_mi_, pred_time, eval_time):

pred = _f_get_pred(sess, model, data_[[0]], data_mi_[[0]], 0)
_, num_Event, num_Category = np.shape(pred)

risk_all = {}
for k in range(num_Event):
risk_all[k] = np.zeros([np.shape(data_)[0], len(pred_time), len(eval_time)])

for p, p_time in enumerate(pred_time):
### PREDICTION
pred_horizon = int(p_time)
pred = _f_get_pred(sess, model, data_, data_mi_, pred_horizon)


for t, t_time in enumerate(eval_time):
eval_horizon = int(t_time) + pred_horizon
#if eval_horizon >= num_Category, output the maximum...
```

```
# calculate F(t | x, Y, t >= t_M) = \sum_{t_M <= \tau < t} P(\tau | x, Y, \tau > t_M)
risk = np.sum(pred[:,:,pred_horizon:(eval_horizon+1)], axis=2)
#risk score until eval_time
risk = risk / (np.sum(np.sum(pred[:,:,pred_horizon:], axis=2),
axis=1, keepdims=True) +_EPSILON)
#conditioniong on t > t_pred

for k in range(num_Event):
risk_all[k][:, p, t] = risk[:, k]

return risk_all


# ### 1. Import Dataset
# ##- Users must prepare dataset in csv format and modify 'import_data.py'
# ## following our examplar 'PBC2'

# In[ ]:


data_mode                    = 'DynDeepHit_data'
seed                         = 2408
np.random.seed(seed)
##### IMPORT DATASET
'''
num_Category            = max event/censoring time * 1.2
num_Event               = number of evetns i.e. len(np.unique(label))-1
max_length              = maximum number of measurements
x_dim                   = data dimension including delta (1 + num_features)
x_dim_cont              = dim of continuous features
x_dim_bin               = dim of binary features
mask1, mask2, mask3     = used for cause-specific network (FCNet structure)
'''

if data_mode == 'DynDeepHit_data':
(x_dim, x_dim_cont, x_dim_bin), (data, time, label),
(mask1, mask2, mask3), (data_mi) = impt.import_dataset(norm_mode = 'standard')

# This must be changed depending on the datasets, prediction/evaliation times of interest
pred_time = [3, 2*3, 3*3, 3*4, 3*5] # prediction time (in months)
eval_time = [3, 6, 9, 12, 15] # months evaluation time (for C-index and Brier-Score)
else:
print ('ERROR:  DATA_MODE NOT FOUND !!!')

_, num_Event, num_Category  = np.shape(mask1)  # dim of mask3: [subj, Num_Event, Num_Category]
max_length                  = np.shape(data)[1]


file_path = '{}'.format(data_mode)
```

```
if not os.path.exists(file_path):
os.makedirs(file_path)


# ### 2. Set Hyper-Parameters
# ##### - Play with your own hyper-parameters!

# In[ ]:


burn_in_mode                = 'ON' #{'ON', 'OFF'}
boost_mode                  = 'ON' #{'ON', 'OFF'}

##### HYPER-PARAMETERS
new_parser = {'mb_size': 32,

 'iteration_burn_in': 3000,
 'iteration': 5000,

 'keep_prob': 0.6,
 'lr_train': 1e-2,

 'h_dim_RNN': 100,
 'h_dim_FC' : 100,
 'num_layers_RNN':2,
 'num_layers_ATT':2,
 'num_layers_CS' :2,

 'RNN_type':'LSTM', #{'LSTM', 'GRU'}

 'FC_active_fn' : tf.nn.relu,
 'RNN_active_fn': tf.nn.tanh,

 'reg_W'         : 1e-5,
 'reg_W_out'     : 1e-5,

 'alpha' :1.0,
 'beta'  :0.1,
 'gamma' :1.0
}


# INPUT DIMENSIONS
input_dims                  = { 'x_dim'         : x_dim,
 'x_dim_cont'    : x_dim_cont,
 'x_dim_bin'     : x_dim_bin,
 'num_Event'     : num_Event,
 'num_Category'  : num_Category,
```

```
 'max_length'    : max_length }

# NETWORK HYPER-PARMETERS
network_settings            = { 'h_dim_RNN'         : new_parser['h_dim_RNN'],
 'h_dim_FC'          : new_parser['h_dim_FC'],
 'num_layers_RNN'    : new_parser['num_layers_RNN'],
 'num_layers_ATT'    : new_parser['num_layers_ATT'],
 'num_layers_CS'     : new_parser['num_layers_CS'],
 'RNN_type'          : new_parser['RNN_type'],
 'FC_active_fn'      : new_parser['FC_active_fn'],
 'RNN_active_fn'     : new_parser['RNN_active_fn'],
 'initial_W'         : tf.contrib.layers.xavier_initializer(),

 'reg_W'             : new_parser['reg_W'],
 'reg_W_out'         : new_parser['reg_W_out']
}


mb_size           = new_parser['mb_size']
iteration         = new_parser['iteration']
iteration_burn_in = new_parser['iteration_burn_in']

keep_prob         = new_parser['keep_prob']
lr_train          = new_parser['lr_train']

alpha             = new_parser['alpha']
beta              = new_parser['beta']
gamma             = new_parser['gamma']

# SAVE HYPERPARAMETERS
log_name = file_path + '/hyperparameters_log.txt'
save_logging(new_parser, log_name)


# ### 3. Split Dataset into Train/Valid/Test Sets

# In[ ]:


### TRAINING-TESTING SPLIT
(tr_data,te_data, tr_data_mi, te_data_mi, tr_time,te_time, tr_label,te_label,
tr_mask1,te_mask1, tr_mask2,te_mask2, tr_mask3,te_mask3)
= train_test_split(data, data_mi, time, label, mask1, mask2, mask3, test_size=0.2,
random_state=seed)

(tr_data,va_data, tr_data_mi, va_data_mi, tr_time,va_time, tr_label,va_label,
tr_mask1,va_mask1, tr_mask2,va_mask2, tr_mask3,va_mask3)
= train_test_split(tr_data, tr_data_mi, tr_time, tr_label, tr_mask1, tr_mask2, tr_mask3,
test_size=0.2, random_state=seed)
```

```
if boost_mode == 'ON':
tr_data, tr_data_mi, tr_time, tr_label, tr_mask1, tr_mask2,
tr_mask3 = f_get_boosted_trainset(tr_data, tr_data_mi, tr_time, tr_label,
tr_mask1, tr_mask2, tr_mask3)


# ### 4. Train the Network

# In[ ]:


##### CREATE DYNAMIC-DEEPFHT NETWORK
tf.reset_default_graph()

config = tf.ConfigProto()
config.gpu_options.allow_growth = True
sess = tf.Session(config=config)

model = Model_Longitudinal_Attention(sess, "Dyanmic-DeepHit", input_dims, network_settings)
saver = tf.train.Saver()

checkpoint_dir = 'DynDeepHit_data/model'
if not os.path.exists(checkpoint_dir):
os.makedirs(checkpoint_dir)

sess.run(tf.global_variables_initializer())

### TRAINING - BURN-IN
if burn_in_mode == 'ON':
print( "BURN-IN TRAINING ...")
for itr in range(iteration_burn_in):
x_mb, x_mi_mb, k_mb, t_mb, m1_mb, m2_mb, m3_mb
= f_get_minibatch(mb_size, tr_data, tr_data_mi, tr_label, tr_time,
tr_mask1, tr_mask2, tr_mask3)
DATA = (x_mb, k_mb, t_mb)
MISSING = (x_mi_mb)

_, loss_curr = model.train_burn_in(DATA, MISSING, keep_prob, lr_train)

if (itr+1)%1000 == 0:
print('itr: {:04d} | loss: {:.4f}'.format(itr+1, loss_curr))


### TRAINING - MAIN
print( "MAIN TRAINING ...")
min_valid = 0

for itr in range(iteration):
```

```
x_mb, x_mi_mb, k_mb, t_mb, m1_mb, m2_mb, m3_mb
= f_get_minibatch(mb_size, tr_data, tr_data_mi, tr_label, tr_time, tr_mask1,
tr_mask2, tr_mask3)
DATA = (x_mb, k_mb, t_mb)
MASK = (m1_mb, m2_mb, m3_mb)
MISSING = (x_mi_mb)
PARAMETERS = (alpha, beta, gamma)

_, loss_curr = model.train(DATA, MASK, MISSING, PARAMETERS, keep_prob, lr_train)

if (itr+1)%1000 == 0:
print('itr: {:04d} | loss: {:.4f}'.format(itr+1, loss_curr))

### VALIDATION  (based on average C-index of our interest)
if (itr+1)%1000 == 0:
risk_all = f_get_risk_predictions(sess, model, va_data,
va_data_mi, pred_time, eval_time)

for p, p_time in enumerate(pred_time):
pred_horizon = int(p_time)
val_result1 = np.zeros([num_Event, len(eval_time)])

for t, t_time in enumerate(eval_time):
eval_horizon = int(t_time) + pred_horizon
for k in range(num_Event):
val_result1[k, t] =
c_index(risk_all[k][:, p, t], va_time,
(va_label[:,0] == k+1).astype(int), eval_horizon)
#-1 for no event (not comparable)

if p == 0:
val_final1 = val_result1
else:
val_final1 = np.append(val_final1, val_result1, axis=0)

tmp_valid = np.mean(val_final1)
print(tmp_valid)
if tmp_valid >  min_valid:
min_valid = tmp_valid
saver.save(sess, file_path + '/model')
print( 'updated.... average c-index = ' + str('%.4f' %(tmp_valid)))

# ### 5. Test the Trained Network

# In[ ]:


saver.restore(sess, file_path + '/model')
```

```
risk_all = f_get_risk_predictions(sess, model, te_data, te_data_mi, pred_time, eval_time)

for p, p_time in enumerate(pred_time):
pred_horizon = int(p_time)
result1, result2 = np.zeros([num_Event, len(eval_time)]),
np.zeros([num_Event, len(eval_time)])

for t, t_time in enumerate(eval_time):
eval_horizon = int(t_time) + pred_horizon
for k in range(num_Event):
result1[k, t] =
c_index(risk_all[k][:, p, t], te_time,
(te_label[:,0] == k+1).astype(int), eval_horizon)
#-1 for no event (not comparable)
result2[k, t] = brier_score(risk_all[k][:, p, t], te_time,
(te_label[:,0] == k+1).astype(int), eval_horizon)
#-1 for no event (not comparable)

if p == 0:
final1, final2 = result1, result2
else:
final1, final2 = np.append(final1, result1, axis=0), np.append(final2, result2, axis=0)


row_header = []
for p_time in pred_time:
for t in range(num_Event):
row_header.append('pred_time {}: event_{}'.format(p_time,k+1))

col_header = []
for t_time in eval_time:
col_header.append('eval_time {}'.format(t_time))

# c-index result
df1 = pd.DataFrame(final1, index = row_header, columns=col_header)

# brier-score result
df2 = pd.DataFrame(final2, index = row_header, columns=col_header)

### PRINT RESULTS
print('=================================================')
print('-------------------------------------------------')
print('- C-INDEX: ')
print(df1)
print('-------------------------------------------------')
print('- BRIER-SCORE: ')
print(df2)
print('=================================================')
#########################################################################################
```

```
## trying to plot shapley values it did not work
import shap
explainer = shap.DeepExplainer(model, tr_data)
shap_values = explainer.shap_values(te_data)
shap.summary_plot(shap_values,te_data)

import shap
import numpy as np

# Initialize KernelExplainer
def predict_fn(data):
return model.predict(data)

explainer = shap.KernelExplainer(predict_fn,tr_data)
shap_values = explainer.shap_values(te_data)

# Plot the SHAP summary plot
shap.summary_plot(shap_values,te_data)


from sklearn.metrics import accuracy_score

# Define the evaluate function
def evaluate(model, data):
X_test = data.drop(columns=['target'])  # Assuming 'target' is the label column
y_test = data['target']  # Assuming 'target' is the label column
y_pred = model.predict(X_test)

# Calculate accuracy as the performance metric (can be replaced with other metrics)
performance = accuracy_score(y_test, y_pred)
return performance

# Example usage with the main code
base_performance = evaluate(model, te_data)

# Loop through each covariate and calculate importance
for covariate in te_data.columns:
if covariate != 'target':  # Avoid shuffling the target/label column
shuffled_data = te_data.copy()  # Create a copy to avoid modifying the original data
shuffled_data[covariate] = np.random.permutation(shuffled_data[covariate])
# Shuffle the covariate
performance = evaluate(model, shuffled_data)
importance = base_performance - performance
print(f"Covariate: {covariate}, Importance: {importance}")



import numpy as np
from lifelines.utils import concordance_index
```

```
# Function to evaluate the model performance (using C-index as an example)
def evaluate(model, data, time_horizon):
# Reshape data if necessary
if data.ndim == 2:
data = data.reshape(-1, 4, 68)  # Adjust based on your data's structure

# Predict survival probabilities
survival_probs = model.predict(data, time_horizon)  # Get predicted survival probabilities

# Compute C-index
performance = concordance_index(te_time, -survival_probs, te_label)
return performance

# Calculate base performance on the unshuffled data
base_performance = evaluate(model, te_data, time_horizon=365)

# Get variable importance by shuffling each covariate and re-evaluating
for covariate in range(te_data.shape[2]):  # Iterate through all covariates (features)
shuffled_data = te_data.copy()  # Make a copy of the data
shuffled_data[:, :, covariate] = np.random.permutation(shuffled_data[:, :, covariate])
# Shuffle the covariate
performance = evaluate(model, shuffled_data, time_horizon=365)
importance = base_performance - performance
print(f"Covariate {covariate} Importance: {importance}")
```