Open in Colab

In [ ]:

```
%cd /content/gdrive/MyDrive/CGM
!ls
# !gzip "/content/drive/MyDrive/CGM/dataset/HIGGS_6M.csv.gz" -d "/content/drive/MyDrive/CGM/dataset"
```

In [ ]:

```
!pip install -U fastbook
```

```
      Successfully uninstalled torch-1.8.0+cu101
  Found existing installation: torchvision 0.9.0+cu101
    Uninstalling torchvision-0.9.0+cu101:
      Successfully uninstalled torchvision-0.9.0+cu101
  Found existing installation: fastai 1.0.61
    Uninstalling fastai-1.0.61:
      Successfully uninstalled fastai-1.0.61
Successfully installed fastai-2.2.7 fastbook-0.0.16 fastcore-1.3.19 fastrelease-0.1.11 ghapi-0.1.16 nbdev-1.1.13 sentencepiece-0.1.95 torch-1.7.1 torchvision-0.8.2
```

In [ ]:

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
import fastbook
fastbook.setup_book()
from fastai.metrics import mse
import torch
import torch.nn as nn
import torch.optim as optim
import torch.utils.data
from torch.autograd import Variable
from torch.utils.data import TensorDataset
from torch.utils.data import DataLoader
from fastai import learner
from fastai.data import core
import time
from fastai.callback import schedule
import os
import numpy as np
from scipy import stats
import seaborn as sns
```

In [ ]:

```
df=pd.read_csv("dataset/HIGGS_6M.csv")
```

Preprocessing dataset

- **Standard Scaling**
- **Min Max Scaling**

In [ ]:

```
dataset=df.to_numpy()
X = dataset[:,1:]
Y = dataset[:,0].astype(int)
print(X[0],Y[0])
print(np.shape(X),np.shape(X[0]),np.shape(Y),np.shape(Y[0]))
from sklearn.preprocessing import StandardScaler
```

```
from sklearn.preprocessing import MinMaxScaler

scaler = StandardScaler()
categorical=[0,9,13,17,21]

for index in range(28):
  X[:,index]=scaler.fit_transform(X[:,index].reshape(-1,1)).reshape(-1)

scaler = MinMaxScaler()
for index in range(28):
  X[:,index]=scaler.fit_transform(X[:,index].reshape(-1,1)).reshape(-1)
```

Compiling DAE Model

In [ ]:

```
class AE_4D_300_LeakyReLU(nn.Module):
    def __init__(self, n_features=28,bottle_neck=8):
        super(AE_4D_300_LeakyReLU, self).__init__()
        self.en1 = nn.Linear(n_features, 300)
        self.en2 = nn.Linear(300, 200)
        self.en3 = nn.Linear(200, 100)
        self.en4 = nn.Linear(100,50)
        self.en5 = nn.Linear(50, bottle_neck)
        self.de1 = nn.Linear(bottle_neck, 50)
        self.de2 = nn.Linear(50, 100)
        self.de3 = nn.Linear(100, 200)
        self.de4 = nn.Linear(200,300)
        self.de5 = nn.Linear(300, n_features)
        self.tanh = nn.Tanh()

    def encode(self, x):
        return self.en5(self.tanh(self.en4(self.tanh(self.en3(self.tanh(self.en2(self.tan
h(self.en1(x)))))))))

    def decode(self, x):
        return self.de5(self.tanh(self.de4(self.tanh(self.de3(self.tanh(self.de2(self.tan
h(self.de1(self.tanh(x)))))))))

    def forward(self, x):
        z = self.encode(x)
        return self.decode(z)

model = AE_4D_300_LeakyReLU()
model.to('cpu')
```

Out[ ]:

```
AE_4D_300_LeakyReLU(
  (en1): Linear(in_features=28, out_features=300, bias=True)
  (en2): Linear(in_features=300, out_features=200, bias=True)
  (en3): Linear(in_features=200, out_features=100, bias=True)
  (en4): Linear(in_features=100, out_features=50, bias=True)
  (en5): Linear(in_features=50, out_features=8, bias=True)
  (de1): Linear(in_features=8, out_features=50, bias=True)
  (de2): Linear(in_features=50, out_features=100, bias=True)
  (de3): Linear(in_features=100, out_features=200, bias=True)
  (de4): Linear(in_features=200, out_features=300, bias=True)
  (de5): Linear(in_features=300, out_features=28, bias=True)
  (tanh): Tanh()
)
```

Loading DAE Model from Drive

In [ ]:

```
model_inf = AE_4D_300_LeakyReLU()
model_inf.to('cpu')
```

```
model_inf.load_state_dict(torch.load("dae_model/model.pth"))
```

Out[ ]:

```
<All keys matched successfully>
```

Generating Encoded Input from DAE Model

You can skip this, next cell loads this from Drive

In [ ]:

```
from tqdm import tqdm

for i in tqdm(range(np.shape(X)[0]//10**6)):
  data = torch.tensor(X[i*(10**6):(i+1)*(10**6)], dtype=torch.float)
  pred = model_inf.encode(data)
  pred = pred.detach().numpy()
  pred = pred.T
  # pred = np.reshape(pred,(np.shape(pred)[0],1))
  if i is 0:
    x=pred
  # save(str(i)+'.npy', pred)
  else:
    x=np.concatenate((x,pred),axis=-1)
data = torch.tensor(X[(i+1)*(10**6):], dtype=torch.float)
pred = model_inf.encode(data)
pred = pred.detach().numpy()
pred = pred.T
x=np.concatenate((x,pred),axis=-1)
```

In [ ]:

```
print(np.shape(x))
x=x.T
print(np.shape(x))
np.save("encoded_x_6M.npy",x)
X = x
del x
```

```
(8, 5999999)
(5999999, 8)
```

In [ ]:

```
X = np.load("encoded_x_6M.npy")
print(np.shape(X), np.shape(Y))
```

```
(5999999, 8) (5999999,)
```

Training XGB Classifier on Encoded input

You can skip training last cell loads the model

In [ ]:

```
import xgboost
from numpy import loadtxt
from xgboost import XGBClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
```

In [ ]:

```
seed = 7
test_size = 0.08
X_train, X_valid, y_train, y_valid = train_test_split(X, Y, test_size=test_size, random_
state=seed)
```

```
# Might try a different test split also to check overfitting
```

In [ ]:

```
eval_set = [(X_valid, y_valid)]
model = XGBClassifier(n_estimators=300, learning_rate=0.1, max_depth=5, gamma=0.1, subsam
ple=0.8, colsample_bytree=0.8)
eval_set = [(X_valid, y_valid)]
model.fit(X_train, y_train, eval_metric="auc", eval_set=eval_set, verbose=True, early_st
opping_rounds=5)
```

```
[18:41:09] WARNING: /workspace/src/learner.cc:686: Tree method is automatically selected
to be 'approx' for faster speed. To use old behavior (exact greedy algorithm on single ma
chine), set tree_method to 'exact'.
[0] validation_0-auc:0.566642
Will train until validation_0-auc hasn't improved in 5 rounds.
[1] validation_0-auc:0.574211
[2] validation_0-auc:0.584001
[3] validation_0-auc:0.587706
[4] validation_0-auc:0.586605
[5] validation_0-auc:0.59076
[6] validation_0-auc:0.591432
[7] validation_0-auc:0.591808
[8] validation_0-auc:0.592128
[9] validation_0-auc:0.593737
[10] validation_0-auc:0.59503
[11] validation_0-auc:0.595247
[12] validation_0-auc:0.595731
[13] validation_0-auc:0.596987
[14] validation_0-auc:0.599021
[15] validation_0-auc:0.599298
[16] validation_0-auc:0.599405
[17] validation_0-auc:0.599925
[18] validation_0-auc:0.600095
[19] validation_0-auc:0.600445
[20] validation_0-auc:0.600821
[21] validation_0-auc:0.601451
[22] validation_0-auc:0.602242
[23] validation_0-auc:0.603133
[24] validation_0-auc:0.603468
[25] validation_0-auc:0.604218
[26] validation_0-auc:0.604766
[27] validation_0-auc:0.605245
[28] validation_0-auc:0.605708
[29] validation_0-auc:0.606007
[30] validation_0-auc:0.606491
[31] validation_0-auc:0.607231
[32] validation_0-auc:0.607708
[33] validation_0-auc:0.608399
[34] validation_0-auc:0.60879
[35] validation_0-auc:0.609354
[36] validation_0-auc:0.609785
[37] validation_0-auc:0.610237
[38] validation_0-auc:0.610733
[39] validation_0-auc:0.610944
[40] validation_0-auc:0.611082
[41] validation_0-auc:0.612208
[42] validation_0-auc:0.612396
[43] validation_0-auc:0.612654
[44] validation_0-auc:0.612858
[45] validation_0-auc:0.61312
[46] validation_0-auc:0.613304
[47] validation_0-auc:0.613689
[48] validation_0-auc:0.614082
[49] validation_0-auc:0.614407
[50] validation_0-auc:0.614764
[51] validation_0-auc:0.615224
[52] validation_0-auc:0.615441
[53] validation_0-auc:0.615886
[54] validation_0-auc:0.616352
[55] validation_0-auc:0.616398
[56] validation_0-auc:0.616553
```

```
[57] validation_0-auc:0.616779
[58] validation_0-auc:0.616899
[59] validation_0-auc:0.617028
[60] validation_0-auc:0.617269
[61] validation_0-auc:0.617462
[62] validation_0-auc:0.618214
[63] validation_0-auc:0.61859
[64] validation_0-auc:0.618881
[65] validation_0-auc:0.619027
[66] validation_0-auc:0.61936
[67] validation_0-auc:0.619673
[68] validation_0-auc:0.61988
[69] validation_0-auc:0.620103
[70] validation_0-auc:0.620319
[71] validation_0-auc:0.620468
[72] validation_0-auc:0.620598
[73] validation_0-auc:0.620805
[74] validation_0-auc:0.62102
[75] validation_0-auc:0.621064
[76] validation_0-auc:0.621348
[77] validation_0-auc:0.621415
[78] validation_0-auc:0.621646
[79] validation_0-auc:0.62225
[80] validation_0-auc:0.622437
[81] validation_0-auc:0.622551
[82] validation_0-auc:0.622628
[83] validation_0-auc:0.62283
[84] validation_0-auc:0.623218
[85] validation_0-auc:0.623681
[86] validation_0-auc:0.624013
[87] validation_0-auc:0.624062
[88] validation_0-auc:0.624233
[89] validation_0-auc:0.624409
[90] validation_0-auc:0.62451
[91] validation_0-auc:0.624805
[92] validation_0-auc:0.624968
[93] validation_0-auc:0.625039
[94] validation_0-auc:0.62506
[95] validation_0-auc:0.625216
[96] validation_0-auc:0.625327
[97] validation_0-auc:0.625465
[98] validation_0-auc:0.625678
[99] validation_0-auc:0.625763
[100] validation_0-auc:0.625837
[101] validation_0-auc:0.626049
[102] validation_0-auc:0.626092
[103] validation_0-auc:0.626381
[104] validation_0-auc:0.626419
[105] validation_0-auc:0.626451
[106] validation_0-auc:0.626522
[107] validation_0-auc:0.626747
[108] validation_0-auc:0.626891
[109] validation_0-auc:0.627102
[110] validation_0-auc:0.627239
[111] validation_0-auc:0.627397
[112] validation_0-auc:0.627743
[113] validation_0-auc:0.627835
[114] validation_0-auc:0.627926
[115] validation_0-auc:0.627958
[116] validation_0-auc:0.628076
[117] validation_0-auc:0.628369
[118] validation_0-auc:0.628429
[119] validation_0-auc:0.628549
[120] validation_0-auc:0.628821
[121] validation_0-auc:0.628966
[122] validation_0-auc:0.629176
[123] validation_0-auc:0.629423
[124] validation_0-auc:0.629628
[125] validation_0-auc:0.629687
[126] validation_0-auc:0.629909
[127] validation_0-auc:0.629969
[128] validation_0-auc:0.630234
```

```
[129] validation_0-auc:0.630336
[130] validation_0-auc:0.630355
[131] validation_0-auc:0.630415
[132] validation_0-auc:0.630527
[133] validation_0-auc:0.630864
[134] validation_0-auc:0.630975
[135] validation_0-auc:0.630991
[136] validation_0-auc:0.631121
[137] validation_0-auc:0.631307
[138] validation_0-auc:0.63139
[139] validation_0-auc:0.631416
[140] validation_0-auc:0.63157
[141] validation_0-auc:0.631728
[142] validation_0-auc:0.631806
[143] validation_0-auc:0.631862
[144] validation_0-auc:0.631951
[145] validation_0-auc:0.632094
[146] validation_0-auc:0.632127
[147] validation_0-auc:0.632356
[148] validation_0-auc:0.632415
[149] validation_0-auc:0.632489
[150] validation_0-auc:0.632604
[151] validation_0-auc:0.632673
[152] validation_0-auc:0.63273
[153] validation_0-auc:0.632779
[154] validation_0-auc:0.632836
[155] validation_0-auc:0.633024
[156] validation_0-auc:0.633142
[157] validation_0-auc:0.633258
[158] validation_0-auc:0.633351
[159] validation_0-auc:0.633435
[160] validation_0-auc:0.633604
[161] validation_0-auc:0.633745
[162] validation_0-auc:0.633785
[163] validation_0-auc:0.633935
[164] validation_0-auc:0.633997
[165] validation_0-auc:0.63406
[166] validation_0-auc:0.634317
[167] validation_0-auc:0.634463
[168] validation_0-auc:0.634491
[169] validation_0-auc:0.634526
[170] validation_0-auc:0.634688
[171] validation_0-auc:0.634741
[172] validation_0-auc:0.634793
[173] validation_0-auc:0.634841
[174] validation_0-auc:0.634924
[175] validation_0-auc:0.635168
[176] validation_0-auc:0.635328
[177] validation_0-auc:0.63543
[178] validation_0-auc:0.635494
[179] validation_0-auc:0.635504
[180] validation_0-auc:0.635539
[181] validation_0-auc:0.635595
[182] validation_0-auc:0.635707
[183] validation_0-auc:0.635844
[184] validation_0-auc:0.635977
[185] validation_0-auc:0.636053
[186] validation_0-auc:0.636139
[187] validation_0-auc:0.636223
[188] validation_0-auc:0.636334
[189] validation_0-auc:0.636505
[190] validation_0-auc:0.636548
[191] validation_0-auc:0.636605
[192] validation_0-auc:0.63673
[193] validation_0-auc:0.636792
[194] validation_0-auc:0.636805
[195] validation_0-auc:0.636946
[196] validation_0-auc:0.63707
[197] validation_0-auc:0.637181
[198] validation_0-auc:0.637288
[199] validation_0-auc:0.637383
[200] validation_0-auc:0.637402
```

```
[201] validation_0-auc:0.637522
[202] validation_0-auc:0.637629
[203] validation_0-auc:0.637784
[204] validation_0-auc:0.637892
[205] validation_0-auc:0.637918
[206] validation_0-auc:0.637986
[207] validation_0-auc:0.63806
[208] validation_0-auc:0.638173
[209] validation_0-auc:0.638258
[210] validation_0-auc:0.638335
[211] validation_0-auc:0.638353
[212] validation_0-auc:0.638434
[213] validation_0-auc:0.638487
[214] validation_0-auc:0.638634
[215] validation_0-auc:0.638665
[216] validation_0-auc:0.638712
[217] validation_0-auc:0.638777
[218] validation_0-auc:0.638873
[219] validation_0-auc:0.638927
[220] validation_0-auc:0.639026
[221] validation_0-auc:0.639107
[222] validation_0-auc:0.639125
[223] validation_0-auc:0.639196
[224] validation_0-auc:0.639241
[225] validation_0-auc:0.639387
[226] validation_0-auc:0.63943
[227] validation_0-auc:0.639528
[228] validation_0-auc:0.639565
[229] validation_0-auc:0.639664
[230] validation_0-auc:0.639726
[231] validation_0-auc:0.63978
[232] validation_0-auc:0.639828
[233] validation_0-auc:0.639845
[234] validation_0-auc:0.639982
[235] validation_0-auc:0.640094
[236] validation_0-auc:0.640187
[237] validation_0-auc:0.640355
[238] validation_0-auc:0.640442
[239] validation_0-auc:0.640523
[240] validation_0-auc:0.640599
[241] validation_0-auc:0.640673
[242] validation_0-auc:0.640774
[243] validation_0-auc:0.640842
[244] validation_0-auc:0.640944
[245] validation_0-auc:0.640988
[246] validation_0-auc:0.641075
[247] validation_0-auc:0.641129
[248] validation_0-auc:0.641203
[249] validation_0-auc:0.641265
[250] validation_0-auc:0.641296
[251] validation_0-auc:0.641387
[252] validation_0-auc:0.64145
[253] validation_0-auc:0.64152
[254] validation_0-auc:0.641546
[255] validation_0-auc:0.641575
[256] validation_0-auc:0.641675
[257] validation_0-auc:0.641732
[258] validation_0-auc:0.641877
[259] validation_0-auc:0.641915
[260] validation_0-auc:0.64196
[261] validation_0-auc:0.642022
[262] validation_0-auc:0.642219
[263] validation_0-auc:0.642303
[264] validation_0-auc:0.642349
[265] validation_0-auc:0.642452
[266] validation_0-auc:0.642514
[267] validation_0-auc:0.642591
[268] validation_0-auc:0.642647
[269] validation_0-auc:0.642786
[270] validation_0-auc:0.642837
[271] validation_0-auc:0.642961
[272] validation_0-auc:0.642974
```

```
[273] validation_0-auc:0.643043
[274] validation_0-auc:0.643095
[275] validation_0-auc:0.643191
[276] validation_0-auc:0.643245
[277] validation_0-auc:0.643287
[278] validation_0-auc:0.643317
[279] validation_0-auc:0.643368
[280] validation_0-auc:0.643435
[281] validation_0-auc:0.643496
[282] validation_0-auc:0.643539
[283] validation_0-auc:0.643582
[284] validation_0-auc:0.643595
[285] validation_0-auc:0.643659
[286] validation_0-auc:0.643737
[287] validation_0-auc:0.643779
[288] validation_0-auc:0.643862
[289] validation_0-auc:0.643911
[290] validation_0-auc:0.643961
[291] validation_0-auc:0.644024
[292] validation_0-auc:0.644064
[293] validation_0-auc:0.644079
[294] validation_0-auc:0.644121
[295] validation_0-auc:0.644172
[296] validation_0-auc:0.644245
[297] validation_0-auc:0.644311
[298] validation_0-auc:0.644319
[299] validation_0-auc:0.6444
```

Out[ ]:

```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=0.8, gamma=0.1,
              learning_rate=0.1, max_delta_step=0, max_depth=5,
              min_child_weight=1, missing=None, n_estimators=300, n_jobs=1,
              nthread=None, objective='binary:logistic', random_state=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
              silent=None, subsample=0.8, verbosity=1)
```

In [ ]:

```
y_pred = model.predict(X_valid)
predictions = [round(value) for value in y_pred]
accuracy = accuracy_score(y_valid, predictions)
print("Accuracy: %.2f%%" % (accuracy * 100.0))
```

```
Accuracy: 60.74%
```

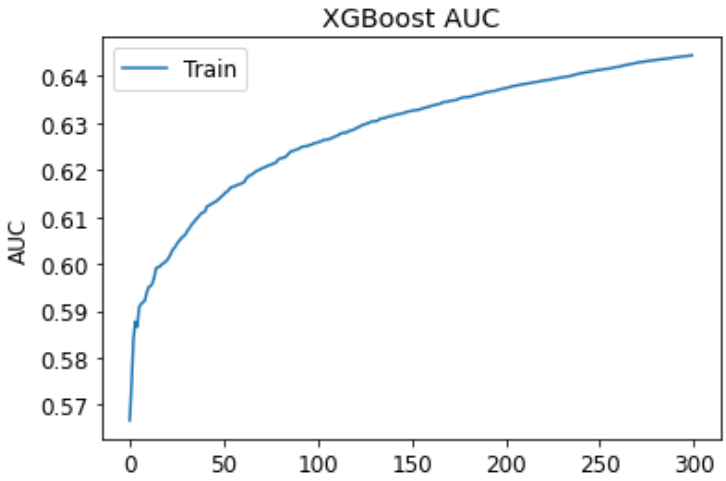
In [ ]:

```
from matplotlib import pyplot

results = model.evals_result()
print(results)
epochs = len(results['validation_0']['auc'])
x_axis = range(0, epochs)
# plot log loss
fig, ax = plt.subplots()
ax.plot(x_axis, results['validation_0']['auc'], label='Train')
ax.legend()
pyplot.ylabel('AUC')
pyplot.title('XGBoost AUC')
pyplot.show()
```

```
{'validation_0': {'auc': [0.566642, 0.574211, 0.584001, 0.587706, 0.586605, 0.59076, 0.59
1432, 0.591808, 0.592128, 0.593737, 0.59503, 0.595247, 0.595731, 0.596987, 0.599021, 0.59
9298, 0.599405, 0.599925, 0.600095, 0.600445, 0.600821, 0.601451, 0.602242, 0.603133, 0.6
03468, 0.604218, 0.604766, 0.605245, 0.605708, 0.606007, 0.606491, 0.607231, 0.607708, 0.
608399, 0.60879, 0.609354, 0.609785, 0.610237, 0.610733, 0.610944, 0.611082, 0.612208, 0.
612396, 0.612654, 0.612858, 0.61312, 0.613304, 0.613689, 0.614082, 0.614407, 0.614764, 0.
615224, 0.615441, 0.615886, 0.616352, 0.616398, 0.616553, 0.616779, 0.616899, 0.617028, 0
.617269, 0.617462, 0.618214, 0.61859, 0.618881, 0.619027, 0.61936, 0.619673, 0.61988, 0.6
20103, 0.620319, 0.620468, 0.620598, 0.620805, 0.62102, 0.621064, 0.621348, 0.621415, 0.6
21646, 0.62225, 0.622437, 0.622551, 0.622628, 0.62283, 0.623218, 0.623681, 0.624013, 0.62
```

4062, 0.624233, 0.624409, 0.62451, 0.624805, 0.624968, 0.625039, 0.62506, 0.625216, 0.625
327, 0.625465, 0.625678, 0.625763, 0.625837, 0.626049, 0.626092, 0.626381, 0.626419, 0.62
6451, 0.626522, 0.626747, 0.626891, 0.627102, 0.627239, 0.627397, 0.627743, 0.627835, 0.6
27926, 0.627958, 0.628076, 0.628369, 0.628429, 0.628549, 0.628821, 0.628966, 0.629176, 0.
629423, 0.629628, 0.629687, 0.629909, 0.629969, 0.630234, 0.630336, 0.630355, 0.630415, 0
.630527, 0.630864, 0.630975, 0.630991, 0.631121, 0.631307, 0.63139, 0.631416, 0.63157, 0.
631728, 0.631806, 0.631862, 0.631951, 0.632094, 0.632127, 0.632356, 0.632415, 0.632489, 0
.632604, 0.632673, 0.63273, 0.632779, 0.632836, 0.633024, 0.633142, 0.633258, 0.633351, 0
.633435, 0.633604, 0.633745, 0.633785, 0.633935, 0.633997, 0.63406, 0.634317, 0.634463, 0
.634491, 0.634526, 0.634688, 0.634741, 0.634793, 0.634841, 0.634924, 0.635168, 0.635328,
0.63543, 0.635494, 0.635504, 0.635539, 0.635595, 0.635707, 0.635844, 0.635977, 0.636053,
0.636139, 0.636223, 0.636334, 0.636505, 0.636548, 0.636605, 0.63673, 0.636792, 0.636805,
0.636946, 0.63707, 0.637181, 0.637288, 0.637383, 0.637402, 0.637522, 0.637629, 0.637784,
0.637892, 0.637918, 0.637986, 0.63806, 0.638173, 0.638258, 0.638335, 0.638353, 0.638434,
0.638487, 0.638634, 0.638665, 0.638712, 0.638777, 0.638873, 0.638927, 0.639026, 0.639107,
0.639125, 0.639196, 0.639241, 0.639387, 0.63943, 0.639528, 0.639565, 0.639664, 0.639726,
0.63978, 0.639828, 0.639845, 0.639982, 0.640094, 0.640187, 0.640355, 0.640442, 0.640523,
0.640599, 0.640673, 0.640774, 0.640842, 0.640944, 0.640988, 0.641075, 0.641129, 0.641203,
0.641265, 0.641296, 0.641387, 0.64145, 0.64152, 0.641546, 0.641575, 0.641675, 0.641732, 0
.641877, 0.641915, 0.64196, 0.642022, 0.642219, 0.642303, 0.642349, 0.642452, 0.642514, 0
.642591, 0.642647, 0.642786, 0.642837, 0.642961, 0.642974, 0.643043, 0.643095, 0.643191,
0.643245, 0.643287, 0.643317, 0.643368, 0.643435, 0.643496, 0.643539, 0.643582, 0.643595,
0.643659, 0.643737, 0.643779, 0.643862, 0.643911, 0.643961, 0.644024, 0.644064, 0.644079,
0.644121, 0.644172, 0.644245, 0.644311, 0.644319, 0.6444]}}



In [ ]:

```
import pickle
pickle.dump(model, open("xgb_encoded.pickle.dat", "wb"))
```

In [ ]:

```
"""
Checking whether load and save is working okay
"""
loaded_model = pickle.load(open("xgb_encoded.pickle.dat", "rb"))
y_pred = loaded_model.predict(X_valid)
predictions = [round(value) for value in y_pred]
accuracy = accuracy_score(y_valid, predictions)
print("Accuracy: %.2f%%" % (accuracy * 100.0))
```

```
Accuracy: 60.74%
```

Incremental Training for XGB

Not needed anymore, found another fix, read apendix of technical report

In [ ]:

```
# import xgboost
# from numpy import loadtxt
# from xgboost import XGBClassifier
```

```
# from sklearn.model_selection import train_test_split
# from sklearn.metrics import accuracy_score

# BATCH_SIZE = 10**4
# def trainXGB():
#   model = None

#   for i in range(np.shape(X)[0]//BATCH_SIZE):
#     print("Iteration: "+str(i))

#     x = X[i*BATCH_SIZE:(i+1)*BATCH_SIZE]
#     y = Y[i*BATCH_SIZE:(i+1)*BATCH_SIZE]

#     x = model_inf(torch.tensor(x, dtype=torch.float)).detach().numpy()
#     y = y

#     seed = 7
#     test_size = 0.08
#     X_train, X_valid, y_train, y_valid = train_test_split(x, y, test_size=test_size, ra
ndom_state=seed)

#     eval_set = [(X_valid, y_valid)]
#     model = xgboost.train({
#             'update':'refresh',
#             'process_type': 'update',
#             'refresh_leaf': True,
#             'silent': False,
#             'obj': 'auc',
#         }, dtrain=xgboost.DMatrix(X_train, y_train), xgb_model=model)
#     y_pred = model.predict(xgboost.DMatrix(X_valid))
#     predictions = [round(value) for value in y_pred]
#     accuracy = accuracy_score(y_valid, predictions)
#     print("Accuracy: %.2f%%" % (accuracy * 100.0))

#   return model
```

In [ ]:

```
# trained_model = trainXGB()
```