AML Final Project Phase-3 By Group 21.

HCDR- PyTorch Deep Learning

In [1]:
```
import numpy as np
import pandas as pd
import os
import datetime
import random
import string
import torch
import torch.nn as nn
from torch.utils.data import Dataset, TensorDataset, DataLoader
from sklearn import preprocessing
from sklearn.metrics import roc_auc_score
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.feature_selection import VarianceThreshold
```

Get Data

In [2]:
```
%%time
ds_names = ("application_train", "application_test", "bureau","bureau_balance","credit_card_balance","installme
            "previous_application","POS_CASH_balance")

def load_data(in_path, name):
    df = pd.read_csv(in_path)
    print(f"{name}: shape is {df.shape}")
    print(df.info())
    display(df.head(5))
    return df

DATA_DIR='./Data/'

datasets={}
for ds_name in ds_names:
    datasets[ds_name] = load_data(os.path.join(DATA_DIR, f'{ds_name}.csv'), ds_name)
```

```
application_train: shape is (307511, 122)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 307511 entries, 0 to 307510
Columns: 122 entries, SK_ID_CURR to AMT_REQ_CREDIT_BUREAU_YEAR
dtypes: float64(65), int64(41), object(16)
memory usage: 286.2+ MB
None
```

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOM
0	100002	1	Cash loans	M	N	Y	0	
1	100003	0	Cash loans	F	N	N	0	
2	100004	0	Revolving loans	M	Y	Y	0	
3	100006	0	Cash loans	F	N	Y	0	
4	100007	0	Cash loans	M	N	Y	0	

5 rows × 122 columns

```
application_test: shape is (48744, 121)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 48744 entries, 0 to 48743
Columns: 121 entries, SK_ID_CURR to AMT_REQ_CREDIT_BUREAU_YEAR
dtypes: float64(65), int64(40), object(16)
memory usage: 45.0+ MB
None
```

	SK_ID_CURR	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_TOTAL
0	100001	Cash loans	F	N	Y	0	135000.0
1	100005	Cash loans	M	N	Y	0	99000.0
2	100013	Cash loans	M	Y	Y	0	202500.0
3	100028	Cash loans	F	N	Y	2	315000.0
4	100038	Cash loans	M	Y	N	1	180000.0

5 rows × 121 columns

```
bureau: shape is (1716428, 17)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1716428 entries, 0 to 1716427
Data columns (total 17 columns):
 #   Column                  Dtype
---  ------                  -----
```

```
 0   SK_ID_CURR              int64
 1   SK_ID_BUREAU            int64
 2   CREDIT_ACTIVE           object
 3   CREDIT_CURRENCY         object
 4   DAYS_CREDIT             int64
 5   CREDIT_DAY_OVERDUE      int64
 6   DAYS_CREDIT_ENDDATE     float64
 7   DAYS_ENDDATE_FACT       float64
 8   AMT_CREDIT_MAX_OVERDUE  float64
 9   CNT_CREDIT_PROLONG      int64
 10  AMT_CREDIT_SUM          float64
 11  AMT_CREDIT_SUM_DEBT     float64
 12  AMT_CREDIT_SUM_LIMIT    float64
 13  AMT_CREDIT_SUM_OVERDUE  float64
 14  CREDIT_TYPE             object
 15  DAYS_CREDIT_UPDATE      int64
 16  AMT_ANNUITY             float64
dtypes: float64(8), int64(6), object(3)
memory usage: 222.6+ MB
None
```

	SK_ID_CURR	SK_ID_BUREAU	CREDIT_ACTIVE	CREDIT_CURRENCY	DAYS_CREDIT	CREDIT_DAY_OVERDUE	DAYS_CREDIT_ENDDATE
0	215354	5714462	Closed	currency 1	-497	0	-153.0
1	215354	5714463	Active	currency 1	-208	0	1075.0
2	215354	5714464	Active	currency 1	-203	0	528.0
3	215354	5714465	Active	currency 1	-203	0	NaN
4	215354	5714466	Active	currency 1	-629	0	1197.0

```
bureau_balance: shape is (27299925, 3)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 27299925 entries, 0 to 27299924
Data columns (total 3 columns):
 #   Column          Dtype
---  ------          -----
 0   SK_ID_BUREAU    int64
 1   MONTHS_BALANCE  int64
 2   STATUS          object
dtypes: int64(2), object(1)
memory usage: 624.8+ MB
None
```

	SK_ID_BUREAU	MONTHS_BALANCE	STATUS

	SK_ID_BUREAU	MONTHS_BALANCE	STATUS
0	5715448	0	C
1	5715448	-1	C
2	5715448	-2	C
3	5715448	-3	C
4	5715448	-4	C

```
credit_card_balance: shape is (3840312, 23)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3840312 entries, 0 to 3840311
Data columns (total 23 columns):
 #   Column                       Dtype  
---  ------                       -----  
 0   SK_ID_PREV                   int64  
 1   SK_ID_CURR                   int64  
 2   MONTHS_BALANCE               int64  
 3   AMT_BALANCE                  float64
 4   AMT_CREDIT_LIMIT_ACTUAL      int64  
 5   AMT_DRAWINGS_ATM_CURRENT     float64
 6   AMT_DRAWINGS_CURRENT         float64
 7   AMT_DRAWINGS_OTHER_CURRENT   float64
 8   AMT_DRAWINGS_POS_CURRENT     float64
 9   AMT_INST_MIN_REGULARITY      float64
 10  AMT_PAYMENT_CURRENT          float64
 11  AMT_PAYMENT_TOTAL_CURRENT    float64
 12  AMT_RECEIVABLE_PRINCIPAL     float64
 13  AMT_RECIVABLE                float64
 14  AMT_TOTAL_RECEIVABLE         float64
 15  CNT_DRAWINGS_ATM_CURRENT     float64
 16  CNT_DRAWINGS_CURRENT         int64  
 17  CNT_DRAWINGS_OTHER_CURRENT   float64
 18  CNT_DRAWINGS_POS_CURRENT     float64
 19  CNT_INSTALMENT_MATURE_CUM    float64
 20  NAME_CONTRACT_STATUS         object 
 21  SK_DPD                       int64  
 22  SK_DPD_DEF                   int64  
dtypes: float64(15), int64(7), object(1)
memory usage: 673.9+ MB
None
```

	SK_ID_PREV	SK_ID_CURR	MONTHS_BALANCE	AMT_BALANCE	AMT_CREDIT_LIMIT_ACTUAL	AMT_DRAWINGS_ATM_CURRENT	AM1
0	2562384	378907	-6	56.970	135000	0.0	
1	2582071	363914	-1	63975.555	45000	2250.0	

	SK_ID_PREV	SK_ID_CURR	MONTHS_BALANCE	AMT_BALANCE	AMT_CREDIT_LIMIT_ACTUAL	AMT_DRAWINGS_ATM_CURRENT	AMT
2	1740877	371185	-7	31815.225	450000	0.0	
3	1389973	337855	-4	236572.110	225000	2250.0	
4	1891521	126868	-1	453919.455	450000	0.0	

5 rows × 23 columns

```
installments_payments: shape is (13605401, 8)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 13605401 entries, 0 to 13605400
Data columns (total 8 columns):
 #   Column                  Dtype  
---  ------                  -----  
 0   SK_ID_PREV              int64  
 1   SK_ID_CURR              int64  
 2   NUM_INSTALMENT_VERSION  float64
 3   NUM_INSTALMENT_NUMBER   int64  
 4   DAYS_INSTALMENT         float64
 5   DAYS_ENTRY_PAYMENT      float64
 6   AMT_INSTALMENT          float64
 7   AMT_PAYMENT             float64
dtypes: float64(5), int64(3)
memory usage: 830.4 MB
None
```

	SK_ID_PREV	SK_ID_CURR	NUM_INSTALMENT_VERSION	NUM_INSTALMENT_NUMBER	DAYS_INSTALMENT	DAYS_ENTRY_PAYMENT
0	1054186	161674	1.0	6	-1180.0	-1187.0
1	1330831	151639	0.0	34	-2156.0	-2156.0
2	2085231	193053	2.0	1	-63.0	-63.0
3	2452527	199697	1.0	3	-2418.0	-2426.0
4	2714724	167756	1.0	2	-1383.0	-1366.0

```
previous_application: shape is (1670214, 37)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1670214 entries, 0 to 1670213
Data columns (total 37 columns):
 #   Column                       Non-Null Count    Dtype  
---  ------                       --------------    -----  
 0   SK_ID_PREV                   1670214 non-null  int64  
 1   SK_ID_CURR                   1670214 non-null  int64  
```

```
 2   NAME_CONTRACT_TYPE           1670214 non-null  object
 3   AMT_ANNUITY                  1297979 non-null  float64
 4   AMT_APPLICATION              1670214 non-null  float64
 5   AMT_CREDIT                   1670213 non-null  float64
 6   AMT_DOWN_PAYMENT             774370 non-null   float64
 7   AMT_GOODS_PRICE              1284699 non-null  float64
 8   WEEKDAY_APPR_PROCESS_START   1670214 non-null  object
 9   HOUR_APPR_PROCESS_START      1670214 non-null  int64
 10  FLAG_LAST_APPL_PER_CONTRACT  1670214 non-null  object
 11  NFLAG_LAST_APPL_IN_DAY       1670214 non-null  int64
 12  RATE_DOWN_PAYMENT            774370 non-null   float64
 13  RATE_INTEREST_PRIMARY        5951 non-null     float64
 14  RATE_INTEREST_PRIVILEGED     5951 non-null     float64
 15  NAME_CASH_LOAN_PURPOSE       1670214 non-null  object
 16  NAME_CONTRACT_STATUS         1670214 non-null  object
 17  DAYS_DECISION                1670214 non-null  int64
 18  NAME_PAYMENT_TYPE            1670214 non-null  object
 19  CODE_REJECT_REASON           1670214 non-null  object
 20  NAME_TYPE_SUITE              849809 non-null   object
 21  NAME_CLIENT_TYPE             1670214 non-null  object
 22  NAME_GOODS_CATEGORY          1670214 non-null  object
 23  NAME_PORTFOLIO               1670214 non-null  object
 24  NAME_PRODUCT_TYPE            1670214 non-null  object
 25  CHANNEL_TYPE                 1670214 non-null  object
 26  SELLERPLACE_AREA             1670214 non-null  int64
 27  NAME_SELLER_INDUSTRY         1670214 non-null  object
 28  CNT_PAYMENT                  1297984 non-null  float64
 29  NAME_YIELD_GROUP             1670214 non-null  object
 30  PRODUCT_COMBINATION          1669868 non-null  object
 31  DAYS_FIRST_DRAWING           997149 non-null   float64
 32  DAYS_FIRST_DUE               997149 non-null   float64
 33  DAYS_LAST_DUE_1ST_VERSION    997149 non-null   float64
 34  DAYS_LAST_DUE                997149 non-null   float64
 35  DAYS_TERMINATION             997149 non-null   float64
 36  NFLAG_INSURED_ON_APPROVAL    997149 non-null   float64
dtypes: float64(15), int64(6), object(16)
memory usage: 471.5+ MB
None
```

	SK_ID_PREV	SK_ID_CURR	NAME_CONTRACT_TYPE	AMT_ANNUITY	AMT_APPLICATION	AMT_CREDIT	AMT_DOWN_PAYMENT	AMT
0	2030495	271877	Consumer loans	1730.430	17145.0	17145.0	0.0	
1	2802425	108129	Cash loans	25188.615	607500.0	679671.0	NaN	
2	2523466	122040	Cash loans	15060.735	112500.0	136444.5	NaN	
3	2819243	176158	Cash loans	47041.335	450000.0	470790.0	NaN	
4	1784265	202054	Cash loans	31924.395	337500.0	404055.0	NaN	

5 rows × 37 columns

```
POS_CASH_balance: shape is (10001358, 8)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10001358 entries, 0 to 10001357
Data columns (total 8 columns):
 #   Column                 Dtype
---  ------                 -----
 0   SK_ID_PREV             int64
 1   SK_ID_CURR             int64
 2   MONTHS_BALANCE         int64
 3   CNT_INSTALMENT         float64
 4   CNT_INSTALMENT_FUTURE  float64
 5   NAME_CONTRACT_STATUS   object
 6   SK_DPD                 int64
 7   SK_DPD_DEF             int64
dtypes: float64(2), int64(5), object(1)
memory usage: 610.4+ MB
None
```

	SK_ID_PREV	SK_ID_CURR	MONTHS_BALANCE	CNT_INSTALMENT	CNT_INSTALMENT_FUTURE	NAME_CONTRACT_STATUS	SK_DPD
0	1803195	182943	-31	48.0	45.0	Active	0
1	1715348	367990	-33	36.0	35.0	Active	0
2	1784872	397406	-32	12.0	9.0	Active	0
3	1903291	269225	-35	48.0	42.0	Active	0
4	2341044	334279	-35	36.0	35.0	Active	0

```
CPU times: user 38.8 s, sys: 6.55 s, total: 45.3 s
Wall time: 49 s
```

In [3]:

```
appsTrain = datasets['application_train']
df_prevApps = datasets["previous_application"]
df_bureau = datasets["bureau"]
df_bureau_bal = datasets['bureau_balance']
df_cc_bal = datasets["credit_card_balance"]
df_installments_payments = datasets["installments_payments"]
```

In [4]:

```
# pd.set_option('display.float_format', lambda x: '%.5f' % x)
# pd.set_option('mode.chained_assignment', None)
```

```
df = appsTrain.copy()
df_test=datasets['application_test'].copy()
```

In [5]:
```
df_test.shape
```

Out[5]: (48744, 121)

In [6]:
```
#previous applications table

temp_previous_df = df_prevApps.groupby('SK_ID_CURR', as_index=False).agg({'NAME_CONTRACT_STATUS': lambda x: ','
temp_previous_df['has_only_approved'] = np.where(temp_previous_df['NAME_CONTRACT_STATUS'] == 'Approved', '1', '
temp_previous_df['has_been_rejected'] = np.where(temp_previous_df['NAME_CONTRACT_STATUS'].str.contains('Refused
```

In [7]:
```
# JOIN DATA
df = pd.merge(df, temp_previous_df, on='SK_ID_CURR', how='left')
```

In [8]:
```
df_test= pd.merge(df_test, temp_previous_df, on='SK_ID_CURR', how='left')
```

In [9]:
```
df_test.shape
```

Out[9]: (48744, 124)

Feature Engineering

In [10]:
```
# CREATE CUSTOM COLUMNS

#total_amt_req_credit_bureau
df['total_amt_req_credit_bureau'] = (
  df['AMT_REQ_CREDIT_BUREAU_YEAR'] * 1 +
  df['AMT_REQ_CREDIT_BUREAU_QRT'] * 2 +
  df['AMT_REQ_CREDIT_BUREAU_MON'] * 8 +
  df['AMT_REQ_CREDIT_BUREAU_WEEK'] * 16 +
  df['AMT_REQ_CREDIT_BUREAU_DAY'] * 32 +
  df['AMT_REQ_CREDIT_BUREAU_HOUR'] * 64)
df['total_amt_req_credit_bureau_isnull'] = np.where(df['total_amt_req_credit_bureau'].isnull(), '1', '0')
df['total_amt_req_credit_bureau'].fillna(0, inplace=True)
```

```
#has_job
df['has_job'] = np.where(df['NAME_INCOME_TYPE'].isin(['Pensioner', 'Student', 'Unemployed']), '1', '0')

#has_children
df['has_children'] = np.where(df['CNT_CHILDREN'] > 0, '1', '0')
```

In [11]:

```
#total_amt_req_credit_bureau
df_test['total_amt_req_credit_bureau'] = (
  df_test['AMT_REQ_CREDIT_BUREAU_YEAR'] * 1 +
  df_test['AMT_REQ_CREDIT_BUREAU_QRT'] * 2 +
  df_test['AMT_REQ_CREDIT_BUREAU_MON'] * 8 +
  df_test['AMT_REQ_CREDIT_BUREAU_WEEK'] * 16 +
  df_test['AMT_REQ_CREDIT_BUREAU_DAY'] * 32 +
  df_test['AMT_REQ_CREDIT_BUREAU_HOUR'] * 64)
df_test['total_amt_req_credit_bureau_isnull'] = np.where(df_test['total_amt_req_credit_bureau'].isnull(), '1',
df_test['total_amt_req_credit_bureau'].fillna(0, inplace=True)

#has_job
df_test['has_job'] = np.where(df_test['NAME_INCOME_TYPE'].isin(['Pensioner', 'Student', 'Unemployed']), '1', '0

#has_children
df_test['has_children'] = np.where(df_test['CNT_CHILDREN'] > 0, '1', '0')
```

In [12]:

```
# clusterise_days_employed
def clusterise_days_employed(x):
    days = x['DAYS_EMPLOYED']
    if days > 0:
      return 'not available'
    else:
      days = abs(days)
      if days < 30:
        return 'less 1 month'
      elif days < 180:
        return 'less 6 months'
      elif days < 365:
        return 'less 1 year'
      elif days < 1095:
        return 'less 3 years'
      elif days < 1825:
        return 'less 5 years'
      elif days < 3600:
```

```
        return 'less 10 years'
      elif days < 7200:
        return 'less 20 years'
      elif days >= 7200:
        return 'more 20 years'
      else:
        return 'not available'

df['cluster_days_employed'] = df.apply(clusterise_days_employed, axis=1)
```

In [13]:

```
df_test['cluster_days_employed'] = df_test.apply(clusterise_days_employed, axis=1)
```

In [14]:

```
#custom_ext_source_3
def clusterise_ext_source(x):
    if str(x) == 'nan':
      return 'not available'
    else:
      if x < 0.1:
        return 'less 0.1'
      elif x < 0.2:
        return 'less 0.2'
      elif x < 0.3:
        return 'less 0.3'
      elif x < 0.4:
        return 'less 0.4'
      elif x < 0.5:
        return 'less 0.5'
      elif x < 0.6:
        return 'less 0.6'
      elif x < 0.7:
        return 'less 0.7'
      elif x < 0.8:
        return 'less 0.8'
      elif x < 0.9:
        return 'less 0.9'
      elif x <= 1:
        return 'less 1'
df['clusterise_ext_source_1'] = df['EXT_SOURCE_1'].apply(lambda x: clusterise_ext_source(x))
df['clusterise_ext_source_2'] = df['EXT_SOURCE_2'].apply(lambda x: clusterise_ext_source(x))
df['clusterise_ext_source_3'] = df['EXT_SOURCE_3'].apply(lambda x: clusterise_ext_source(x))
```

In [15]:

```
df_test['clusterise_ext_source_1'] = df_test['EXT_SOURCE_1'].apply(lambda x: clusterise_ext_source(x))
df_test['clusterise_ext_source_2'] = df_test['EXT_SOURCE_2'].apply(lambda x: clusterise_ext_source(x))
df_test['clusterise_ext_source_3'] = df_test['EXT_SOURCE_3'].apply(lambda x: clusterise_ext_source(x))
```

In [16]:
```
# house_variables_sum
house_vars = ['APARTMENTS_AVG','APARTMENTS_MEDI','APARTMENTS_MODE','BASEMENTAREA_AVG',
  'BASEMENTAREA_MEDI','BASEMENTAREA_MODE','COMMONAREA_AVG','COMMONAREA_MEDI',
  'COMMONAREA_MODE','ELEVATORS_AVG','ELEVATORS_MEDI','ELEVATORS_MODE','EMERGENCYSTATE_MODE',
  'ENTRANCES_AVG','ENTRANCES_MEDI','ENTRANCES_MODE','FLOORSMAX_AVG','FLOORSMAX_MEDI',
  'FLOORSMAX_MODE','FLOORSMIN_AVG','FLOORSMIN_MEDI','FLOORSMIN_MODE','FONDKAPREMONT_MODE',
  'HOUSETYPE_MODE','LANDAREA_AVG','LANDAREA_MEDI','LANDAREA_MODE','LIVINGAPARTMENTS_AVG',
  'LIVINGAPARTMENTS_MEDI','LIVINGAPARTMENTS_MODE','LIVINGAREA_AVG','LIVINGAREA_MEDI','LIVINGAREA_MODE',
  'NONLIVINGAPARTMENTS_AVG','NONLIVINGAPARTMENTS_MEDI','NONLIVINGAPARTMENTS_MODE','NONLIVINGAREA_AVG',
  'NONLIVINGAREA_MEDI','NONLIVINGAREA_MODE','TOTALAREA_MODE','WALLSMATERIAL_MODE',
  'YEARS_BEGINEXPLUATATION_AVG','YEARS_BEGINEXPLUATATION_MEDI','YEARS_BEGINEXPLUATATION_MODE',
  'YEARS_BUILD_AVG','YEARS_BUILD_MEDI','YEARS_BUILD_MODE']
df['house_variables_sum'] = df[house_vars].sum(axis=1)
df['house_variables_sum_isnull'] = np.where(df['house_variables_sum'].isnull(), '1', '0')
df['house_variables_sum'].fillna(value=df['house_variables_sum'].median(), inplace=True)
```

In [17]:
```
df_test['house_variables_sum'] = df_test[house_vars].sum(axis=1)
df_test['house_variables_sum_isnull'] = np.where(df_test['house_variables_sum'].isnull(), '1', '0')
df_test['house_variables_sum'].fillna(value=df_test['house_variables_sum'].median(), inplace=True)
```

In [18]:
```
df_test.shape
```

Out[18]:
```
(48744, 134)
```

In [19]:
```
categorical = []
for column in df.columns:
    if df[column].dtype == 'object':
        categorical.append(column)

categorical, len(categorical)
```

Out[19]:
```
(['NAME_CONTRACT_TYPE',
  'CODE_GENDER',
  'FLAG_OWN_CAR',
  'FLAG_OWN_REALTY',
  'NAME_TYPE_SUITE',
```

```
 'NAME_INCOME_TYPE',
 'NAME_EDUCATION_TYPE',
 'NAME_FAMILY_STATUS',
 'NAME_HOUSING_TYPE',
 'OCCUPATION_TYPE',
 'WEEKDAY_APPR_PROCESS_START',
 'ORGANIZATION_TYPE',
 'FONDKAPREMONT_MODE',
 'HOUSETYPE_MODE',
 'WALLSMATERIAL_MODE',
 'EMERGENCYSTATE_MODE',
 'NAME_CONTRACT_STATUS',
 'has_only_approved',
 'has_been_rejected',
 'total_amt_req_credit_bureau_isnull',
 'has_job',
 'has_children',
 'cluster_days_employed',
 'clusterise_ext_source_1',
 'clusterise_ext_source_2',
 'clusterise_ext_source_3',
 'house_variables_sum_isnull'],
27)
```

In [20]:
```
df_test.shape
```

Out[20]:
```
(48744, 134)
```

In [21]:
```
categorical_test = []
for col in df_test.columns:
    if df_test[col].dtype == 'object':
        categorical_test.append(col)

categorical_test, len(categorical_test)
```

Out[21]:
```
(['NAME_CONTRACT_TYPE',
  'CODE_GENDER',
  'FLAG_OWN_CAR',
  'FLAG_OWN_REALTY',
  'NAME_TYPE_SUITE',
  'NAME_INCOME_TYPE',
  'NAME_EDUCATION_TYPE',
  'NAME_FAMILY_STATUS',
  'NAME_HOUSING_TYPE',
  'OCCUPATION_TYPE',
```

```
 'WEEKDAY_APPR_PROCESS_START',
 'ORGANIZATION_TYPE',
 'FONDKAPREMONT_MODE',
 'HOUSETYPE_MODE',
 'WALLSMATERIAL_MODE',
 'EMERGENCYSTATE_MODE',
 'NAME_CONTRACT_STATUS',
 'has_only_approved',
 'has_been_rejected',
 'total_amt_req_credit_bureau_isnull',
 'has_job',
 'has_children',
 'cluster_days_employed',
 'clusterise_ext_source_1',
 'clusterise_ext_source_2',
 'clusterise_ext_source_3',
 'house_variables_sum_isnull'],
27)
```

In [22]:

```
numerical = []
for column in df.columns:
    if not df[column].dtype == 'object':
        numerical.append(column)

numerical, len(numerical)
```

Out[22]:

```
(['SK_ID_CURR',
  'TARGET',
  'CNT_CHILDREN',
  'AMT_INCOME_TOTAL',
  'AMT_CREDIT',
  'AMT_ANNUITY',
  'AMT_GOODS_PRICE',
  'REGION_POPULATION_RELATIVE',
  'DAYS_BIRTH',
  'DAYS_EMPLOYED',
  'DAYS_REGISTRATION',
  'DAYS_ID_PUBLISH',
  'OWN_CAR_AGE',
  'FLAG_MOBIL',
  'FLAG_EMP_PHONE',
  'FLAG_WORK_PHONE',
  'FLAG_CONT_MOBILE',
  'FLAG_PHONE',
  'FLAG_EMAIL',
  'CNT_FAM_MEMBERS',
  'REGION_RATING_CLIENT',
```

```
'REGION_RATING_CLIENT_W_CITY',
'HOUR_APPR_PROCESS_START',
'REG_REGION_NOT_LIVE_REGION',
'REG_REGION_NOT_WORK_REGION',
'LIVE_REGION_NOT_WORK_REGION',
'REG_CITY_NOT_LIVE_CITY',
'REG_CITY_NOT_WORK_CITY',
'LIVE_CITY_NOT_WORK_CITY',
'EXT_SOURCE_1',
'EXT_SOURCE_2',
'EXT_SOURCE_3',
'APARTMENTS_AVG',
'BASEMENTAREA_AVG',
'YEARS_BEGINEXPLUATATION_AVG',
'YEARS_BUILD_AVG',
'COMMONAREA_AVG',
'ELEVATORS_AVG',
'ENTRANCES_AVG',
'FLOORSMAX_AVG',
'FLOORSMIN_AVG',
'LANDAREA_AVG',
'LIVINGAPARTMENTS_AVG',
'LIVINGAREA_AVG',
'NONLIVINGAPARTMENTS_AVG',
'NONLIVINGAREA_AVG',
'APARTMENTS_MODE',
'BASEMENTAREA_MODE',
'YEARS_BEGINEXPLUATATION_MODE',
'YEARS_BUILD_MODE',
'COMMONAREA_MODE',
'ELEVATORS_MODE',
'ENTRANCES_MODE',
'FLOORSMAX_MODE',
'FLOORSMIN_MODE',
'LANDAREA_MODE',
'LIVINGAPARTMENTS_MODE',
'LIVINGAREA_MODE',
'NONLIVINGAPARTMENTS_MODE',
'NONLIVINGAREA_MODE',
'APARTMENTS_MEDI',
'BASEMENTAREA_MEDI',
'YEARS_BEGINEXPLUATATION_MEDI',
'YEARS_BUILD_MEDI',
'COMMONAREA_MEDI',
'ELEVATORS_MEDI',
'ENTRANCES_MEDI',
'FLOORSMAX_MEDI',
'FLOORSMIN_MEDI',
'LANDAREA_MEDI',
```

```
 'LIVINGAPARTMENTS_MEDI',
 'LIVINGAREA_MEDI',
 'NONLIVINGAPARTMENTS_MEDI',
 'NONLIVINGAREA_MEDI',
 'TOTALAREA_MODE',
 'OBS_30_CNT_SOCIAL_CIRCLE',
 'DEF_30_CNT_SOCIAL_CIRCLE',
 'OBS_60_CNT_SOCIAL_CIRCLE',
 'DEF_60_CNT_SOCIAL_CIRCLE',
 'DAYS_LAST_PHONE_CHANGE',
 'FLAG_DOCUMENT_2',
 'FLAG_DOCUMENT_3',
 'FLAG_DOCUMENT_4',
 'FLAG_DOCUMENT_5',
 'FLAG_DOCUMENT_6',
 'FLAG_DOCUMENT_7',
 'FLAG_DOCUMENT_8',
 'FLAG_DOCUMENT_9',
 'FLAG_DOCUMENT_10',
 'FLAG_DOCUMENT_11',
 'FLAG_DOCUMENT_12',
 'FLAG_DOCUMENT_13',
 'FLAG_DOCUMENT_14',
 'FLAG_DOCUMENT_15',
 'FLAG_DOCUMENT_16',
 'FLAG_DOCUMENT_17',
 'FLAG_DOCUMENT_18',
 'FLAG_DOCUMENT_19',
 'FLAG_DOCUMENT_20',
 'FLAG_DOCUMENT_21',
 'AMT_REQ_CREDIT_BUREAU_HOUR',
 'AMT_REQ_CREDIT_BUREAU_DAY',
 'AMT_REQ_CREDIT_BUREAU_WEEK',
 'AMT_REQ_CREDIT_BUREAU_MON',
 'AMT_REQ_CREDIT_BUREAU_QRT',
 'AMT_REQ_CREDIT_BUREAU_YEAR',
 'total_amt_req_credit_bureau',
 'house_variables_sum'],
108)
```

In [23]:

```
numerical_test = []
for cl in df_test.columns:
    if not df_test[cl].dtype == 'object':
        numerical_test.append(cl)

numerical_test, len(numerical_test)
```

Out[23]:
```
(['SK_ID_CURR',
  'CNT_CHILDREN',
  'AMT_INCOME_TOTAL',
  'AMT_CREDIT',
  'AMT_ANNUITY',
  'AMT_GOODS_PRICE',
  'REGION_POPULATION_RELATIVE',
  'DAYS_BIRTH',
  'DAYS_EMPLOYED',
  'DAYS_REGISTRATION',
  'DAYS_ID_PUBLISH',
  'OWN_CAR_AGE',
  'FLAG_MOBIL',
  'FLAG_EMP_PHONE',
  'FLAG_WORK_PHONE',
  'FLAG_CONT_MOBILE',
  'FLAG_PHONE',
  'FLAG_EMAIL',
  'CNT_FAM_MEMBERS',
  'REGION_RATING_CLIENT',
  'REGION_RATING_CLIENT_W_CITY',
  'HOUR_APPR_PROCESS_START',
  'REG_REGION_NOT_LIVE_REGION',
  'REG_REGION_NOT_WORK_REGION',
  'LIVE_REGION_NOT_WORK_REGION',
  'REG_CITY_NOT_LIVE_CITY',
  'REG_CITY_NOT_WORK_CITY',
  'LIVE_CITY_NOT_WORK_CITY',
  'EXT_SOURCE_1',
  'EXT_SOURCE_2',
  'EXT_SOURCE_3',
  'APARTMENTS_AVG',
  'BASEMENTAREA_AVG',
  'YEARS_BEGINEXPLUATATION_AVG',
  'YEARS_BUILD_AVG',
  'COMMONAREA_AVG',
  'ELEVATORS_AVG',
  'ENTRANCES_AVG',
  'FLOORSMAX_AVG',
  'FLOORSMIN_AVG',
  'LANDAREA_AVG',
  'LIVINGAPARTMENTS_AVG',
  'LIVINGAREA_AVG',
  'NONLIVINGAPARTMENTS_AVG',
  'NONLIVINGAREA_AVG',
  'APARTMENTS_MODE',
  'BASEMENTAREA_MODE',
  'YEARS_BEGINEXPLUATATION_MODE',
  'YEARS_BUILD_MODE',
```

```
'COMMONAREA_MODE',
'ELEVATORS_MODE',
'ENTRANCES_MODE',
'FLOORSMAX_MODE',
'FLOORSMIN_MODE',
'LANDAREA_MODE',
'LIVINGAPARTMENTS_MODE',
'LIVINGAREA_MODE',
'NONLIVINGAPARTMENTS_MODE',
'NONLIVINGAREA_MODE',
'APARTMENTS_MEDI',
'BASEMENTAREA_MEDI',
'YEARS_BEGINEXPLUATATION_MEDI',
'YEARS_BUILD_MEDI',
'COMMONAREA_MEDI',
'ELEVATORS_MEDI',
'ENTRANCES_MEDI',
'FLOORSMAX_MEDI',
'FLOORSMIN_MEDI',
'LANDAREA_MEDI',
'LIVINGAPARTMENTS_MEDI',
'LIVINGAREA_MEDI',
'NONLIVINGAPARTMENTS_MEDI',
'NONLIVINGAREA_MEDI',
'TOTALAREA_MODE',
'OBS_30_CNT_SOCIAL_CIRCLE',
'DEF_30_CNT_SOCIAL_CIRCLE',
'OBS_60_CNT_SOCIAL_CIRCLE',
'DEF_60_CNT_SOCIAL_CIRCLE',
'DAYS_LAST_PHONE_CHANGE',
'FLAG_DOCUMENT_2',
'FLAG_DOCUMENT_3',
'FLAG_DOCUMENT_4',
'FLAG_DOCUMENT_5',
'FLAG_DOCUMENT_6',
'FLAG_DOCUMENT_7',
'FLAG_DOCUMENT_8',
'FLAG_DOCUMENT_9',
'FLAG_DOCUMENT_10',
'FLAG_DOCUMENT_11',
'FLAG_DOCUMENT_12',
'FLAG_DOCUMENT_13',
'FLAG_DOCUMENT_14',
'FLAG_DOCUMENT_15',
'FLAG_DOCUMENT_16',
'FLAG_DOCUMENT_17',
'FLAG_DOCUMENT_18',
'FLAG_DOCUMENT_19',
'FLAG_DOCUMENT_20',
```

```
 'FLAG_DOCUMENT_21',
 'AMT_REQ_CREDIT_BUREAU_HOUR',
 'AMT_REQ_CREDIT_BUREAU_DAY',
 'AMT_REQ_CREDIT_BUREAU_WEEK',
 'AMT_REQ_CREDIT_BUREAU_MON',
 'AMT_REQ_CREDIT_BUREAU_QRT',
 'AMT_REQ_CREDIT_BUREAU_YEAR',
 'total_amt_req_credit_bureau',
 'house_variables_sum'],
107)
```

In [24]:
```
# MANAGE MISSING VALUES
for numerical_column in numerical:
    if df[numerical_column].isnull().values.any():
        df[numerical_column + '_isnull'] = np.where(df[numerical_column].isnull(), '1', '0')
        df[numerical_column].fillna(value=df[numerical_column].median(), inplace=True)

for categorical_column in categorical:
    df[categorical_column].fillna('NULL', inplace=True)
```

In [25]:
```
df.shape
```

Out[25]:
```
(307511, 196)
```

In [26]:
```
# MANAGE MISSING VALUES
for numerical_column in numerical_test:
    if df_test[numerical_column].isnull().values.any():
        df_test[numerical_column + '_isnull'] = np.where(df_test[numerical_column].isnull(), '1', '0')
        df_test[numerical_column].fillna(value=df_test[numerical_column].median(), inplace=True)

for categorical_column in categorical_test:
    df_test[categorical_column].fillna('NULL', inplace=True)
```

In [27]:
```
# STANDARDISE
min_max_scaler = preprocessing.MinMaxScaler()
df[numerical] = pd.DataFrame(min_max_scaler.fit_transform(df[numerical]))

for column in categorical:
    df[column] = LabelEncoder().fit_transform(df[column].astype(str))
    df[column] = df[column].astype('category')
```

In [28]:
```
df.shape
```

Out[28]:
```
(307511, 196)
```

In [29]:
```
# STANDARDISE
min_max_scaler = preprocessing.MinMaxScaler()
df_test[numerical_test] = pd.DataFrame(min_max_scaler.fit_transform(df_test[numerical_test]))

for column in categorical_test:
    df_test[column] = LabelEncoder().fit_transform(df_test[column].astype(str))
    df_test[column] = df_test[column].astype('category')
```

In [30]:
```
dfcolumns= []
for column in df.columns:
    dfcolumns.append(column)

len(dfcolumns)
```

Out[30]:
```
196
```

In [31]:
```
df_test.shape
```

Out[31]:
```
(48744, 192)
```

Modelling Pipelines (MLP)

In [32]:
```
# SPLIT DATA INTO TRAINING vs Validation

train_df =df
train_output_df = pd.DataFrame(train_df['TARGET'], columns=['TARGET'])

# test_df = datasets['application_test']

# REMOVE NOT USEFUL COLUMNS
train_df.drop(columns=['TARGET'], axis=0, inplace=True)
#test_df.drop(columns=['CSV_SOURCE', 'TARGET'], axis=0, inplace=True)

# CREATE VALIDATION TEST
x_train, x_validation, y_train, y_validation = train_test_split(train_df, train_output_df, test_size=0.2, randc
```

In [33]:
```
x_train.shape
```

Out[33]:
```
(246008, 195)
```

In [34]:
```
# CREATE TENSORS
def create_tensors(input_df):
    stack = []
    for column in input_df.columns:
        if input_df.dtypes[column] == np.int64 or input_df.dtypes[column] == np.float64:
            stack.append(input_df[column].astype(np.float64))
        else:
            stack.append(input_df[column].cat.codes.values)
    return torch.tensor(np.stack(stack, 1), dtype=torch.float)
```

In [35]:
```
numerical.remove('TARGET')
```

In [36]:
```
tensor_x_train_cat = create_tensors(x_train[categorical]).float()
tensor_x_train_num = create_tensors(x_train[numerical]).float()
tensor_y_train = torch.tensor(y_train.values).flatten().float()

tensor_x_valid_cat = create_tensors(x_validation[categorical]).float()
tensor_x_valid_num = create_tensors(x_validation[numerical]).float()
tensor_y_valid = torch.tensor(y_validation.values).flatten().float()
```

In [37]:
```
tensor_y_train.shape
```

Out[37]:
```
torch.Size([246008])
```

In [38]:
```
# CREATE CATEGORICAL EMBEDDING SIZES
categorical_columns_size = [len(df[column].cat.categories) for column in categorical]
categorical_embedding_sizes = [(col_size, min(50, (col_size + 1) // 2)) for col_size in categorical_columns_siz
```

In [39]:
```
categorical_embedding_sizes
```

Out[39]:
```
[(2, 1),
```

```
(3, 2),
(2, 1),
(2, 1),
(8, 4),
(8, 4),
(5, 3),
(6, 3),
(6, 3),
(19, 10),
(7, 4),
(58, 29),
(5, 3),
(4, 2),
(8, 4),
(3, 2),
(20, 10),
(3, 2),
(3, 2),
(2, 1),
(2, 1),
(2, 1),
(9, 5),
(11, 6),
(10, 5),
(10, 5),
(1, 1)]
```

In [40]:

```
# DEFINE NEURAL NETWORK MODEL
class Model(nn.Module):
    def __init__(self, embedding_size, input_size, num_numerical_cols, layers):
        super().__init__()

        self.all_embeddings = nn.ModuleList([nn.Embedding(ni, nf) for ni, nf in embedding_size])
        #self.emb_drop = nn.Dropout(hp_emb_drop)

        self.bn_cont = nn.BatchNorm1d(num_numerical_cols)

        layerlist = []
        for i, elem in enumerate(layers):
            layerlist.append(nn.Linear(input_size, elem))
            layerlist.append(nn.ReLU(inplace=True))
            layerlist.append(nn.BatchNorm1d(layers[i]))
#             layerlist.append(nn.Dropout(ps[i]))
            input_size = elem
        layerlist.append(nn.Linear(layers[-1], 1))

        self.layers = nn.Sequential(*layerlist)
```

```
    def forward(self, x_c, x_n):

        embeddings = [e(x_c[:,i].long()) for i, e in enumerate(self.all_embeddings)]

        x = torch.cat(embeddings, 1)
#         x = self.emb_drop(x)

        x_n = self.bn_cont(x_n)

        x = torch.cat([x, x_n], 1)
        x = self.layers(x)

        return x
```

In [41]:

```
# SET HYPERPARAMETERS
hr_batch_size = 100
hp_lr= 0.00001
hp_layers = [512, 256, 128]


# INSTANCIATE MODEL
num_numerical_cols = tensor_x_train_num.shape[1]
num_categorical_cols = sum((nf for ni, nf in categorical_embedding_sizes))

initial_input_size = num_categorical_cols + num_numerical_cols

mlp_model = Model(categorical_embedding_sizes, initial_input_size, num_numerical_cols, layers=hp_layers)
sigmoid = nn.Sigmoid()
loss_function = nn.BCELoss()
optimizer = torch.optim.Adam(mlp_model.parameters(), lr=hp_lr)
```

In [42]:

```
# TRAIN NEURAL NETWORK MODEL
train_tensor_dataset = TensorDataset(tensor_x_train_cat, tensor_x_train_num, tensor_y_train)
train_loader = DataLoader(dataset=train_tensor_dataset, batch_size=hr_batch_size, shuffle=True)
from torch.utils.tensorboard import SummaryWriter
writer = SummaryWriter()

mlp_model.train()

tot_y_train_in = []
tot_y_train_out = []
```

```
for epoch in range(50):
    train_losses = []
    for x_cat, x_num, y in train_loader:
        y_train = mlp_model(x_cat, x_num)
        single_loss = loss_function(sigmoid(y_train.squeeze()), y)
        writer.add_scalar("Loss/train", single_loss, epoch)
        single_loss.backward()
        optimizer.step()
        train_losses.append(single_loss.item())
        tot_y_train_in.append(y)
        tot_y_train_out.append(y_train)

    epoch_loss = 1.0 * sum(train_losses) / len(train_losses)
    epoch_auc = roc_auc_score(torch.cat(tot_y_train_in).cpu().numpy(), torch.cat(tot_y_train_out).cpu().detach(
    tot_y_train_in = []
    tot_y_train_out = []
    print("\tepoch: " + str(epoch) + "\tloss: " + str(epoch_loss) + "\tauc: " + str(epoch_auc))
```

```
epoch: 0        loss: 0.6118271087350423        auc: 0.6838375470903345
epoch: 1        loss: 0.46276422931817623       auc: 0.703885933015179
epoch: 2        loss: 0.32905352884606104       auc: 0.720951730923828
epoch: 3        loss: 0.2611776528889839        auc: 0.7361607664546705
epoch: 4        loss: 0.2484955261697957        auc: 0.7529911482892232
epoch: 5        loss: 0.261940286809269 auc: 0.7631637097504218
epoch: 6        loss: 0.279778764947601 auc: 0.7674183261936423
epoch: 7        loss: 0.29236252351911124       auc: 0.7689226916663442
epoch: 8        loss: 0.3002688138814221        auc: 0.7697884841325506
epoch: 9        loss: 0.3077548192916169        auc: 0.7709040875033828
epoch: 10       loss: 0.31460094327627036       auc: 0.7670689634853776
epoch: 11       loss: 0.32051867795582106       auc: 0.7672849851735136
epoch: 12       loss: 0.3253876885103288        auc: 0.7655776542367678
epoch: 13       loss: 0.32768817336178563       auc: 0.7620763420424719
epoch: 14       loss: 0.3254136730111115        auc: 0.7623972756334707
epoch: 15       loss: 0.31755435069563526       auc: 0.7632035183017165
epoch: 16       loss: 0.31603116205447285       auc: 0.7592305538639353
epoch: 17       loss: 0.31199726231520325       auc: 0.7571370411441354
epoch: 18       loss: 0.3076841333401925        auc: 0.7585336728831023
epoch: 19       loss: 0.30378089700165145       auc: 0.7551387854071716
epoch: 20       loss: 0.30657135021003556       auc: 0.7486862509492591
epoch: 21       loss: 0.29963759849077315       auc: 0.7522072852561476
epoch: 22       loss: 0.29308713812415843       auc: 0.7498738531690424
epoch: 23       loss: 0.28803657609870986       auc: 0.7487915664475808
epoch: 24       loss: 0.281337328106831 auc: 0.7479946549775359
epoch: 25       loss: 0.2727034668983458        auc: 0.751920423220468
epoch: 26       loss: 0.262994976200342 auc: 0.7592900124413129
epoch: 27       loss: 0.2553618251717294        auc: 0.7682260841247717
```

```
epoch: 28       loss: 0.25300275810306594       auc: 0.7702755982649001
epoch: 29       loss: 0.2520141741839888        auc: 0.7669883484073583
epoch: 30       loss: 0.2492190497287438        auc: 0.7722051769170795
epoch: 31       loss: 0.24937147708978294       auc: 0.7731544610245276
epoch: 32       loss: 0.24961339293463305       auc: 0.7736076127325238
epoch: 33       loss: 0.24752140582540277       auc: 0.7725632519697585
epoch: 34       loss: 0.24423601794707878       auc: 0.774151299534481
epoch: 35       loss: 0.24346797769508532       auc: 0.7733756847332915
epoch: 36       loss: 0.241859458084152 auc: 0.7743337206769922
epoch: 37       loss: 0.24184881460431726       auc: 0.7738946212773596
epoch: 38       loss: 0.2423181417892009        auc: 0.7724121989159634
epoch: 39       loss: 0.24228060804685894       auc: 0.7721750704535277
epoch: 40       loss: 0.2419439029737199        auc: 0.7726538999770025
epoch: 41       loss: 0.24203267454955696       auc: 0.7723854300133134
epoch: 42       loss: 0.24235437883080044       auc: 0.7719661539588716
epoch: 43       loss: 0.2419929914131052        auc: 0.7730543784728365
epoch: 44       loss: 0.24227278379587952       auc: 0.771364078530232
epoch: 45       loss: 0.24240743479366372       auc: 0.7711093219125685
epoch: 46       loss: 0.24222775964669607       auc: 0.771762577873143
epoch: 47       loss: 0.242320222805085 auc: 0.771383677495037
epoch: 48       loss: 0.24252618560457753       auc: 0.7715066045731389
epoch: 49       loss: 0.2425613955952975        auc: 0.770503733601516
```

In [53]:

```
writer.flush()
writer.close()
```

In [ ]:

Tensorboard Visualization of the Train loss vs Epoch

In [1]:

```
%load_ext tensorboard
%tensorboard --logdir runs
```

Index of /

Name	Size	Date Modified
.vol/		3/26/22, 3:21:13 AM
Applications/		4/27/22, 1:04:06 AM
bin/		3/26/22, 3:21:13 AM
cores/		4/24/21, 11:39:11 PM

cores/		4/24/21, 11:59:11 PM
dev/		4/26/22, 11:36:47 PM
etc/		4/26/22, 11:36:32 PM
home/		4/26/22, 11:36:57 PM
Library/		4/13/22, 12:01:16 AM
opt/		12/3/21, 9:11:01 AM
private/		4/26/22, 11:36:53 PM
sbin/		3/26/22, 3:21:13 AM
System/		3/26/22, 3:21:13 AM
tmp/		4/30/22, 9:07:55 PM
Users/		3/26/22, 3:21:13 AM
usr/		3/26/22, 3:21:13 AM
var/		4/26/22, 11:36:53 PM
Volumes/		4/26/22, 11:36:54 PM
.file	0 B	3/26/22, 3:21:13 AM
.VolumeIcon.icns	0 B	

In [45]:
```
# VALIDATE NEURAL NETWORK MODEL

validation_tensor_dataset = TensorDataset(tensor_x_valid_cat, tensor_x_valid_num, tensor_y_valid)
validation_loader = DataLoader(dataset=validation_tensor_dataset, batch_size=hr_batch_size, shuffle=True)
```

```
valid_losses = []

mlp_model.eval()

tot_y_valid_in = []
tot_y_valid_out = []

with torch.no_grad():
    for x_cat, x_num, y in validation_loader:
        y_valid = mlp_model(x_cat, x_num)
        validation_loss = loss_function(sigmoid(y_valid.squeeze()), y)
        valid_losses.append(validation_loss.item())

        tot_y_valid_in.append(y_valid)
        tot_y_valid_out.append(y)

    valid_loss = round(1.0 * sum(valid_losses) / len(valid_losses), 5)
    print("\tloss: " + str(valid_loss))
    valid_auc = roc_auc_score(torch.cat(tot_y_valid_out).cpu(), torch.cat(tot_y_valid_in).cpu())
    print("\tauc: " + str(valid_auc))
```

```
	loss: 0.45607
	auc: 0.7368927015668744
```

In [46]:

```
mlp_model
```

Out[46]:

```
Model(
  (all_embeddings): ModuleList(
    (0): Embedding(2, 1)
    (1): Embedding(3, 2)
    (2): Embedding(2, 1)
    (3): Embedding(2, 1)
    (4): Embedding(8, 4)
    (5): Embedding(8, 4)
    (6): Embedding(5, 3)
    (7): Embedding(6, 3)
    (8): Embedding(6, 3)
    (9): Embedding(19, 10)
    (10): Embedding(7, 4)
    (11): Embedding(58, 29)
    (12): Embedding(5, 3)
    (13): Embedding(4, 2)
    (14): Embedding(8, 4)
    (15): Embedding(3, 2)
    (16): Embedding(20, 10)
    (17): Embedding(3, 2)
```

```
    (18): Embedding(3, 2)
    (19): Embedding(2, 1)
    (20): Embedding(2, 1)
    (21): Embedding(2, 1)
    (22): Embedding(9, 5)
    (23): Embedding(11, 6)
    (24): Embedding(10, 5)
    (25): Embedding(10, 5)
    (26): Embedding(1, 1)
  )
  (bn_cont): BatchNorm1d(107, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (layers): Sequential(
    (0): Linear(in_features=222, out_features=512, bias=True)
    (1): ReLU(inplace=True)
    (2): BatchNorm1d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (3): Linear(in_features=512, out_features=256, bias=True)
    (4): ReLU(inplace=True)
    (5): BatchNorm1d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (6): Linear(in_features=256, out_features=128, bias=True)
    (7): ReLU(inplace=True)
    (8): BatchNorm1d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (9): Linear(in_features=128, out_features=1, bias=True)
  )
)
```

In [47]:

```
test_df=df_test
data =test_df[categorical]
np.array(data).shape
```

Out[47]:

```
(48744, 27)
```

In [48]:

```
test_df.shape
```

Out[48]:

```
(48744, 192)
```

In [49]:

```
tensor_x_test_cat = create_tensors(test_df[categorical].astype('category')).float()
tensor_x_test_num = create_tensors(test_df[numerical]).float()
```

In [50]:

```
# MAKE PREDICTIONS
import torch.nn.functional as F
print("MAKING PREDICTIONS...")
preds=[]
```

```
with torch.no_grad():
    y_test = mlp_model(tensor_x_test_cat, tensor_x_test_num)
    prob = F.softmax(y_test, dim=1)
    preds.append(prob)

preds
```

```
MAKING PREDICTIONS...
```

Out[50]:

```
[tensor([[1.],
         [1.],
         [1.],
         ...,
         [1.],
         [1.],
         [1.]])]
```

In [51]:

```
# GENERATE SUBMISSION.csv
print("GENERATING SUBMISSIONS...")
nn_prediction_df = pd.DataFrame(y_test).astype("float")
x_scaled = min_max_scaler.fit_transform(nn_prediction_df)
nn_prediction_df = pd.DataFrame(x_scaled)
nn_prediction_df = pd.concat([nn_prediction_df, datasets['application_test']['SK_ID_CURR']], axis=1)
nn_prediction_df.columns = ['TEMP_TARGET', 'SK_ID_CURR']
nn_prediction_df['TARGET'] = nn_prediction_df['TEMP_TARGET']
nn_prediction_df = nn_prediction_df[['SK_ID_CURR', 'TARGET']]
nn_prediction_df.to_csv('./submission5.csv', index=False)

print("EXECUTION COMPLETED.")
```

```
GENERATING SUBMISSIONS...
EXECUTION COMPLETED.
```

In [52]:

```
nn_prediction_df
```

Out[52]:

	SK_ID_CURR	TARGET
0	100001	0.014726
1	100005	0.014727
2	100013	0.014637
3	100028	0.014725
4	100038	0.014726

	SK_ID_CURR	TARGET
...	...	...
48739	456221	0.014726
48740	456222	0.014726
48741	456223	0.009636
48742	456224	0.014725
48743	456250	0.014682

48744 rows × 2 columns

In [ ]:

Write-up

For this phase of the project, you will need to submit a write-up summarizing the work you did.

Home Credit Default risk prediction Phase-3 Final Report By Group 21.

Team Profile & Project Metadata Information:

- Sri Veera Sesha Sai Bhavik Kollipara (srikoll@iu.edu)
- Aditya Shekhar Camarushy (adcama@iu.edu)
- Pradeep Reddy Rokkam (prokkam@iu.edu)
- Sai Prajwal Reddy (reddysai@iu.edu)

Team Member Responsibilities:

All the members of the team were involved in brainstorming, discussions and development of strategies for the final stretch of the project.

Bhavik worked on building the Multilayer Perceptron with multiple hidden layers.

Aditya worked on engineering nine new features for from the existing dataset by combining existing fields with fields from other tables.

Prajwal worked on visualization of Accuracy, loss, AUC and ROC using Tensorboard.

Pradeep worked on Hyperparameter tuning of parameters such as the batch size, activation functions, epochs, the usage of dropout layer along with Batch Normalization.

Abstract

Our objective with the Home Credit Default Risk Prediction project is to predict whether or not a client will repay a loan with high accuracy using alternative data sources such as telco, transactional data, payment behavior.etc. This phase of the project focuses on implementing a neural network model with Binary CXE Loss, evaluating for data and memory leakage, created Train-test splits for the dataset. We also evaluated the model using ROC, AUC metric and finally uploaded the predictions to Kaggle to obtain the final scores. We have implemented MLP with three hidden layers, the activation function used was ReLU (after each layer) ,the number of

neurons we used layerwise are 512,256,128 followed batch normlaization and a dropout layer. We found that the best performing model had the following parameters, batch size = 100, learning rate = 0.00001, layer = [512, 256, 128] and epochs = 50. The model evaluation metrics we received are as follows - We also visualized the model evaluation metrics (ROC, AUC .etc) using Tensorboard post which we uploaded our predictions to Kaggle. The final dataset we fed to MLP consisted of 192 features that we obtained after performing feature engineerning.The MLP model gave an test ROC-AUC score of 0.726.We submitted scores of the Multi-Layer Perceptron model on Kaggle to get a public score of 0.72408 and private score of 0.72666.

Data and Task Description

Source: Home Credit Default Risk | Kaggle : https://www.kaggle.com/competitions/home-credit-default-risk/data

The dataset consists of 9 relevant files present in '.csv' format. Here is a brief overview of the data -

application_{Train|Test}.csv (Primary Key - SK_ID_CURR) : This is the main file used for training and testing purposes, each record corresponds to a loan application made by an individual (at Home Credit) , it consists of 122 columns including data contract type, credit amount, annual income, number of children, age, occupation.etc.

bureau.csv (Primary Key - SK_ID_CURR, SK_ID_BUREAU) : This file consists of application data from previous loans that the individual has applied for in other financial institutions (and these were reported to the credit bureau). There could be multiple entries for individuals depending on the number of applications they made to other financial institutions.

bureau_balance.csv (Primary Key - SK_ID_BUREAU, MONTHS_BALANCE) : This file is a monthly breakdown of the loan balance status for a given individual's specific loan from the 'bureau.csv' data above. An individual can have multiple records for a given loan (corresponding to each month) and multiple loans (which again correspond to multiple records at the monthly level).

previous_application.csv (Primary Key: SK_ID_PREV, Foreign Key: SK_ID_CURR) : Here we find the previous application data of individuals who applied for a loan at Home Credit and the data is similar to application(train/test) data albeit with reduced columns. Each record corresponds to an individual's previously filed loan application.

installments_payments.csv (Primary Key: SK_ID_PREV, Foreign Key: SK_ID_CURR) : This file consists of the payment data for installments of approved loans from the previous application data seen above. Here we can see the number of installments, installment amounts.etc.

credit_card_balance.csv (Primary Key: SK_ID_PREV, SK_ID_CURR,MONTHS_BALANCE) : Here we can observe the balance amount of each individual's previous credit card loans at Home credit on a monthly basis. There may be multiple records

corresponding to a current applicant's previous application if they have a credit history lasting longer than a month.

POS_CASH_balance.csv (Primary Key: SK_ID_PREV, SK_ID_CURR,MONTHS_BALANCE) : This data consists of an individual's monthly balance of loans previously taken from Home credit (Consumer credit and cash). There may be multiple records corresponding to each user if they have a credit history exceeding one month at Home Credit.

HomeCredit_columns_description.csv : This is a data dictionary that essentially describes each column in the given tables (csv files) in the dataset.

TASK: Our objective with the Home Credit Default Risk Prediction project is to predict whether or not a client will repay a loan with high accuracy using alternative data sources such as telco, transactional data, payment behavior.etc.This phase of the project focuses on performing feature engineering on data attributes, creation and evaluation,hyper-parameter tuning of the various pipelines and do a Kaggle submission.

*In this phase, we are tackling tasks of building a multi-layer perceptron model in PyTorch for loan default classification, evaluating our model for data and memory leakage, model the pipeline with three hidden layers, evaluate the model using AUC-ROC score, visualize accuracy, loss and AUC score using Tensorboard and making Kaggle submission.

We have built a neural network model using three hidden layers with 512, 256 and 128 neurons respectively. We evaluated the model for any data and memory leakage using machine learning cardinal sins criteria. We built a multi-layer perceptron pipeline with batch size of 100 and epoch of 50. We applied a rectified linear unit function (ReLU) after each hidden layer. The loss function for our model is Binary Cross Entropy Loss and we have implemented it using BCEWithLogitsLoss function of PyTorch. This PyTorch function combines a Sigmoid layer and Binary Cross Entropy loss in one single class. Thus, it's more stable than using a plain sigmoid followed by BCELoss. We have optimized the loss function using stochastic gradient descent with a learning rate of 0.00001, momentum factor of 0.9 and dampening for momentum of zero.We evaluated our model and finally submitted the scores to Kaggle.

Feature Engineering

- **Important Features added in this phase are following** :
  total_amt_req_credit_bureau,has_job,has_children,cluster_days_employed,house_variables_sum

- Step 1: - We discarded columns which are having more than 40% of Null Values. The above table shows the count of NA values of remaining columns and their percentages

- Step 2: - We refer to the application_train data (and also application_test data also) as the primary table and the other files as the secondary tables (e.g., previous_application dataset). All tables can be joined using the primary key SK_ID_PREV.

- Step 3: - Transform all the secondary tables to features that can be joined into the main table the application table (labeled and unlabeled)'bureau', 'bureau_balance', 'credit_card_balance', 'installments_payments','previous_application', 'POS_CASH_balance'.

- Step 4: -Merge the transformed secondary tables with the primary tables (i.e., the application_train data (the labeled dataset) and with the application_test data (the unlabeled submission dataset)), thereby leading to X_train, y_train, X_valid, etc. When joining this data in the context of pipelines, different strategies come to mind with various tradeoffs. A simple feature could be the number of previous applications. Other summary features of original features such as AMT_APPLICATION, AMT_CREDIT could be based on average, min,max, median, etc.

- Step 5: - **Features selected from the installments_payments and their correlations with TARGET are** : AMT_DOWN_PAYMENT 0.002496 CNT_PAYMENT 0.002341 DAYS_LAST_DUE_1ST_VERSION 0.001908 AMT_CREDIT 0.001833 AMT_APPLICATION 0.001689 AMT_GOODS_PRICE 0.001676 SK_ID_CURR 0.001107 NFLAG_INSURED_ON_APPROVAL 0.000879 RATE_DOWN_PAYMENT 0.000850 RATE_INTEREST_PRIMARY 0.000542 SK_ID_PREV 0.000362 DAYS_DECISION -0.000482 AMT_ANNUITY -0.000492 DAYS_FIRST_DUE -0.000943 SELLERPLACE_AREA -0.000954 DAYS_TERMINATION -0.001072 NFLAG_LAST_APPL_IN_DAY -0.001256 DAYS_FIRST_DRAWING -0.001293 DAYS_LAST_DUE -0.001940 HOUR_APPR_PROCESS_START -0.002285 RATE_INTEREST_PRIVILEGED -0.026427

- Step 6: - **Features selected from the installments_payments and their correlations with TARGET are** : SK_ID_PREV 0.002891 NUM_INSTALMENT_VERSION 0.002511 NUM_INSTALMENT_NUMBER 0.000626 SK_ID_CURR -0.000781 AMT_PAYMENT -0.003512 DAYS_INSTALMENT -0.003955 AMT_INSTALMENT -0.003972 DAYS_ENTRY_PAYMENT -0.004046

- Step 7: - **Features selected from the POS_CASH_balance and their correlations with TARGET are** : CNT_INSTALMENT_FUTURE 0.002811 MONTHS_BALANCE 0.002775 SK_ID_PREV 0.002164 CNT_INSTALMENT 0.001434 SK_DPD 0.000050 SK_ID_CURR -0.000136 SK_DPD_DEF -0.001362

- Step 8: - **Features selected from the bureau and bureau_balance and their correlations with TARGET are** : DAYS_CREDIT_UPDATE 0.002159 DAYS_CREDIT_ENDDATE 0.002048 SK_ID_BUREAU 0.001550 DAYS_CREDIT 0.001443 AMT_CREDIT_SUM 0.000218 DAYS_ENDDATE_FACT 0.000203 AMT_ANNUITY 0.000189 AMT_CREDIT_MAX_OVERDUE -0.000389 CNT_CREDIT_PROLONG -0.000495 AMT_CREDIT_SUM_LIMIT -0.000558 AMT_CREDIT_SUM_DEBT -0.000946 SK_ID_CURR -0.001070 AMT_CREDIT_SUM_OVERDUE -0.001464 CREDIT_DAY_OVERDUE -0.001815 SK_ID_BUREAU 0.001223 MONTHS_BALANCE -0.005262

- Step 9: - **Features selected from the bureau and bureau_balance and their correlations with TARGET are** : CNT_DRAWINGS_ATM_CURRENT 0.001908 AMT_DRAWINGS_ATM_CURRENT 0.001520 AMT_INST_MIN_REGULARITY 0.001435 SK_ID_CURR 0.001086 AMT_CREDIT_LIMIT_ACTUAL 0.000515 AMT_BALANCE 0.000448 SK_ID_PREV 0.000446 AMT_RECIVABLE 0.000412 AMT_TOTAL_RECEIVABLE 0.000407 AMT_RECEIVABLE_PRINCIPAL 0.000383 SK_DPD 0.000092 SK_DPD_DEF -0.000201 CNT_INSTALMENT_MATURE_CUM -0.000342 MONTHS_BALANCE -0.000768 AMT_PAYMENT_CURRENT -0.001129 AMT_PAYMENT_TOTAL_CURRENT -0.001395 AMT_DRAWINGS_CURRENT -0.001419 CNT_DRAWINGS_CURRENT -0.001764 CNT_DRAWINGS_OTHER_CURRENT -0.001833 CNT_DRAWINGS_POS_CURRENT -0.002387 AMT_DRAWINGS_OTHER_CURRENT -0.002672 AMT_DRAWINGS_POS_CURRENT -0.003518

- Step 10: - **Proceed with the learning pipeline using X_train, y_train, X_valid, etc.Generate a submission file using the learnt model**

Neural Network

- We have implemented a Neural Network Model using PyTorch. In this multi-layer perceptron model, we have used three hidden layers with 512, 256 and 128 neurons for each hidden layer respectively.
- We have used a batch size of 100 and number of iterations i.e. epochs of 50. We have applied a rectified linear unit function after each hidden layer.
- We have used the BCEWithLogitsLoss function from PyTorch. The Sigmoid layer and the BCELoss is combined into one single class in this function which is numerically more stable than using a plain Sigmoid followed by a BCELoss.
- Thus, by combining the operations into one layer, we take advantage of the log-sum-exp trick for numerical stability.
- The Binary Cross Entropy loss function is given as

$l(x, y) = L = l_1, ..., l_N{}^T$
$l_n = -w_n[y_n . log\sigma(x_n) + (1 - y_n) . log(1 - \sigma(x_n))]$
where
N is the batch size

- We have optimized the loss function using stochastic gradient descent. We have implemented the SGD function from PyTorch with a learning rate of 0.001, momentum factor of 0.9 and dampening for moment We um of zero. We have trained models 20 times and found train and validation loss for each iteration. We have applied a sigmoid activation function from PyTorch while evaluating our model.

Here's the Final Architecture of the Neural Network used:

```
mlp_model
```

```
Model(
  (all_embeddings): ModuleList(
    (0): Embedding(2, 1)
    (1): Embedding(3, 2)
    (2): Embedding(2, 1)
    (3): Embedding(2, 1)
    (4): Embedding(8, 4)
    (5): Embedding(8, 4)
    (6): Embedding(5, 3)
    (7): Embedding(6, 3)
    (8): Embedding(6, 3)
    (9): Embedding(19, 10)
    (10): Embedding(7, 4)
    (11): Embedding(58, 29)
    (12): Embedding(5, 3)
    (13): Embedding(4, 2)
    (14): Embedding(8, 4)
    (15): Embedding(3, 2)
    (16): Embedding(20, 10)
    (17): Embedding(3, 2)
    (18): Embedding(3, 2)
    (19): Embedding(2, 1)
    (20): Embedding(2, 1)
    (21): Embedding(2, 1)
    (22): Embedding(9, 5)
    (23): Embedding(11, 6)
    (24): Embedding(10, 5)
    (25): Embedding(10, 5)
    (26): Embedding(1, 1)
  )
  (bn_cont): BatchNorm1d(107, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (layers): Sequential(
    (0): Linear(in_features=222, out_features=512, bias=True)
    (1): ReLU(inplace=True)
    (2): BatchNorm1d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (3): Linear(in_features=512, out_features=256, bias=True)
    (4): ReLU(inplace=True)
    (5): BatchNorm1d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (6): Linear(in_features=256, out_features=128, bias=True)
    (7): ReLU(inplace=True)
    (8): BatchNorm1d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (9): Linear(in_features=128, out_features=1, bias=True)
```

```
    )
)
```

Leakage

- The main objective of our project is to predict whether a person should be given a loan or not, essentially we perfrom predictive modeling. The challenge here is that we have to make predictions on unseen data using data we possess, this is the data we use to train and evaluate our model, therefore we must be very careful how we handle the data.
- What often happens is that before training the model we split the data into train and test splits and unless this is done properly there is seepage of train data into test and vice versa and as a result our train and test accuracies will come out to be high, but it may perform poorly in production deployment environment.
- One example is when we standardize/normalize data before the test and train split we are using the mean and standard deviation and these figures are influenced by values that wil eventually be a part of our test data thus causing a leakage of information between train and test sets, therefore we use fit_transform on our train data and transform on the test data.
- In the case of our project, the advantage we had with our dataset was that the train and test data was split beforehand from the source, therefore reducing the possibility of data leakage. To further reduce the scope of Data leakage we also set aside a validation set that was 20% of the train data. We also applied the normalization separately for the train and test sets in order to minimize the leakage.
- Considering all the measures we have taken to minimize we believe that there is no considerable leakage in our pipeline.

Violation of Cardinal sins of ML

Based on our investigation of the following we believe we have avoided Cardinal sins of ML -

1. As our Train accuracy 0.765 and Validation accuracy 0.726 are relatively close we know that the model does not overfit.
2. We ensure that the test data remains unseen by the model until the model evaluation step.

Modeling Pipelines:

(**Phase-3 Pipeline**)

1.) **Data Cleaning and Pre-processing:**

- Get data from Kaggle.
- Understanding the credit domain and datasets.
- Prepping the data to better explore the underlying patterns.
- Numerical features: Missing values are imputed with mean and then data is standardized.
- Categorical features: Missing values are imputed with “missing” and then one hot encoded.

2.) **Feature Engineering**

- Create new features from existing
- Join Tables
- Select best features
- A logistic regression model with L1 regularization was fitted. Using “SelectFromModel” meta estimator, features with zero weights were ignored and rest of them were passed downstream

3.) **MLP Model**

- We have used a Multi-layer perceptron model with three hidden layers and 512, 256, 128 neurons respectively for each layer.

Hyperparameters and settings considered

Epochs: 50

Batch size: 100

Hidden layers: 3

Neuron count: 512, 256 and 128

Hyper Parmeters	Parameter with the best perfomance
Number of Epochs	50
Number of hidden layers and units	3 and [512,256,128]
Batch Size	100
Activation Function	ReLU
Dropout regularization	0.08
Networks Weights Initialization	Uniform Initialization
Learning Rate	0.00001

Loss function: Binary Cross Entropy Loss

Optimizer: Stochastic Gradient Descent with momentum=0.9, dampening=0

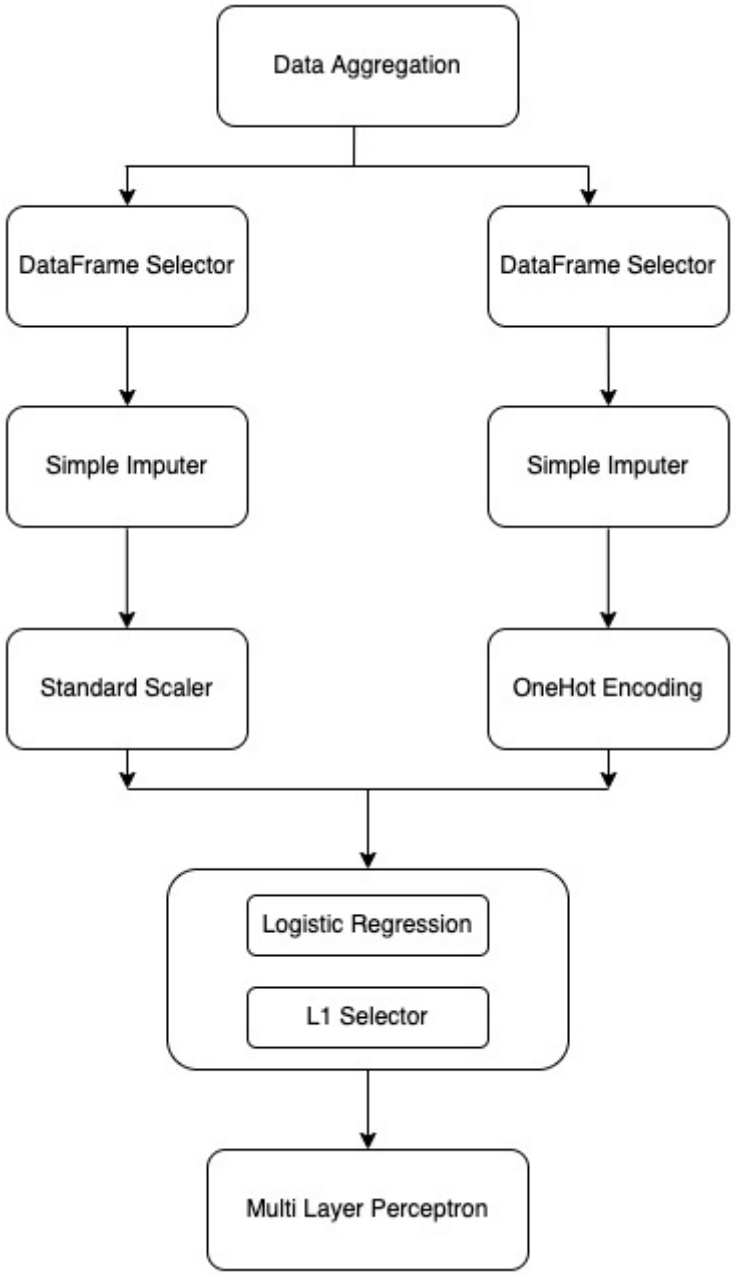
Binary Cross Entropy loss equation:

Latex code:

$$l(x, y) = L = l_1, \ldots\ldots, l_N{}^T$$
$$l_n = -w_n[y_n . log\sigma(x_n) + (1 - y_n) . log(1 - \sigma(x_n))]$$

where
N is the batch size

Block Diagram



MLP Results Dataframe

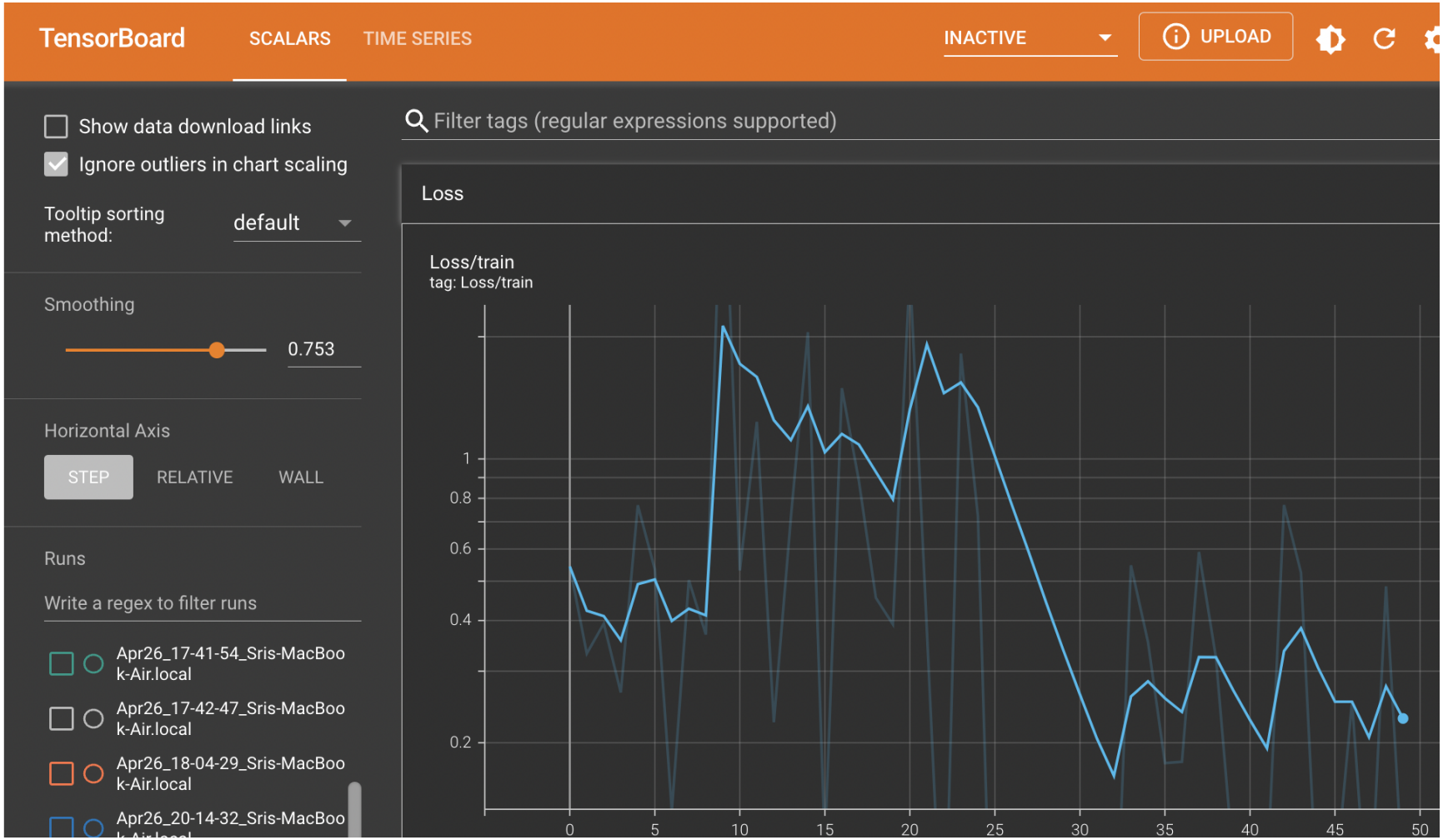
S.No	Epochs	No. of hidden layers	Neurons per layer	Train AUC	Validation AUC	Test AUC
1	10	1	800	0.7	0.58	0.55
2	50	3	[512,256,128]	0.765	0.726	0.724

Experimental results and Discussion

- We evaluated our MLP neural network model based on the ROC-AUC metric. The MLP model gave an test ROC-AUC score of 0.726
- We submitted the Multi-Layer Perceptron model on Kaggle to get a public score of 0.72408 and private score of 0.72666.
- We do not have a significant increase in the test- AUC scores compared to Phase-2 even after Hyper-parameter tuning
- This slight difference in AUC scores from previous phases could be because we have taken less data compared to previous phases for GPU performance of the MLP model.
- But It's still considerably one of the top scores submitted to kaggle. So, In future we could try other deep learning models such as DNN's and CNN's to improve the performance further.
- Following are the Results Comparision of all the phases, highlighting Phase-3 Results.

Model	Train Accuracy	Test Accuracy	Train AUC	Test AUC	
Baseline (Logistic Regression)	92	91.9	0.503	0.502	Past Results
Decision Tree	85.1	85.1	0.538	0.536	
Random Forest	92.4	91.3	0.714	0.748	
Gradient Boosting - Light GBM	92.8	91.84	0.757	0.748	
Gradient Boosting - XGBoost	92	91.23	0.748	0.741	
MLP- 1 hidden layer	-	-	**0.7**	**0.55**	Current Results
MLP- 3 hidden layers	-	-	**0.765**	**0.724**	

Tensorboard Visualization of Loss/Epoch (Train)



Conclusion

The HCDR project predicts whether or not a client will repay a loan.These predictions are important as they will help banks to provide clients with loans who have no credit histories thus benefiting both the client and banks.

In this phase we have trained and evaluated a Multi-Layer perceptron neural network model with three layers, ReLU activation function, with a sigmoid function to map the output to their classes. We also checked our pipeline for data leakages and found there to be none.

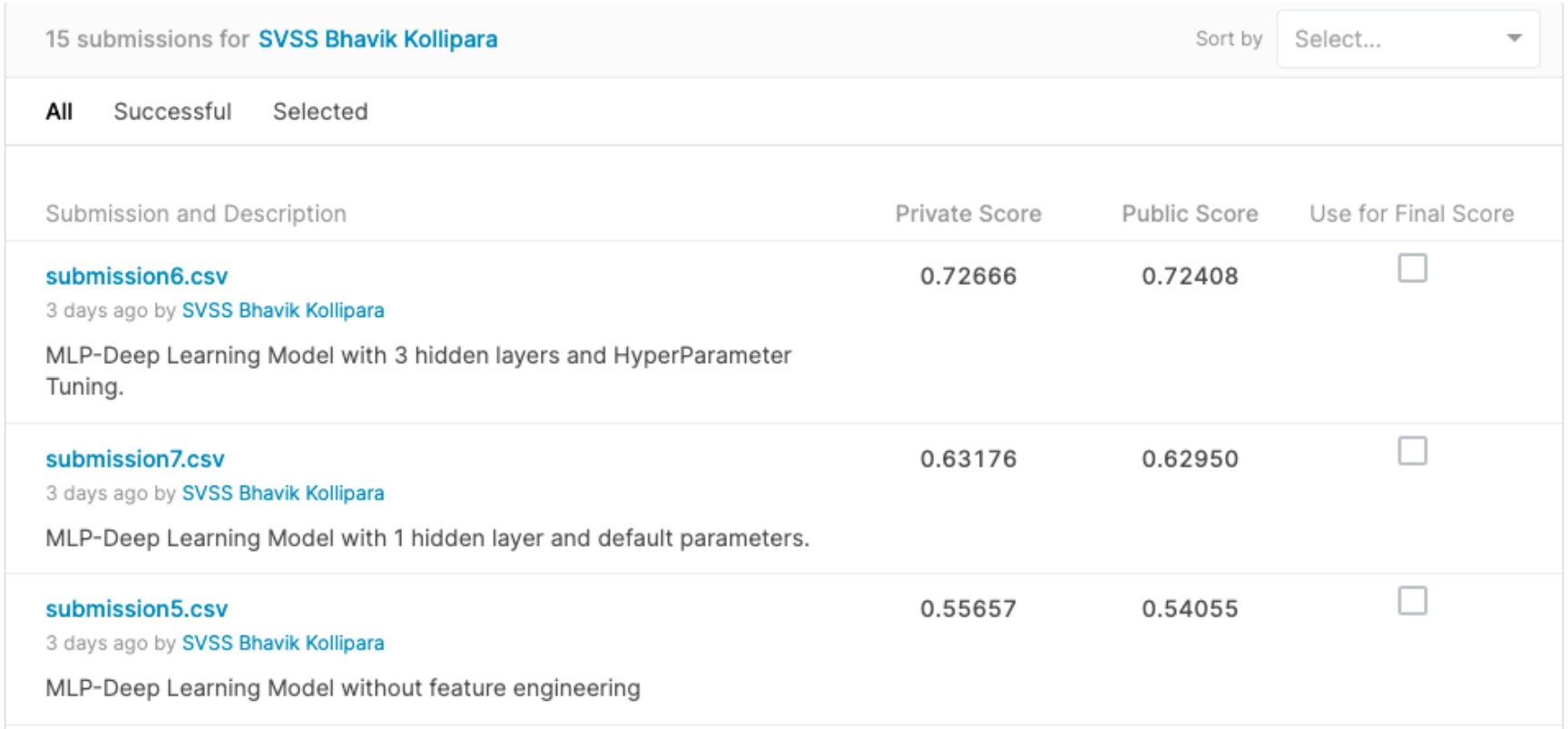
Our Multi-Layer Perceptron model had a test AUC of 0.726. We made a Kaggle submission for our Multi-Layer Perceptron model to get a Kaggle public score of 0.72408 and private score of 0.72666.

Our Future steps include:

- In Further, we would like to address the performance of Deep Learning Model using various other Architectures.

Kaggle Submission

Submission Link: https://www.kaggle.com/competitions/home-credit-default-risk/submissions Please find a screenshot of our best kaggle submission.



15 submissions for **SVSS Bhavik Kollipara** Sort by Select...

All Successful Selected

Submission and Description	Private Score	Public Score	Use for Final Score
submission6.csv 3 days ago by SVSS Bhavik Kollipara MLP-Deep Learning Model with 3 hidden layers and HyperParameter Tuning.	0.72666	0.72408	☐
submission7.csv 3 days ago by SVSS Bhavik Kollipara MLP-Deep Learning Model with 1 hidden layer and default parameters.	0.63176	0.62950	☐
submission5.csv 3 days ago by SVSS Bhavik Kollipara MLP-Deep Learning Model without feature engineering	0.55657	0.54055	☐

In [ ]: