06_2_Intro_NVTabular_XGBoost

January 28, 2021

[1]:
```
# The MIT License (MIT)

# Copyright (c) 2020, NVIDIA CORPORATION.

# Permission is hereby granted, free of charge, to any person obtaining a copy␣
↪of
# this software and associated documentation files (the "Software"), to deal in
# the Software without restriction, including without limitation the rights to
# use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies␣
↪of
# the Software, and to permit persons to whom the Software is furnished to do␣
↪so,
# subject to the following conditions:

# The above copyright notice and this permission notice shall be included in all
# copies or substantial portions of the Software.

# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
# IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,␣
↪FITNESS
# FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
# COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER
# IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN
# CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE
```

1 Tutorial: Feature Engineering for Recommender Systems

2 6. Scaling to Production Systems

2.1 6.2. Introduction to NVTabular

With the rapid growth in scale of industry datasets, deep learning (DL) recommender models have started to gain advantages over traditional methods by capitalizing on large amounts of training data.

The current challenges for training large-scale recommenders include:

- **Huge datasets:** Commercial recommenders are trained on huge datasets, often several

terabytes in scale.
- **Complex data preprocessing and feature engineering pipelines:** Datasets need to be preprocessed and transformed into a form relevant to be used with DL models and frameworks. In addition, feature engineering creates an extensive set of new features from existing ones, requiring multiple iterations to arrive at an optimal solution.
- **Input bottleneck:** Data loading, if not well optimized, can be the slowest part of the training process, leading to under-utilization of high-throughput computing devices such as GPUs.
- **Extensive repeated experimentation:** The whole data engineering, training, and evaluation process is generally repeated many times, requiring significant time and computational resources.

NVTabular is a library for fast tabular data tranformation and loading, manipulating terabyte-scale datasets quickly. It provides best practices for feature engineering and preprocessing and a high-level abstraction to simplify code accelerating computation on the GPU using the RAPIDS cuDF library.

Resoruces: * GitHub * GTC2020 Keynote Announcement * GTC2020 Session * NVIDIA DevBlog * Examples

2.2 HandsOn

NVTabular has 4 main components: **1. Dataset:** A *dataset* contains a list of files and iterates over the files. If necessary, it will read a file in chunks. **2. Op:** An *Op* defines the calculation, which should be exectued. For example, an op could be to collect the mean/std for a column, fill in missing values or combine two categories. **3. Workflow:** A *workflow* orchestrates the pipeline * It defines the contex, which columns are categorical, numerical or the label * It registers the operations (calculation) for the different column types * It optimizes the tasks by reordering the operations * It collects the required statistics for operations (e.g. the mean/std for normalization) * It applies the final operations to the dataset

4. Dataloader: NVTabular provides optimized dataloader for tabular data in PyTorch and Tensorflow

Let's build the preprocessing and feature engineering pipeline, learned in the handson tutorial, with NVTabular

3 Data processing for XGBoost training

[1]:
```
import nvtabular as nvt
from nvtabular import ops
```

First, we define the paths for the training and validation dataset.

[2]:
```
import glob

train_paths = glob.glob('./data/train.parquet')
valid_paths = glob.glob('./data/valid.parquet')
```

```
train_dataset = nvt.Dataset(train_paths, engine='parquet', part_mem_fraction=0.
↪15)
valid_dataset = nvt.Dataset(valid_paths, engine='parquet', part_mem_fraction=0.
↪15)
```

```
/conda/envs/nvtabular/lib/python3.7/site-packages/nvtabular/io/parquet.py:75:
UserWarning: Row group size 2565426129 is bigger than requested part_size
2376558182
  f"Row group size {rg_byte_size_0} is bigger than requested part_size "
```

[3]:
```
train_paths, valid_paths
```

[3]:
```
(['../data/train.parquet'], ['../data/valid.parquet'])
```

We define the data schema.

[4]:
```
proc = nvt.Workflow(
    cat_names=['product_id', 'brand', 'user_id',
       'user_session', 'cat_0', 'cat_1', 'cat_2', 'cat_3',
       'ts_hour', 'ts_minute', 'ts_weekday', 'ts_day', 'ts_month', 'ts_year'],
    cont_names=['price', 'timestamp'],
    label_name=['target']
)
```

[5]:
```
proc.add_feature([
    ops.LambdaOp(
        op_name = 'user_id',
        f = lambda col, gdf: col.astype(str) + '_' + gdf['user_id'].astype(str),
        columns = ['product_id', 'brand', 'ts_hour', 'ts_minute'],
        replace=False
    ),
    ops.LambdaOp(
        op_name = 'user_id_brand',
        f = lambda col, gdf: col.astype(str) + '_' + gdf['user_id'].astype(str)␣
↪+ '_' + gdf['brand'].astype(str),
        columns = ['ts_hour', 'ts_weekday', 'cat_0', 'cat_1', 'cat_2'],
        replace=False
    ),
    ops.Categorify(
        freq_threshold=15,
        columns = [x + '_user_id' for x in ['product_id', 'brand', 'ts_hour',␣
↪'ts_minute']] + [x + '_user_id_brand' for x in ['ts_hour', 'ts_weekday',␣
↪'cat_0', 'cat_1', 'cat_2']] + ['product_id', 'brand', 'user_id',␣
↪'user_session', 'cat_0', 'cat_1', 'cat_2', 'cat_3', 'ts_hour', 'ts_minute',␣
↪'ts_weekday', 'ts_day', 'ts_month', 'ts_year']
    ),
    ops.LambdaOp(
```

```
        op_name = 'product_id',
        f = lambda col, gdf: col.astype(str) + '_' + gdf['product_id'].
↪astype(str),
        columns = ['brand', 'user_id', 'cat_0'],
        replace=False
    ),
    ops.JoinGroupby(
        cont_names=[]
    ),
    ops.TargetEncoding(
        cat_groups = ['brand', 'user_id', 'product_id', 'cat_2',␣
↪['ts_weekday','ts_day']],
        cont_target= 'target',
        kfold=5,
        fold_seed=42,
        p_smooth=20,
    )
])
```

[6]:
```
!rm -r ./output_nvt_train/
!mkdir ./output_nvt_train/

!rm -r ./output_nvt_valid/
!mkdir ./output_nvt_valid/

proc.apply(train_dataset,
           output_path='./output_nvt_train/'
          )

proc.apply(valid_dataset,
           output_path='./output_nvt_valid/',
           record_stats=False
          )
```

We added following data operations:

Combine Categories

Categorify Categories

Count Encoding (JoinGroupBy)

Target Encoding More features/ops are continously added to NVTabular and the other ops will follow soon. We load the train and valid dataset and run the additional features. First, we load the data, again.

[7]:
```
import IPython

app = IPython.Application.instance()
```

```
app.kernel.do_shutdown(True)
```

[7]: {'status': 'ok', 'restart': True}

[2]:
```
import cudf
import pandas as pd
import glob

train_paths = glob.glob('./output_nvt_train/*.parquet')
valid_paths = glob.glob('./output_nvt_valid/*.parquet')
```

[3]:
```
train = cudf.concat([cudf.read_parquet(x) for x in train_paths])
valid = cudf.concat([cudf.read_parquet(x) for x in valid_paths])
```

[4]:
```
train.drop(['user_session', 'brand_product_id', 'user_id_product_id',␣
 ↪'cat_0_product_id'], inplace=True)
valid.drop(['user_session', 'brand_product_id', 'user_id_product_id',␣
 ↪'cat_0_product_id'], inplace=True)
```

[4]:
```
           price           timestamp  product_id  brand  user_id  cat_0  \
0          49.91 2020-03-01 00:00:59       10955   2914    55813      8
1         397.10 2020-03-01 00:01:20          31    155        0      6
2         823.70 2020-03-01 00:01:52         100    155        0      6
3         422.15 2020-03-01 00:02:14       23020   2195        0     12
4          69.24 2020-03-01 00:02:15        5632    176        0      3
…            …                   …         …      …      …     …
2461714     2.65 2020-03-31 23:57:47       28311    566        0      3
2461715   234.96 2020-03-31 23:58:19       41306      0        0      0
2461716   223.92 2020-03-31 23:58:20        5336   2281    23311      3
2461717   100.36 2020-03-31 23:59:19       42652   2771        0      3
2461718   319.41 2020-03-31 23:59:27       42366   2281        0      6

         cat_1  cat_2  cat_3  ts_hour  …  cat_1_user_id_brand  \
0           49      0      0        1  …                    0
1           51     41      0        1  …                    0
2           51     41      0        1  …                    0
3           53      0      0        1  …                    0
4           20     82      0        1  …                    0
…          …    …    …      …  …                  …
2461714     35     42      0       24  …                    0
2461715      0      0      0       24  …                    0
2461716     20     82      0       24  …                    0
2461717     20     82      0       24  …                    0
2461718     51     41      0       24  …                    0

         cat_2_user_id_brand  brand_product_id_count  \
0                          0                     156
```

```
1                        0                141423
2                        0                  7568
3                        0                    25
4                        0                    50
…                    …                    …
2461714                  0                    66
2461715                  0                    51
2461716                  0                  1013
2461717                  0                    55
2461718                  0                 15300

         user_id_product_id_count cat_0_product_id_count  TE_brand_target  \
0                            <NA>                    156         0.257370
1                           84792                 103293         0.421481
2                            4856                   3500         0.421481
3                              24                     22         0.106112
4                              43                     50         0.323030
…                         …                    …                …
2461714                        47                     80         0.166703
2461715                        46                     52         0.301020
2461716                         1                   1013         0.439617
2461717                        44                     55         0.318733
2461718                      9802                  15301         0.439617

         TE_user_id_target  TE_product_id_target  TE_cat_2_target  \
0                 0.239448              0.399650         0.322939
1                 0.338761              0.521025         0.459643
2                 0.338761              0.328590         0.459643
3                 0.338761              0.251966         0.322939
4                 0.338761              0.347693         0.350999
…                     …                     …                …
2461714           0.338761              0.221259         0.396022
2461715           0.338761              0.338034         0.322939
2461716           0.549198              0.349795         0.350999
2461717           0.338761              0.564513         0.350999
2461718           0.338761              0.525706         0.459643

         TE_ts_weekday_ts_day_target
0                           0.389486
1                           0.389486
2                           0.389486
3                           0.389486
4                           0.389486
…                              …
2461714                     0.374029
2461715                     0.374029
2461716                     0.374029
```

```
2461717                        0.374029
2461718                        0.374029

[2461719 rows x 33 columns]
```

We define the functions for additional feature engineering.

```python
[5]: import cupy

# TARGET ENCODE WITH KFOLD

def target_encode2(train, valid, col, target='target', kfold=5, smooth=20,␣
 ↪verbose=True):
    """
        train:  train dataset
        valid:  validation dataset
        col:    column which will be encoded (in the example RESOURCE)
        target: target column which will be used to calculate the statistic
    """

    # We assume that the train dataset is shuffled
    train['kfold'] = ((train.index) % kfold)
    # We keep the original order as cudf merge will not preserve the original␣
 ↪order
    train['org_sorting'] = cupy.arange(len(train), dtype="int32")
    # We create the output column, we fill with 0
    col_name = '_'.join(col)+'_'+str(smooth)
    train['TE_' + col_name] = 0.
    for i in range(kfold):
        ###################################
        # filter for out of fold
        # calculate the mean/counts per group category
        # calculate the global mean for the oof
        # calculate the smoothed TE
        # merge it to the original dataframe
        ###################################

        df_tmp = train[train['kfold']!=i]
        mn = df_tmp[target].mean()
        df_tmp = df_tmp[col + [target]].groupby(col).agg(['mean', 'count']).
 ↪reset_index()
        df_tmp.columns = col + ['mean', 'count']
        df_tmp['TE_tmp'] = ((df_tmp['mean']*df_tmp['count'])+(mn*smooth)) /␣
 ↪(df_tmp['count']+smooth)
        df_tmp_m = train[col + ['kfold', 'org_sorting', 'TE_' + col_name]].
 ↪merge(df_tmp, how='left', left_on=col, right_on=col).
 ↪sort_values('org_sorting')
```

```
        df_tmp_m.loc[df_tmp_m['kfold']==i, 'TE_' + col_name] = df_tmp_m.
↪loc[df_tmp_m['kfold']==i, 'TE_tmp']
        train['TE_' + col_name] = df_tmp_m['TE_' + col_name].fillna(mn).values


    ###################################
    # calculate the mean/counts per group for the full training dataset
    # calculate the global mean
    # calculate the smoothed TE
    # merge it to the original dataframe
    # drop all temp columns
    ###################################

    df_tmp = train[col + [target]].groupby(col).agg(['mean', 'count']).
↪reset_index()
    mn = train[target].mean()
    df_tmp.columns = col + ['mean', 'count']
    df_tmp['TE_tmp'] = ((df_tmp['mean']*df_tmp['count'])+(mn*smooth)) /␣
↪(df_tmp['count']+smooth)
    valid['org_sorting'] = cupy.arange(len(valid), dtype="int32")
    df_tmp_m = valid[col + ['org_sorting']].merge(df_tmp, how='left',␣
↪left_on=col, right_on=col).sort_values('org_sorting')
    valid['TE_' + col_name] = df_tmp_m['TE_tmp'].fillna(mn).values

    valid = valid.drop('org_sorting', axis=1)
    train = train.drop('kfold', axis=1)
    train = train.drop('org_sorting', axis=1)
    return (train, valid, 'TE_'+col_name)

def group_binning(df, valid, q_list = [0.1, 0.25, 0.5, 0.75, 0.9]):
    df['price_bin'] = -1
    valid['price_bin'] = -1

    for i, q_value in enumerate(q_list):
        print(q_value)
        q = df[['cat_012', 'price']].groupby(['cat_012']).quantile(q_value)
        q = q.reset_index()
        q.columns = ['cat_012', 'price' + str(q_value)]
        df = df.merge(q, how='left', on='cat_012')
        valid = valid.merge(q, how='left', on='cat_012')
        if i == 0:
            df.loc[df['price']<=df['price' + str(q_value)], 'price_bin'] = i
            valid.loc[valid['price']<=valid['price' + str(q_value)],␣
↪'price_bin'] = i
        else:
            df.loc[(df['price']>df['price' + str(q_list[i-1])]) &␣
↪(df['price']<=df['price' + str(q_value)]), 'price_bin'] = i
```

```
            valid.loc[(valid['price']>valid['price' + str(q_list[i-1])]) &␣
↪(valid['price']<=valid['price' + str(q_value)]), 'price_bin'] = i
        if i>=2:
            df.drop(['price' + str(q_list[i-2])], axis=1, inplace=True)
            valid.drop(['price' + str(q_list[i-2])], axis=1, inplace=True)

    df.loc[df['price']>df['price' + str(q_value)], 'price_bin'] = i+1
    df.drop(['price' + str(q_list[i-1])], axis=1, inplace=True)
    df.drop(['price' + str(q_list[i])], axis=1, inplace=True)

    valid.loc[valid['price']>valid['price' + str(q_value)], 'price_bin'] = i+1
    valid.drop(['price' + str(q_list[i-1])], axis=1, inplace=True)
    valid.drop(['price' + str(q_list[i])], axis=1, inplace=True)

def rolling_window(train, valid, col, offset):
    df = cudf.concat([train, valid])
    data_window = df[[col, 'date', 'target']].groupby([col, 'date']).
↪agg(['count', 'sum']).reset_index()
    data_window.columns = [col, 'date', 'count', 'sum']
    data_window.index = data_window['date']

    data_window_roll = data_window[[col, 'count', 'sum']].groupby([col]).
↪rolling(offset).sum().drop(col, axis=1)
    data_window_roll = data_window_roll.reset_index()
    data_window_roll.columns = [col, 'date', col + '_count_' + offset, col +␣
↪'_sum_' + offset]
    data_window_roll[[col + '_count_' + offset, col + '_sum_' + offset]] =␣
↪data_window_roll[[col + '_count_' + offset, col + '_sum_' + offset]].shift(1)
    data_window_roll.loc[data_window_roll[col]!=data_window_roll[col].shift(1),␣
↪[col + '_count_' + offset, col + '_sum_' + offset]] = 0
    data_window_roll[col + '_avg_' + offset] = (data_window_roll[col + '_sum_'␣
↪+ offset]/data_window_roll[col + '_count_' + offset]).fillna(-1)
    df = df.merge(data_window_roll, how='left', on=[col, 'date'])
    train = df[df['ts_month']!=3]
    valid = df[df['ts_month']==3]
    return(train, valid)
```

We bin the price and target encode the price bins.

[6]:
```
train['cat_012'] = train['cat_0'].astype(str) + '_' + train['cat_1'].
↪astype(str) + '_' + train['cat_2'].astype(str)
valid['cat_012'] = valid['cat_0'].astype(str) + '_' + valid['cat_1'].
↪astype(str) + '_' + valid['cat_2'].astype(str)
```

[7]:
```
group_binning(train, valid)
train, valid, name = target_encode2(train, valid, ['price_bin'], 'target',␣
↪smooth=20)
```

```
0.1
0.25
0.5
0.75
0.9
```

We create the time window features.

[8]:
```
train['date'] = cudf.from_pandas(pd.to_datetime(train['timestamp'].to_pandas()).
 ↪dt.date)
valid['date'] = cudf.from_pandas(pd.to_datetime(valid['timestamp'].to_pandas()).
 ↪dt.date)
```

```
/conda/envs/nvtabular/lib/python3.7/site-
packages/cudf/core/column/column.py:1396: UserWarning: Date32 values are not yet
supported so this will be typecast to a Date64 value
  UserWarning,
```

[9]:
```
train.columns
```

[9]:
```
Index(['price', 'timestamp', 'product_id', 'brand', 'user_id', 'cat_0',
       'cat_1', 'cat_2', 'cat_3', 'ts_hour', 'ts_minute', 'ts_weekday',
       'ts_day', 'ts_month', 'ts_year', 'target', 'product_id_user_id',
       'brand_user_id', 'ts_hour_user_id', 'ts_minute_user_id',
       'ts_hour_user_id_brand', 'ts_weekday_user_id_brand',
       'cat_0_user_id_brand', 'cat_1_user_id_brand', 'cat_2_user_id_brand',
       'brand_product_id_count', 'user_id_product_id_count',
       'cat_0_product_id_count', 'TE_brand_target', 'TE_user_id_target',
       'TE_product_id_target', 'TE_cat_2_target',
       'TE_ts_weekday_ts_day_target', 'cat_012', 'price_bin',
       'TE_price_bin_20', 'date'],
      dtype='object')
```

[10]:
```
train['product_user'] = train['product_id'].astype(str) + '_' +␣
 ↪train['user_id'].astype(str) + '_' + train['cat_2'].astype(str)
valid['product_user'] = valid['product_id'].astype(str) + '_' +␣
 ↪valid['user_id'].astype(str) + '_' + valid['cat_2'].astype(str)
# LABEL ENCODE CATEGORIES
comb = cudf.concat([train,valid],ignore_index=True)
for c in ['product_user']:
    tmp,code = comb[c].factorize()
    train[c] = tmp[:len(train)].values
    valid[c] = tmp[len(train):].values
```

[11]:
```
train.columns
```

[11]:
```
Index(['price', 'timestamp', 'product_id', 'brand', 'user_id', 'cat_0',
       'cat_1', 'cat_2', 'cat_3', 'ts_hour', 'ts_minute', 'ts_weekday',
```

```
'ts_day', 'ts_month', 'ts_year', 'target', 'product_id_user_id',
 'brand_user_id', 'ts_hour_user_id', 'ts_minute_user_id',
 'ts_hour_user_id_brand', 'ts_weekday_user_id_brand',
 'cat_0_user_id_brand', 'cat_1_user_id_brand', 'cat_2_user_id_brand',
 'brand_product_id_count', 'user_id_product_id_count',
 'cat_0_product_id_count', 'TE_brand_target', 'TE_user_id_target',
 'TE_product_id_target', 'TE_cat_2_target',
 'TE_ts_weekday_ts_day_target', 'cat_012', 'price_bin',
 'TE_price_bin_20', 'date', 'product_user'],
dtype='object')
```

We drop the unused columns

[12]:
```
train.drop(['timestamp', 'cat_012', 'price_bin', 'date'] , inplace=True)
valid.drop(['timestamp', 'cat_012', 'price_bin', 'date'] , inplace=True)
```

[12]:
```
          price  product_id  brand  user_id  cat_0  cat_1  cat_2  cat_3  \
0         49.91       10955   2914    55813      8     49      0      0
1        397.10          31    155        0      6     51     41      0
2        823.70         100    155        0      6     51     41      0
3        422.15       23020   2195        0     12     53      0      0
4         69.24        5632    176        0      3     20     82      0
…           …        …     …       …      …     …      …      …
2461714    2.65       28311    566        0      3     35     42      0
2461715  234.96       41306      0        0      0      0      0      0
2461716  223.92        5336   2281    23311      3     20     82      0
2461717  100.36       42652   2771        0      3     20     82      0
2461718  319.41       42366   2281        0      6     51     41      0

         ts_hour  ts_minute  …  brand_product_id_count  \
0              1          1  …                     156
1              1          2  …                  141423
2              1          2  …                    7568
3              1          3  …                      25
4              1          3  …                      50
…            …          …  …                       …
2461714       24         58  …                      66
2461715       24         59  …                      51
2461716       24         59  …                    1013
2461717       24         60  …                      55
2461718       24         60  …                   15300

         user_id_product_id_count cat_0_product_id_count  TE_brand_target  \
0                            <NA>                    156         0.257370
1                           84792                 103293         0.421481
2                            4856                   3500         0.421481
3                              24                     22         0.106112
```

```
4                                43                  50         0.323030
…                             …                   …             …
2461714                          47                  80         0.166703
2461715                          46                  52         0.301020
2461716                           1                1013         0.439617
2461717                          44                  55         0.318733
2461718                        9802               15301         0.439617

         TE_user_id_target  TE_product_id_target  TE_cat_2_target  \
0                 0.239448              0.399650         0.322939
1                 0.338761              0.521025         0.459643
2                 0.338761              0.328590         0.459643
3                 0.338761              0.251966         0.322939
4                 0.338761              0.347693         0.350999
…                      …                     …                  …
2461714           0.338761              0.221259         0.396022
2461715           0.338761              0.338034         0.322939
2461716           0.549198              0.349795         0.350999
2461717           0.338761              0.564513         0.350999
2461718           0.338761              0.525706         0.459643

         TE_ts_weekday_ts_day_target  TE_price_bin_20  product_user
0                           0.389486         0.366924        107700
1                           0.389486         0.366924        678289
2                           0.389486         0.366924         80568
3                           0.389486         0.366924        411417
4                           0.389486         0.366924       1116076
…                                …                …             …
2461714                     0.374029         0.366924        552770
2461715                     0.374029         0.366924        901890
2461716                     0.374029         0.366924       1088926
2461717                     0.374029         0.366924        940873
2461718                     0.374029         0.366924        931768

[2461719 rows x 34 columns]
```

[13]:
```
#train, valid = rolling_window(train, valid, 'product_user', '1D')
#train, valid = rolling_window(train, valid, 'product_user', '7D')
#train, valid = rolling_window(train, valid, 'product_user', '14D')
```

We save the new dataframes to disk.

[14]:
```
train.to_parquet('train_fe.parquet')
valid.to_parquet('valid_fe.parquet')
```

4 Train XGBoost

We train a XGBoost classifier

[15]:
```python
import IPython

app = IPython.Application.instance()
app.kernel.do_shutdown(True)
```

[15]:
```
{'status': 'ok', 'restart': True}
```

[3]:
```python
import cudf
```

[4]:
```python
train = cudf.read_parquet('train_fe.parquet')
valid = cudf.read_parquet('valid_fe.parquet')
```

[5]:
```python
train.columns
```

[5]:
```
Index(['price', 'product_id', 'brand', 'user_id', 'cat_0', 'cat_1', 'cat_2',
       'cat_3', 'ts_hour', 'ts_minute', 'ts_weekday', 'ts_day', 'ts_month',
       'ts_year', 'target', 'product_id_user_id', 'brand_user_id',
       'ts_hour_user_id', 'ts_minute_user_id', 'ts_hour_user_id_brand',
       'ts_weekday_user_id_brand', 'cat_0_user_id_brand',
       'cat_1_user_id_brand', 'cat_2_user_id_brand', 'brand_product_id_count',
       'user_id_product_id_count', 'cat_0_product_id_count', 'TE_brand_target',
       'TE_user_id_target', 'TE_product_id_target', 'TE_cat_2_target',
       'TE_ts_weekday_ts_day_target', 'TE_price_bin_20', 'product_user'],
      dtype='object')
```

[6]:
```python
features = [
    'price',
    'product_id',
    'brand',
    'user_id',
    'cat_0',
    'cat_1',
    'cat_2',
    'cat_3',
    'ts_hour',
    'ts_minute',
    'ts_weekday',
    'ts_day',
    'ts_month',
    'ts_year',
    'product_id_user_id',
    'brand_user_id',
    'ts_hour_user_id',
    'ts_minute_user_id',
```

```
    'ts_hour_user_id_brand',
    'ts_weekday_user_id_brand',
    'cat_0_user_id_brand',
    'cat_1_user_id_brand',
    'cat_2_user_id_brand',
    'brand_product_id_count',
    'user_id_product_id_count',
    'cat_0_product_id_count',
    'TE_brand_target',
    'TE_user_id_target',
    'TE_product_id_target',
    'TE_cat_2_target',
    'TE_ts_weekday_ts_day_target',
    'TE_price_bin_20'
]
```

[7]:

```
xgb_parms = {
    'max_depth':12,
    'learning_rate':0.02,
    'subsample':0.4,
    'colsample_bytree':0.4,
    #'eval_metric':'logloss',
    'eval_metric':'auc',
    'objective':'binary:logistic',
    'tree_method':'gpu_hist',
    'seed': 123
}
```

[8]:

```
import xgboost as xgb


NROUND = 1000
ESR = 50
VERBOSE_EVAL = 25
dtrain = xgb.DMatrix(data=train[features],label=train.target)
dvalid = xgb.DMatrix(data=valid[features],label=valid.target)

model = xgb.train(xgb_parms,
                  dtrain=dtrain,
                  evals=[(dtrain,'train'),(dvalid,'valid')],
                  num_boost_round=NROUND,
                  early_stopping_rounds=ESR,
                  verbose_eval=VERBOSE_EVAL)
```

```
[0]     train-auc:0.69630       valid-auc:0.59201
Multiple eval metrics have been passed: 'valid-auc' will be used for early
stopping.
```

```
Will train until valid-auc hasn't improved in 50 rounds.
[25]    train-auc:0.76714       valid-auc:0.64098
[50]    train-auc:0.76994       valid-auc:0.64318
[75]    train-auc:0.77490       valid-auc:0.64397
[100]   train-auc:0.77682       valid-auc:0.64449
[125]   train-auc:0.77918       valid-auc:0.64530
[150]   train-auc:0.78142       valid-auc:0.64558
[175]   train-auc:0.78271       valid-auc:0.64567
Stopping. Best iteration:
[137]   train-auc:0.78063       valid-auc:0.64599
```