02_1_Preprocessing

January 28, 2021

[ ]:
```
# 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 2. Preprocessing

In real-world applications, datasets are often messy. They are exported from production databases and often contains missing values. Treating missing values are important, as models handle them differently and some data operations/feature engineering ignore missing values.

[1]:
```
import IPython

import pandas as pd
import cudf
```

```python
import numpy as np
import cupy
import matplotlib.pyplot as plt

df_train = cudf.read_parquet('./data/train.parquet')
df_valid = cudf.read_parquet('./data/valid.parquet')
df_test = cudf.read_parquet('./data/test.parquet')
```

We can see, that multiple features have missing values.

[2]:
```python
df_train.isna().sum()
```

[2]:
```
event_time              0
event_type              0
product_id              0
brand              946612
price                   0
user_id                 0
user_session           66
target                  0
cat_0             1524551
cat_1             1524551
cat_2             5058060
cat_3            11454881
timestamp               0
ts_hour                 0
ts_minute               0
ts_weekday              0
ts_day                  0
ts_month                0
ts_year                 0
dtype: uint64
```

If we groupby the column brand to calculate the average and count, we can see, that the missing values with 946k observations are not included.

[3]:
```python
df_train[['brand', 'target']].groupby(['brand']).agg(['mean', 'count']).
↪sort_values(('target', 'count'), ascending=False).head(10)
```

[3]:
```
            target
              mean    count
brand
samsung   0.439618  2323417
apple     0.421482  2031101
xiaomi    0.339367  1082336
huawei    0.431646   357427
oppo      0.465100   154913
lg        0.347476   153196
```

```
lucente   0.454330   152024
sony      0.351208   140922
artel     0.340474   110264
cordiant  0.247761   109872
```

Depending on the datatype, there are different strategies for imputing missing values.

Categorical Features:

Imputing categorical features is easy - a unique category value (e.g. "UNKNOWN") can be imputed Important: Before imputing the missing values, it is beneficial to create a indicator column, which indicate if the a value was imputed or not. There is maybe a underlying pattern for the missing values and models can learn the pattern.

[4]:
```python
cols = ['brand', 'user_session', 'cat_0', 'cat_1', 'cat_2', 'cat_3']

for col in cols:
    df_train['NA_' + col] = df_train[col].isna().astype(np.int8)
    df_train[col].fillna('UNKNOWN', inplace=True)
```

[5]:
```python
df_train.isna().sum()
```

[5]:
```
event_time         0
event_type         0
product_id         0
brand              0
price              0
user_id            0
user_session       0
target             0
cat_0              0
cat_1              0
cat_2              0
cat_3              0
timestamp          0
ts_hour            0
ts_minute          0
ts_weekday         0
ts_day             0
ts_month           0
ts_year            0
NA_brand           0
NA_user_session    0
NA_cat_0           0
NA_cat_1           0
NA_cat_2           0
NA_cat_3           0
dtype: uint64
```

If we repeat the previous command, we can see that UNKOWN brands get calculated.

[6]:
```
df_train[['brand', 'target']].groupby(['brand']).agg(['mean', 'count']).
↪sort_values(('target', 'count'), ascending=False).head(10)
```

[6]:

	target	
	mean	count
brand		
samsung	0.439618	2323417
apple	0.421482	2031101
xiaomi	0.339367	1082336
UNKNOWN	0.301577	946612
huawei	0.431646	357427
oppo	0.465100	154913
lg	0.347476	153196
lucente	0.454330	152024
sony	0.351208	140922
artel	0.340474	110264

Numerical Features:

Imputing median for the numerical value (per group)

Imputing mean for numercial value (per group)

In some cases, we may know what value should be used as the default value (e.g. 0 for historical data or the max)

Important: For the same reason as in the categorical case, it is important to add a indicator column that the datapoint was imputed.

In our case, we do not have missing values in the numerical column price. Therefore, we artificially inject nans and then compare the difference.

[7]:
```
np.random.seed(42)
df_train.loc[np.random.random(df_train.shape[0])<0.01, 'price'] = None
df_train['price'].isna().mean()
```

[7]: 0.009995587782493818

We calculate the median per cat_2 and merge it to the dataset.

[8]:
```
df_median = df_train[['cat_2', 'price']].groupby('cat_2').median().reset_index()
df_median.columns = ['cat_2', 'price_median_per_cat2']
df_train = df_train.merge(df_median, how='left', on='cat_2')
```

We create an indicator column, when price was not available and then overwrite the missing values with the median.

[9]:
```
df_train['NA_price'] = df_train[col].isna().astype(np.int8)
```

```python
df_train.loc[df_train['price'].isna(), 'price'] = df_train.
 ↪loc[df_train['price'].isna(), 'price_median_per_cat2']
df_train.drop('price_median_per_cat2', inplace=True).head(5)
```

[9]:

```
                event_time event_type  product_id    brand  price    user_id  \
0  2019-12-01 12:27:02 UTC       cart    12700214  UNKNOWN  35.38  580243411
1  2019-12-01 12:27:02 UTC       cart    12700214  UNKNOWN  35.38  580243411
2  2019-12-01 12:27:02 UTC       cart    12700214  UNKNOWN  35.38  580243411
3  2019-12-01 12:27:02 UTC       cart    12700214  UNKNOWN  35.38  580243411
4  2019-12-01 12:27:02 UTC       cart    12700214  UNKNOWN  35.38  580243411

                           user_session  target    cat_0    cat_1  … ts_day  \
0  0cbf5e06-a782-4c74-8002-acf282026d82       0  UNKNOWN  UNKNOWN  …      1
1  0cbf5e06-a782-4c74-8002-acf282026d82       0  UNKNOWN  UNKNOWN  …      1
2  0cbf5e06-a782-4c74-8002-acf282026d82       0  UNKNOWN  UNKNOWN  …      1
3  0cbf5e06-a782-4c74-8002-acf282026d82       0  UNKNOWN  UNKNOWN  …      1
4  0cbf5e06-a782-4c74-8002-acf282026d82       0  UNKNOWN  UNKNOWN  …      1

   ts_month  ts_year  NA_brand  NA_user_session  NA_cat_0  NA_cat_1  NA_cat_2  \
0        12     2019         1                0         1         1         1
1        12     2019         1                0         1         1         1
2        12     2019         1                0         1         1         1
3        12     2019         1                0         1         1         1
4        12     2019         1                0         1         1         1

   NA_cat_3  NA_price
0         1         0
1         1         0
2         1         0
3         1         0
4         1         0

[5 rows x 26 columns]
```

[10]:

```python
df_train['price'].isna().mean()
```

[10]: 0.0

Predicting missing values: In Improving Deep Learning For Airbnb Search, the authors propose to use a DNN for missing user engagement features of new items (listenings). New items have no historical user engagements, such as # of views, # of bookings, etc.. In the paper, they train a DNN based on the meta information, such as price, location and predict the user engagements feature. This could be interpreted in what are the expected user engagement. Instead of the hand-crafted default values for missing user engagement, the authors replaced the missing values with the prediction of the DNN and showed that it reduced the error by 43% (offline test) and improved the overall bookings by 0.38% (online A/B test).

We shutdown the kernel.

[11]:
```python
app = IPython.Application.instance()
app.kernel.do_shutdown(False)
```

[11]: {'status': 'ok', 'restart': False}