05_2_TimeSeries_Differences

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 5. Feature Engineering

2.1 5.2. Differences

[1]:
```
import IPython

import pandas as pd
import numpy as np

import cudf
```

```
import cupy

np.random.seed(42)
```

[2]:
```
itemid = [1000001]*10 + [1000002]*5 + [1000001]*5 + [1000002]*5 + [1000001]*1 +␣
↪[1000002]*1 + [1000001]*2 + [1000002]*2
itemid += [1000001]*3 + [1000002]*2 + [1000001]*1 + [1000002]*1 + [1000001]*6 +␣
↪[1000002]*3 + [1000001]*2 + [1000002]*2
userid = np.random.choice(list(range(10000)), len(itemid))
action = np.random.choice(list(range(2)), len(itemid), p=[0.2, 0.8])

price = [100.00]*10 + [25.00]*5 + [100.00]*5 + [30.00]*5 + [125.00]*1 + [30.
↪00]*1 + [125.00]*2 + [30.00]*2
price += [110.00]*3 + [30.00]*2 + [110.00]*1 + [20.00]*1 + [90.00]*6 + [20.
↪00]*3 + [90.00]*2 + [20.00]*2

timestamp = [pd.to_datetime('2020-01-01')]*15
timestamp += [pd.to_datetime('2020-01-02')]*10
timestamp += [pd.to_datetime('2020-01-03')]*2
timestamp += [pd.to_datetime('2020-01-04')]*4
timestamp += [pd.to_datetime('2020-01-05')]*5
timestamp += [pd.to_datetime('2020-01-07')]*2
timestamp += [pd.to_datetime('2020-01-08')]*9
timestamp += [pd.to_datetime('2020-01-09')]*4

data = pd.DataFrame({
    'itemid': itemid,
    'userid': userid,
    'price': price,
    'action': action,
    'timestamp': timestamp
})

data = cudf.from_pandas(data)
```

2.2 Theory

Another category of powerful features is to calculate the differences to previous datapoints based on a timestamp. For example, we can calculate if the price changed of a product and how much the price change was.

[3]:
```
data[data['itemid']==1000001].head(10)
```

[3]:
```
    itemid  userid  price  action  timestamp
0  1000001    7270  100.0       1 2020-01-01
1  1000001     860  100.0       1 2020-01-01
2  1000001    5390  100.0       0 2020-01-01
```

```
3  1000001   5191  100.0        1 2020-01-01
4  1000001   5734  100.0        0 2020-01-01
5  1000001   6265  100.0        1 2020-01-01
6  1000001    466  100.0        1 2020-01-01
7  1000001   4426  100.0        1 2020-01-01
8  1000001   5578  100.0        1 2020-01-01
9  1000001   8322  100.0        0 2020-01-01
```

Tree-based or deep learning based models have difficulties processing these relationships on their own. Providing the models with these features can significantly improve the performance.

[4]:
```
offset = 1
data_shift = data[['itemid', 'timestamp', 'price']].groupby(['itemid',␣
↪'timestamp']).mean().reset_index()
data_shift.columns = ['itemid', 'timestamp', 'mean']
data_shift['mean_' + str(offset)] = data_shift['mean'].shift(1)
data_shift.loc[data_shift['itemid']!=data_shift['itemid'].shift(1), 'mean_' +␣
↪str(offset)] = None
data_shift['diff_' + str(offset)] = data_shift['mean'] - data_shift['mean_' +␣
↪str(offset)]
```

[5]:
```
data_shift.head(10)
```

[5]:
```
    itemid  timestamp   mean mean_1 diff_1
0  1000001 2020-01-01  100.0   <NA>   <NA>
1  1000001 2020-01-02  100.0  100.0    0.0
2  1000001 2020-01-03  125.0  100.0   25.0
3  1000001 2020-01-04  125.0  125.0    0.0
4  1000001 2020-01-05  110.0  125.0  -15.0
5  1000001 2020-01-07  110.0  110.0    0.0
6  1000001 2020-01-08   90.0  110.0  -20.0
7  1000001 2020-01-09   90.0   90.0    0.0
8  1000002 2020-01-01   25.0   <NA>   <NA>
9  1000002 2020-01-02   30.0   25.0    5.0
```

[6]:
```
data_shift.columns = ['itemid', 'timestamp', 'c1', 'c2', 'price_diff_1']
data_shift.drop(['c1', 'c2'], inplace=True).head(10)
```

[6]:
```
    itemid  timestamp price_diff_1
0  1000001 2020-01-01         <NA>
1  1000001 2020-01-02          0.0
2  1000001 2020-01-03         25.0
3  1000001 2020-01-04          0.0
4  1000001 2020-01-05        -15.0
5  1000001 2020-01-07          0.0
6  1000001 2020-01-08        -20.0
7  1000001 2020-01-09          0.0
```

```
8  1000002 2020-01-01         <NA>
9  1000002 2020-01-02          5.0
```

[7]:
```
data = data.merge(data_shift, how='left', on=['itemid', 'timestamp'])
```

[8]:
```
data.head()
```

[8]:

	itemid	userid	price	action	timestamp	price_diff_1
0	1000001	4658	110.0	0	2020-01-05	-15.0
1	1000001	1899	110.0	0	2020-01-05	-15.0
2	1000002	7734	30.0	1	2020-01-05	0.0
3	1000002	1267	30.0	1	2020-01-05	0.0
4	1000001	1528	110.0	1	2020-01-07	0.0

We can combine techniques of TimeSeries data and chain them together. For example, we can calculate the # of purchases per item and then compare the previous week with a the week, 2, 3 or 5 weeks ago. We can recognize patterns over time.

2.3 Practise

[9]:
```
import pandas as pd
import cudf
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')

df_train['brand'] = df_train['brand'].fillna('UNKNOWN')
df_valid['brand'] = df_valid['brand'].fillna('UNKNOWN')
df_test['brand'] = df_test['brand'].fillna('UNKNOWN')

df_train['cat_0'] = df_train['cat_0'].fillna('UNKNOWN')
df_valid['cat_0'] = df_valid['cat_0'].fillna('UNKNOWN')
df_test['cat_0'] = df_test['cat_0'].fillna('UNKNOWN')

df_train['cat_1'] = df_train['cat_1'].fillna('UNKNOWN')
df_valid['cat_1'] = df_valid['cat_1'].fillna('UNKNOWN')
df_test['cat_1'] = df_test['cat_1'].fillna('UNKNOWN')

df_train['cat_2'] = df_train['cat_2'].fillna('UNKNOWN')
df_valid['cat_2'] = df_valid['cat_2'].fillna('UNKNOWN')
df_test['cat_2'] = df_test['cat_2'].fillna('UNKNOWN')
```

cuDF does not support date32, right now. We use pandas to transform the timestamp in only date values.

```
[10]: df_train['date'] = cudf.from_pandas(pd.to_datetime(df_train['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,
```

ToDo:

Let's get the price difference of the previous price to the current price per item

2.4 Optimisation

Let's compare a CPU with the GPU version.

```
[14]: def difference_feature(df, offset):
          data_shift = df[['product_id', 'date', 'price']].groupby(['product_id',␣
       ↪'date']).mean().reset_index()
          data_shift.columns = ['product_id', 'date', 'mean']
          data_shift['mean_' + str(offset)] = data_shift['mean'].shift(offset)
          data_shift.loc[data_shift['product_id']!=data_shift['product_id'].
       ↪shift(offset), 'mean_' + str(offset)] = None
          data_shift['diff_' + str(offset)] = data_shift['mean'] - data_shift['mean_'␣
       ↪+ str(offset)]
          data_shift.columns = ['product_id', 'date', 'c1', 'c2', 'price_diff_' +␣
       ↪str(offset)]
          data_shift.drop(['c1', 'c2'], axis=1, inplace=True)
          df = df.merge(data_shift, how='left', on=['product_id', 'date'])
```

```
[15]: df_train_pd = df_train.to_pandas()
```

```
[16]: %%time

      _ = difference_feature(df_train_pd, 1)
```

```
CPU times: user 10.2 s, sys: 4.81 s, total: 15 s
Wall time: 15 s
```

```
[17]: %%time

      _ = difference_feature(df_train, 1)
```

```
CPU times: user 196 ms, sys: 252 ms, total: 448 ms
Wall time: 444 ms
```

In our experiments, we achieved a speedup of 43.1s

We shutdown the kernel.

[1]:
```python
import IPython

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

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