03_1_CombineCategories

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 3. Feature Engineering - Categorical

2.1 3.1. Combining Categories / Cross Columns

2.2 Theory

Combining Categories (CC) is a simple, powerful technique, but often undervalued. We will use this strategy in other feature engineering techniques, as well, and will introduce its value in a simple example. In some datasets, categories by itself provide no information to predict the target. But if we combine multiple categories, together, then we can indentify patterns. For example, we have the following categories:

Weekday

Hour of the day

Each of them independently has no significant pattern in the dataset. If we combine them with *Weekday_HourOfTheDay*, then we can observe some strong behavior for certainn times on the weekend Decision Trees determine the split in the dataset on single features. If each categorical feature by itself does not provide the information gain, then Decision Trees cannot find a good split. If we provide a combined categorical feature, the Decision Tree can easier split the dataset.

Combining categories, also called Cross Column or Cross Product, is used in the Wide Deep Architecture by Google and is implemented in Tensorflow

[2]:
```python
import IPython

import cudf

import pandas as pd
import numpy as np
```

[3]:
```python
f1 = [0]*45 + [1]*45 + [2]*10 + [0]*5 + [1]*5 + [2]*90 + [0]*5 + [1]*5 + [2]*90␣
 ↪+ [0]*45 + [1]*45 + [2]*10
f2 = [0]*45 + [0]*45 + [0]*10 + [1]*5 + [1]*5 + [1]*90 + [0]*5 + [0]*5 + [0]*90␣
 ↪+ [1]*45 + [1]*45 + [1]*10
t = [1]*45 + [1]*45 + [1]*10 + [1]*5 + [1]*5 + [1]*90 + [0]*5 + [0]*5 + [0]*90␣
 ↪+ [0]*45 + [0]*45 + [0]*10

data = cudf.DataFrame({
    'f1': f1,
    'f2': f2,
})

for i in range(3,5):
    data['f' + str(i)] = np.random.choice(list(range(3)), data.shape[0])

data['target'] = t
```

[4]:
```python
data.head()
```

[4]:

	f1	f2	f3	f4	target
0	0	0	0	0	1
1	0	0	1	1	1
2	0	0	0	2	1
3	0	0	0	0	1
4	0	0	1	2	1

We take a look on the features *f1* and *f2*. Each of the feature provides no information gain as each category has a 0.5 probability for the target.

[5]:
```
data.groupby('f1').target.agg(['mean', 'count'])
```

[5]:
```
    mean  count
f1
0    0.5    100
1    0.5    100
2    0.5    200
```

[6]:
```
data.groupby('f2').target.agg(['mean', 'count'])
```

[6]:
```
    mean  count
f2
0    0.5    200
1    0.5    200
```

If we analyze the features *f1* and *f2* together, we can observe a significant pattern in the target variable.

[7]:
```
data.groupby(['f1', 'f2']).target.agg(['mean', 'count'])
```

[7]:
```
       mean  count
f1 f2
0  0    0.9     50
   1    0.1     50
1  0    0.9     50
   1    0.1     50
2  0    0.1    100
   1    0.9    100
```

Next, we train a simple Decision Tree to show how combining categories will support the decision boundaries.

[8]:
```
df = data.to_pandas()
```

[9]:
```
import pydotplus
import sklearn.tree as tree
from IPython.display import Image
```

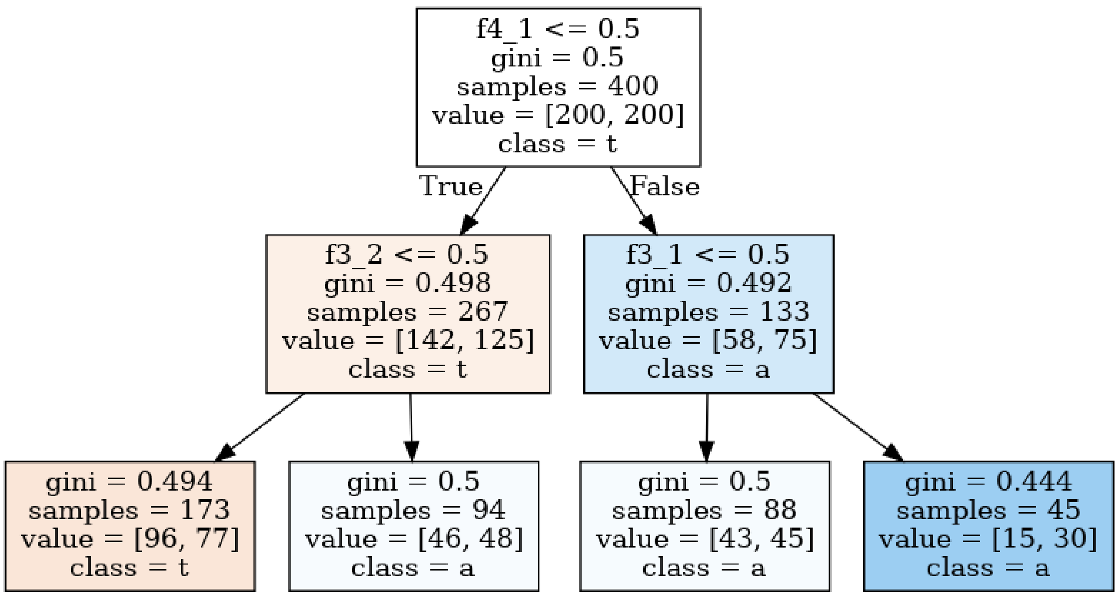
[10]:
```
def get_hotn_features(df):
    out = []
    for col in df.columns:
        if col != 'target':
            out.append(pd.get_dummies(df[col], prefix=col))
    return(pd.concat(out, axis=1))

def viz_tree(df, lf):
    dt_feature_names = list(get_hotn_features(df).columns)
    dt_target_names = 'target'
```

```
    tree.export_graphviz(lf, out_file='tree.dot',
                         feature_names=dt_feature_names,␣
↪class_names=dt_target_names,
                         filled=True)
    graph = pydotplus.graph_from_dot_file('tree.dot')
    return(graph.create_png())
```

First, we train it without the combined categories *f1* and *f2*. We can see, that the Decision Trees creates the split on the random features *f3* and *f4*. The leaves have only a small information gain (e.g. 98 negative vs. 82 positive).

[11]:
```
lf = tree.DecisionTreeClassifier(max_depth=2)
lf.fit(get_hotn_features(df), df[['target']])
Image(viz_tree(df, lf))
```

[11]:

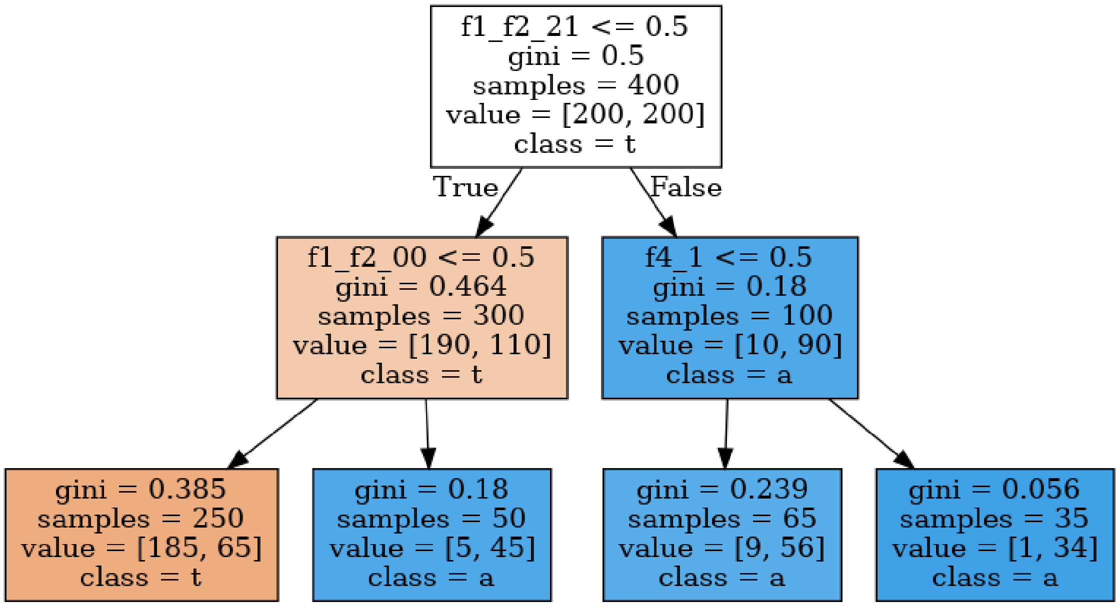


Now, we combine the categories *f1* and *f2* as an additional feature. We can see that the Decision Tree uses that feature first and that the splits have a high information gain. For example, 190 negative vs. 110 positives.

[12]:
```
df['f1_f2'] = df['f1'].astype(str) + df['f2'].astype(str)
```

[13]:
```
lf.fit(get_hotn_features(df), df[['target']])
Image(viz_tree(df, lf))
```

[13]:



This simple technique will be used in combination with other feature engineering techniques. We may have the idea - that is great, let's combine all categories into one feature. Unfortunately, this is not that easy. We want to balance the number of categories used, the number of observations in resulting category values and the information gain:

The more categories we combine, we will identify more underlying patterns - but combining more categories together reduces the number of observation per categoy in the resulting features

Higher number of observation in the resulting category shows a strong pattern and it is more generalizable

High information gain supports our model, but only if it is generalizable

The extreme example is that we combine all features f1, f2, f3 and f4 together. But the observation per category (count) is very small (4-20)

[14]:
```
df.groupby([x for x in df.columns if 'target' not in x and 'f1_f2' not in x]).
↪target.agg(['mean', 'count']).head(10)
```

[14]:
```
                   mean  count
f1 f2 f3 f4
0  0  0  0   1.000000      6
         1   1.000000      5
         2   0.750000      4
      1  0   0.750000      4
         1   0.857143      7
         2   1.000000      5
      2  0   0.750000      4
         1   0.750000      4
```

```
      2   1.000000      11
1  0  0   0.000000       6
```

Best practicse:

Combining low cardinal categories is a good start. For example, the dataset size is 100M rows and there are multiple categories with a caridnality (# of unique values) of 10-50, then combining them should not result in low observation count

Exploratory Data Analysis (EDA) is faster than training a model. Analyzing the information value for different combination of categorical features (on a sample) is really fast.

Example of getting the cardinality for categories:

[15]:
```
df.astype(str).describe()
```

[15]:
```
         f1   f2   f3   f4 target f1_f2
count   400  400  400  400    400   400
unique    3    2    3    3      2     6
top       2    0    1    0      0    21
freq    200  200  136  142    200   100
```

Summary Combining categories identifies underlying patterns in the dataset

The technique can support Decision Trees to create better splits as Decision Trees analyze features independently of each other

2.3 Practice

Now, it is your turn. What are good combinations of categories in our dataset?

ToDo:

Define which categorical featueres should be combined? Why should these be combined? What are your hypotheses?

[16]:
```
import cudf
```

[17]:
```
df_train = cudf.read_parquet('./data/train.parquet')
```

[18]:
```
df_train.head()
```

[18]:
```
                event_time event_type  product_id    brand   price    user_id  \
0  2019-12-01 00:00:28 UTC       cart    17800342     zeta   66.90  550465671
1  2019-12-01 00:00:39 UTC       cart     3701309  polaris   89.32  543733099
2  2019-12-01 00:00:40 UTC       cart     3701309  polaris   89.32  543733099
3  2019-12-01 00:00:41 UTC       cart     3701309  polaris   89.32  543733099
4  2019-12-01 00:01:56 UTC       cart     1004767  samsung  235.60  579970209

                           user_session  target         cat_0        cat_1  \
```

```
0  22650a62-2d9c-4151-9f41-2674ec6d32d5       0     computers      desktop
1  a65116f4-ac53-4a41-ad68-6606788e674c       0    appliances  environment
2  a65116f4-ac53-4a41-ad68-6606788e674c       0    appliances  environment
3  a65116f4-ac53-4a41-ad68-6606788e674c       0    appliances  environment
4  c6946211-ce70-4228-95ce-fd7fccdde63c       0  construction        tools

   cat_2 cat_3            timestamp  ts_hour  ts_minute  ts_weekday  ts_day  \
0   <NA>  <NA>  2019-12-01 00:00:28        0          0           6       1
1 vacuum  <NA>  2019-12-01 00:00:39        0          0           6       1
2 vacuum  <NA>  2019-12-01 00:00:40        0          0           6       1
3 vacuum  <NA>  2019-12-01 00:00:41        0          0           6       1
4  light  <NA>  2019-12-01 00:01:56        0          1           6       1

   ts_month  ts_year
0        12     2019
1        12     2019
2        12     2019
3        12     2019
4        12     2019
```

[19]:
```
###ToDo
def explore_cat(df, cats):
    df_agg = df_train[cats + ['target']].groupby(cats).agg(['mean', 'count']).
 ↪reset_index()
    df_agg.columns = cats + ['mean', 'count']
    print(df_agg.sort_values('count', ascending=False).head(20))

cats = ['product_id', 'user_id']
explore_cat(df_train, cats)
```

```
         product_id    user_id      mean  count
640663      1004767  545442548  0.000000    807
620114      1004767  525325337  0.000000    753
1560100     1005107  553431815  0.599185    736
4882910    15300303  512875426  0.000000    709
2021336     1005174  563599039  0.931238    509
1148644     1004873  515032042  0.002041    490
3626024     4804718  536911254  0.000000    471
4154253     8800045  557590749  0.000000    380
3076839     3601537  578263741  0.000000    363
839819      1004833  564068124  0.793872    359
1477094     1005100  611998200  0.000000    333
6442        1002524  515598234  0.677215    316
348093      1004249  513901034  0.648562    313
1301035     1005008  521558076  0.003236    309
1941657     1005161  512924342  0.537162    296
1593381     1005115  516010934  0.750000    288
```

```
3631591     4804718  576154686  0.550523    287
5935044   100007950  515481166  0.000000    287
3062423     3601489  513824664  0.000000    275
85419       1002544  545376441  0.896296    270
```

[20]:
```
############### Solution ###############
```

[23]:
```
############### Solution End ###########
```

2.4 Optimization

There is not much optimization technique to apply. We will "chain" the idea of combining categories with other Feature Engineering techniques, which does NOT require us to actually combine and store the new feature in the dataset. Instead, we will create features based on the combined categories directly and won't store the combined categories as a separate feature. One advice is to use cuDF instead of pandas. Analzying the dataset requires calculating different groupby combination multiple times by a data scientist. GPU acceleration can significantly speed-up the calculations and enables you to run more comparisons.

[24]:
```
big_df = df_train.to_pandas()
big_data = df_train
```

[25]:
```
print('Pandas Shape:' + str(big_df.shape))
print('cudf Shape:' + str(big_df.shape))
```

```
Pandas Shape:(11461357, 19)
cudf Shape:(11461357, 19)
```

[26]:
```
%%time

big_df.groupby(['cat_0', 'cat_1', 'cat_2', 'cat_3', 'brand']).target.
 ↪agg(['mean', 'count'])
print('')
```

```
CPU times: user 3.02 s, sys: 616 ms, total: 3.63 s
Wall time: 3.63 s
```

[27]:
```
%%time

big_data.groupby(['cat_0', 'cat_1', 'cat_2', 'cat_3', 'brand']).target.
 ↪agg(['mean', 'count'])
print('')
```

```
CPU times: user 812 ms, sys: 800 ms, total: 1.61 s
Wall time: 1.61 s
```

A dataset with 12M rows is ~4-6x faster on GPU with cuDF as on CPU with pandas. This difference can even increase with larger dataset size as the groupby operation is not linear in complexity.

We shutdown the kernel.

[28]:
```
app = IPython.Application.instance()
app.kernel.do_shutdown(False)
```

[28]:
```
{'status': 'ok', 'restart': False}
```