catch_the_fraudster

June 20, 2021

[6]:
```
import pandas as pd
import numpy as np
%load_ext autoreload
```

1 Load data

[7]:
```
train_data = pd.read_csv('data/train_v2.csv')
columns = list(train_data)
N_original, M_original = train_data.shape
columns, N_original
```

[7]:
```
(['id',
  'timestamp',
  'product_id',
  'product_department',
  'product_category',
  'card_id',
  'user_id',
  'C15',
  'C16',
  'C17',
  'C18',
  'C19',
  'C20',
  'C21',
  'amount',
  'isfraud'],
 32369524)
```

2 Data pipeline

- With trees, normalizing features is not necessary

[8]:
```
%autoreload
from sklearn.pipeline import Pipeline, FeatureUnion
#from sklearn.preprocessing import OneHotEncoder
#from include.DatetimeFromTimestamp import DatetimeFromTimestamp
```

```
#from include.HourOfDay import HourOfDay
#from include.DataFrameDropper import DataFrameDropper
#from include.DataFrameSelector import DataFrameSelector
#from include.FilterNMostCommon import FilterNMostCommon
#from include.UserEvaluator import UserEvaluator
from include.ConcatEncoder import ConcatEncoder
#from sklearn.impute import SimpleImputer
from include.ImputedColumn import ImputedColumn

#columns_to_drop = ['id', 'timestamp', 'product_id', 'product_department',␣
↪'product_category', 'card_id', 'user_id']
columns_to_use = ['C15_C16', 'C18_C17', 'C19', 'C20', 'C21', 'amount',␣
↪'cat_dep_id', 'card_user','-1s']
mrf_prod = 1e-5
mrf_card = 1e-5
mrf_C = 5e-4
#the hot encoded attributes are also used
pipeline_normal = Pipeline([
    #('hour_creator', HourOfDay()),
    #('datetime_creator', DatetimeFromTimestamp()),
    #('user_evaluator', UserEvaluator()),
    ('imputed_column', ImputedColumn(missing_value=-1, target_column='C20',␣
↪new_column='-1s')),
    #('C20_imputer', SimpleImputer(missing_values=-1, strategy='constant',␣
↪fill_value=10010)), #10010 because it is value that has a close probability␣
↪of fraud from -1
    ('concat_encoder_product', ConcatEncoder(['product_category',␣
↪'product_department', 'product_id'], attr_name='cat_dep_id',␣
↪min_rel_freq=mrf_prod)),
    ('concat_encoder_card', ConcatEncoder(['card_id', 'user_id'],␣
↪attr_name='card_user', min_rel_freq=mrf_card)),
    ('concat_encoder_C15_C16', ConcatEncoder(['C15', 'C16'],␣
↪attr_name='C15_C16', min_rel_freq=mrf_C)),
    ('concat_encoder_C18_C17', ConcatEncoder(['C18', 'C17'],␣
↪attr_name='C18_C17', min_rel_freq=mrf_C)),
    ('encoder_C19', ConcatEncoder(['C19'], attr_name='C19',␣
↪min_rel_freq=mrf_C)),
    ('encoder_C20', ConcatEncoder(['C20'], attr_name='C20',␣
↪min_rel_freq=mrf_C)),
    ('encoder_C21', ConcatEncoder(['C21'], attr_name='C21',␣
↪min_rel_freq=mrf_C)),
    ('dataframe_selector', DataFrameSelector(attribute_names=columns_to_use)),
])

#pipeline_1hot = Pipeline([
    #('dataframe_selector', DataFrameSelector(['product_category'])),
```

```
    #('filter_n_most_common', FilterNMostCommon(N=5,␣
↪attribute_name='product_category', minRelFreq=0.05)),
    #('1hot_encoder', OneHotEncoder(sparse = False))
#])

#pipeline_full = FeatureUnion(transformer_list=[
    #('pipeline_normal', pipeline_normal),
    #('pipeline_1hot', pipeline_1hot),
#])

random_seed = 42
```

[37]:
```
#from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier

current_model = DecisionTreeClassifier
```

3 Train/Test split

- Normally I would use random sampling, stratified by an attribute of major relevance, however, in this case the test data that was given follows the train data in time. Therefore, in order to do local testing my first guess would be that it is better to remake that scenario and sample the data by simply splitting it sorted as it is, by time.
- Cross validation is not necessary given that we have a test set big enough

[35]:
```
split_by = 2
N_train = 6000000
N_test = 6000000

start_at = N_original - N_train - N_test
split_at = start_at + N_train

train_X = pd.DataFrame(train_data.iloc[start_at:split_at,:-1])
train_Y = train_data.iloc[start_at:split_at,-1]
test_X = pd.DataFrame(train_data.iloc[split_at:,:-1])
test_Y = train_data.iloc[split_at:,-1]

train_X.shape, train_Y.shape, test_X.shape, test_Y.shape
```

[35]:
```
((6000000, 15), (6000000,), (6000000, 15), (6000000,))
```

4 Test

[49]:
```
#pipeline_normal.fit(train_data, train_data['isfraud'])
train_X_treated = pipeline_normal.transform(train_X)
test_X_treated = pipeline_normal.transform(test_X)
```

```
train_X_treated.shape, test_X_treated.shape
```

[49]: ((6000000, 9), (6000000, 9))

[60]:
```
from sklearn.metrics import roc_auc_score

crit='gini'
h=16
leaf=1/(4**4)
split=1/(4**9)
model = current_model(random_state=random_seed, criterion=crit, max_depth=h,␣
↪min_samples_leaf=leaf, min_samples_split=split)
model.fit(train_X_treated, train_Y)
test_pred_prob = model.predict_proba(test_X_treated)[:,1]

roc_auc_score(test_Y, test_pred_prob)
```

[60]: 0.6929880743149611

5 Submit

1. 0.6654 - 1 hot encoding of `product_category`
2. 0.6264 - `hour` actualy decreases score. It will be removed for now, however it might be useful while combined with other attributes.

…

7. A lot of attempts which didn't increase the score with two features I created: `daily_transactions_ratio` and `daily_amount_ratio`, which represented the ratio between the number of transactions/the amount made by that user/card on that day with the average daily number of transactions/amount from that specific user.
8. 0.6594 - Decision tree with label encoding of product stuff
9. 0.6760 - also with label encoding of card and user stuff
10. 0.6849 - Com product min rel freq a 1e-5
11. 0.72364 - Otimized tree with: `crit='gini', h=16, leaf=1/(4**4), split=1/(4**9)`

5.0.1 Load submit data

[9]:
```
submit_data = pd.read_csv('data/test_v2.csv')
```

5.0.2 Prepare train and submit data

[10]:
```
train_data_X = train_data.iloc[:,:-1]
train_data_Y = train_data.iloc[:,-1]
train_data_X.shape, train_data_Y.shape
```

[10]: ((32369524, 15), (32369524,))

[11]:
```
pipeline_normal.fit(train_data, train_data['isfraud'])
train_data_X_treated = pipeline_normal.transform(train_data_X)
del train_data_X
train_data_X_treated.shape
```

[11]:
```
(32369524, 9)
```

[30]:
```
submit_data_treated = pipeline_normal.transform(submit_data)
```

5.0.3 Train model & predict

[38]:
```
crit='gini'
h=16
leaf=1/(4**4)
split=1/(4**9)
model = current_model(random_state=random_seed, criterion=crit, max_depth=h,␣
↪min_samples_leaf=leaf, min_samples_split=split)
model.fit(train_data_X_treated, train_data_Y)
```

[38]:
```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                       max_depth=16, max_features='auto', max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=0.00390625,
                       min_samples_split=3.814697265625e-06,
                       min_weight_fraction_leaf=0.0, n_estimators=20, n_jobs=None,
                       oob_score=False, random_state=42, verbose=0, warm_start=False)
```

[39]:
```
pred_prob = model.predict_proba(submit_data_treated)[:,1]
```

[40]:
```
submission = pd.DataFrame()
submission['id'] = submit_data['id']
submission['isfraud'] = pred_prob
submission.head()
```

[40]:

	id	isfraud
0	32263877	0.019840
1	32263886	0.100284
2	32263890	0.306197
3	32263895	0.306197
4	32263896	0.065333

[41]:
```
submission.to_csv(path_or_buf = 'data/submit.csv', index = False)
```

6 Visualize the Tree

[36]:
```
print('Node count:', model.tree_.node_count)
```

```
Node count: 301
```

7 Optimizing the model

[15]:
```
n_train = 8000000
n_test =  8000000
start_at=n_train+n_test
train_X_opt = train_data_X_treated.iloc[-start_at:-n_test,:]
train_Y_opt = train_data_Y[-start_at:-n_test]
test_X_opt = train_data_X_treated.iloc[-n_test:,:]
test_Y_opt = train_data_Y[-n_test:]
train_X_opt.shape, train_Y_opt.shape, test_X_opt.shape, test_Y_opt.shape
```

[15]:
```
((8000000, 9), (8000000,), (8000000, 9), (8000000,))
```

[23]:
```
criteria=['gini','entropy']
s1 = [1/(4**idx) for idx in range(8, 1, -1)]
s2 = [1/(4**idx) for idx in range(9, 1, -1)]
height = [2**idx for idx in range(5,2,-1)]
s1, s2, height,1/(4**9)
```

[23]:
```
([1.52587890625e-05,
  6.103515625e-05,
  0.000244140625,
  0.0009765625,
  0.00390625,
  0.015625,
  0.0625],
 [3.814697265625e-06,
  1.52587890625e-05,
  6.103515625e-05,
  0.000244140625,
  0.0009765625,
  0.00390625,
  0.015625,
  0.0625],
 [32, 16, 8],
 3.814697265625e-06)
```

1. 0.743 - gini; h=16; leaf=^4; split=^9
2. 0.741 - entropy; h=16; leaf=^4; split=^9
3. 0.743 - gini; h=16; leaf=^4; split=^10

[27]:

```python
crit='gini'
h=16
leaf=1/(4**4)
split=1/(4**9)
model = current_model(random_state=random_seed, criterion=crit, max_depth=h,␣
↪min_samples_leaf=leaf, min_samples_split=split)
model.fit(train_X_opt, train_Y_opt)
test_pred_prob = model.predict_proba(test_X_opt)[:,1]

score = roc_auc_score(test_Y_opt, test_pred_prob)
score
```

[27]: 0.7430119254143245

[ ]:

```python
from sklearn.metrics import roc_auc_score

best_score=0

for crit in criteria:
    for split in s2:
        for leaf in s1:
            for h in height:
                model = current_model(random_state=random_seed, criterion=crit,␣
↪max_depth=h, min_samples_leaf=leaf, min_samples_split=split)
                model.fit(train_X_opt, train_Y_opt)
                test_pred_prob = model.predict_proba(test_X_opt)[:,1]

                score = roc_auc_score(test_Y_opt, test_pred_prob)
                if score > best_score:
                    print('---------')
                    print('New best score: ', score, ' with: ')
                    best_crit = crit
                    best_split = split
                    best_leaf = leaf
                    best_h = h
                    print('---------')
                    best_score = score
                print(crit)
                print(split)
                print(leaf)
                print(h, '\n')
```