original_experiment

March 5, 2021

[2]:
```
import numpy as np
import pandas as pd
import os
import warnings
from tqdm import tqdm
import math
import seaborn as sns

from scipy.stats import gamma
import datetime as dt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
import lightgbm as lgb
# import optperfprofpy

import sys
sys.path.append('..')
from utils.demand_pkg import *
import matplotlib.pyplot as plt
from utils.algorithms import get_opt, get_EQ, get_end2end_iid, get_end2end,␣
↪get_normal_basestock, get_gamma_basestock, get_normal_basestock2
from subprocess import call

import tensorflow as tf

warnings.filterwarnings('ignore')
```

0.0.1 1. Clean up the dataset. (o0 is grouped by SKU#DC id and sorted by date)

[3]:
```
with open('../data/1320_feature/df_e2e.pkl', 'rb') as fp:
    o0 = pickle.load(fp)
df_sales = pd.read_csv('../data/1320/rdc_sales_1320_replenishment_V1_filled_pp.
↪csv')
df_sl = df_sales.set_index('row')
df_sl.rename(columns=lambda x: (dt.datetime(2016,1,1) + dt.
↪timedelta(days=int(x)-730)).date(), inplace=True)
```

```
[4]: o0 = o0[o0.next_complete_dt < dt.datetime(2018,8,31)]
o0.groupby('item_sku_id').pur_bill_id.count().sort_values(ascending=False)[2000:
 ↪2001]
#o0.columns.tolist()
o0.shape
```

```
[4]: (82138, 157)
```

0.0.2 2. Define all the features

```
[11]: IDX = ['item_sku_id','sku_id']
TIME = ['create_tm', 'complete_dt', 'next_complete_dt', 'actual_pur_qtty',␣
 ↪'pur_bill_id', 'vlt_actual']
CAT_FEA = [
    'item_third_cate_cd',
    'int_org_num',
    ]

VLT_FEA = [
    'uprc', 'contract_stk_prc',
    'wt', 'width', 'height', 'calc_volume', 'len',
    'vlt_count', 'vlt_sum', 'vlt_min', 'vlt_max', 'vlt_mean', 'vlt_std',
    'qtty_sum', 'qtty_min', 'qtty_max', 'qtty_mean', 'qtty_std',
    'amount_sum', 'amount_min', 'amount_max', 'amount_mean', 'amount_std',
    'vlt_count_6mo', 'vlt_sum_6mo', 'vlt_min_6mo', 'vlt_max_6mo',␣
 ↪'vlt_mean_6mo', 'vlt_std_6mo',
    'vendor_vlt_count', 'vendor_vlt_sum', 'vendor_vlt_min', 'vendor_vlt_max',␣
 ↪'vendor_vlt_mean', 'vendor_vlt_std',
    'vendor_vlt_count_6mo', 'vendor_vlt_sum_6mo', 'vendor_vlt_min_6mo',
    'vendor_vlt_max_6mo', 'vendor_vlt_mean_6mo', 'vendor_vlt_std_6mo',
    'vendor_qtty_sum', 'vendor_qtty_min', 'vendor_qtty_max',
    'vendor_qtty_mean', 'vendor_qtty_std', 'vendor_amount_sum',
    'vendor_amount_min', 'vendor_amount_max', 'vendor_amount_mean'
        ]

SF_FEA = [
        'q_7', 'q_14', 'q_28', 'q_56', 'q_140',
        'mean_3', 'mean_7', 'mean_14', 'mean_28', 'mean_56',  'mean_140',
        'diff_140_mean', 'mean_140_decay', 'median_140', 'min_140', 'max_140',␣
 ↪'std_140',
        'diff_60_mean', 'mean_60_decay', 'median_60', 'min_60', 'max_60',␣
 ↪'std_60',
        'diff_30_mean', 'mean_30_decay', 'median_30', 'min_30', 'max_30',␣
 ↪'std_30',
        'diff_14_mean', 'mean_14_decay', 'median_14', 'min_14', 'max_14',␣
 ↪'std_14',
```

```
        'diff_7_mean', 'mean_7_decay', 'median_7', 'min_7', 'max_7', 'std_7',
        'diff_3_mean', 'mean_3_decay', 'median_3', 'min_3', 'max_3', 'std_3',
        'has_sales_days_in_last_140', 'last_has_sales_day_in_last_140',
        'first_has_sales_day_in_last_140', 'has_sales_days_in_last_60',
        'last_has_sales_day_in_last_60', 'first_has_sales_day_in_last_60',
        'has_sales_days_in_last_30', 'last_has_sales_day_in_last_30',
        'first_has_sales_day_in_last_30', 'has_sales_days_in_last_14',
        'last_has_sales_day_in_last_14', 'first_has_sales_day_in_last_14',
        'has_sales_days_in_last_7', 'last_has_sales_day_in_last_7',␣
 ↪'first_has_sales_day_in_last_7'
            ]

MORE_FEA =[
           'review_period',
           'normal',
           'gamma',
            'eq'
            ]

IS_FEA = [
           'initial_stock',
        ]

CAT_FEA_HOT = ['item_third_cate_cd_1591',
             'item_third_cate_cd_2677',
             'item_third_cate_cd_5022',
             'item_third_cate_cd_5024',
             'int_org_num_3',
             'int_org_num_4',
             'int_org_num_5',
             'int_org_num_6',
             'int_org_num_9',
             'int_org_num_10',
             'int_org_num_316',
             'int_org_num_772']

SEQ2SEQ = ['Enc_X', 'Enc_y', 'Dec_X', 'Dec_y']

LABEL = ['demand_RV']
LABEL_vlt = ['vlt_actual']
LABEL_sf = ['label_sf']
```

[12]:

```
SCALE_FEA =  VLT_FEA + SF_FEA + MORE_FEA + IS_FEA + CAT_FEA_HOT + LABEL_vlt +␣
 ↪LABEL_sf
CUT_FEA = VLT_FEA + SF_FEA + MORE_FEA
MODEL_FEA = VLT_FEA + SF_FEA + MORE_FEA + IS_FEA + CAT_FEA_HOT
```

```
MODEL_FEA2 =  IDX + TIME + VLT_FEA + SF_FEA + MORE_FEA + IS_FEA + CAT_FEA_HOT +␣
↪LABEL
len(SCALE_FEA)
```

[12]: 131

0.0.3 3. Training and test dataset splitting

[13]:
```
sku_set = o0.sku_id.unique()
sku_train, sku_test = train_test_split(sku_set, random_state=12, train_size=0.
↪9, test_size=0.1)
df_train = o0[o0['create_tm'] < dt.datetime(2018,7,27)]
df_test = o0[o0['create_tm'] >= dt.datetime(2018,8,1)]
```

[14]:
```
df_train.shape, df_test.shape
```

[14]: ((45996, 157), (28048, 157))

[15]:
```
X_train_ns, y_train_ns, id_train = df_train[SCALE_FEA], df_train[LABEL],␣
↪df_train[IDX]
X_test_ns, y_test_ns, id_test = df_test[SCALE_FEA], df_test[LABEL], df_test[IDX]
n_train, n_test = len(X_train_ns), len(X_test_ns)
```

[16]:
```
print(X_train_ns.mean_140.mean(), X_test_ns.mean_140.mean())
```

21.414720472587987 29.11260036366226

[17]:
```
X_scaler = MinMaxScaler() # For normalizing dataset
y_scaler = MinMaxScaler() # For normalizing dataset

# normalize the training and test dataset
X_train = pd.DataFrame(X_scaler.fit_transform(X_train_ns), columns=X_train_ns.
↪columns)
y_train = pd.DataFrame(y_scaler.fit_transform(y_train_ns), columns=y_train_ns.
↪columns)
X_test = pd.DataFrame(X_scaler.transform(X_test_ns), columns=X_test_ns.columns)
y_test = pd.DataFrame(y_scaler.transform(y_test_ns), columns=y_test_ns.columns)
```

0.0.4 Load the GBM model

[276]:
```
import lightgbm as lgb
import pandas as pd
from sklearn.metrics import mean_squared_error

# create dataset for lightgbm
lgb_train = lgb.Dataset(X_train, y_train)
```

```python
lgb_eval = lgb.Dataset(X_test, y_test, reference=lgb_train)

# specify your configurations as a dict
params = {
    'boosting_type': 'gbdt',
    'objective': 'regression',
    'metric': {'l2', 'l1'},
    'num_leaves':80,
    'learning_rate': 0.1,
    'feature_fraction': 0.8,
    'bagging_fraction': 0.9,
    'bagging_freq': 4,
    'verbose': 0
}

print('Starting training...')
# train
gbm = lgb.train(params,
                lgb_train,
                num_boost_round=25,
                valid_sets=lgb_eval,
                early_stopping_rounds=5)

print('Saving model...')
# # save model to file
# gbm.save_model('model.txt')

print('Loading saved model...')
gbm = lgb.Booster(model_file='model.txt')

print('Starting predicting...')
# predict
y_pred = gbm.predict(X_test, num_iteration=gbm.best_iteration)
# eval
print('The rmse of prediction is:', mean_squared_error(y_test, y_pred) ** 0.5)
```

```
Starting training…
[1]	valid_0's l2: 0.00296156	valid_0's l1: 0.0285396
Training until validation scores don't improve for 5 rounds.
[2]	valid_0's l2: 0.00245583	valid_0's l1: 0.0258298
[3]	valid_0's l2: 0.0020621	valid_0's l1: 0.0234654
[4]	valid_0's l2: 0.00174123	valid_0's l1: 0.0213629
[5]	valid_0's l2: 0.00148049	valid_0's l1: 0.0195269
[6]	valid_0's l2: 0.00127711	valid_0's l1: 0.0178965
[7]	valid_0's l2: 0.00111158	valid_0's l1: 0.0165211
[8]	valid_0's l2: 0.000980538	valid_0's l1: 0.0153096
[9]	valid_0's l2: 0.000869485	valid_0's l1: 0.014249
```

```
[10]	valid_0's l2: 0.000780219	valid_0's l1: 0.0133396
[11]	valid_0's l2: 0.000706021	valid_0's l1: 0.0125497
[12]	valid_0's l2: 0.00065045	valid_0's l1: 0.0119101
[13]	valid_0's l2: 0.000605	valid_0's l1: 0.0113649
[14]	valid_0's l2: 0.000566234	valid_0's l1: 0.0108764
[15]	valid_0's l2: 0.000536814	valid_0's l1: 0.0104635
[16]	valid_0's l2: 0.000512268	valid_0's l1: 0.0101158
[17]	valid_0's l2: 0.000492898	valid_0's l1: 0.0098247
[18]	valid_0's l2: 0.000477946	valid_0's l1: 0.00958028
[19]	valid_0's l2: 0.000465866	valid_0's l1: 0.00937818
[20]	valid_0's l2: 0.000455669	valid_0's l1: 0.00920544
[21]	valid_0's l2: 0.000448287	valid_0's l1: 0.00905265
[22]	valid_0's l2: 0.000441849	valid_0's l1: 0.00891983
[23]	valid_0's l2: 0.000437441	valid_0's l1: 0.00880532
[24]	valid_0's l2: 0.000434243	valid_0's l1: 0.00871172
[25]	valid_0's l2: 0.000430439	valid_0's l1: 0.00862553
Did not meet early stopping. Best iteration is:
[25]	valid_0's l2: 0.000430439	valid_0's l1: 0.00862553
Saving model…
Starting predicting…
The rmse of prediction is: 0.020747027070404226
```

[277]:
```
# store the scaler for transformting the data back
pd_scaler = pd.concat([pd.DataFrame([y_scaler.data_min_,y_scaler.scale_],
 ↪columns=y_train_ns.columns),
            pd.DataFrame([X_scaler.data_min_,X_scaler.scale_],
 ↪columns=X_train_ns.columns)], axis=1)
pd_scaler.to_csv('../data/1320_feature/scaler.csv', index=False)
pd_scaler = pd.read_csv('../data/1320_feature/scaler.csv')
```

[278]:
```
X_train = X_train[MODEL_FEA]
X_test = X_test[MODEL_FEA]
```

[279]:
```
out_gbm = y_pred / pd_scaler.loc[1, LABEL[0]] + pd_scaler.loc[0, LABEL[0]]
```

0.0.5 4. Load models

[280]:
```
pred_path = '../logs/torch/pred_v5_new.csv'
pred_v5 = pd.read_csv(pred_path)
pred_path = '../logs/torch3/pred_v6.csv'
pred_v6 = pd.read_csv(pred_path)
sf_rnn = pickle.load(open('../logs/torch3/pred_E2E_SF_RNN.pkl', 'rb'))
vlt_rnn = pickle.load(open('../logs/torch3/pred_E2E_VLT_RNN.pkl', 'rb'))
```

[281]:
```
## o4 is the test dataset
o4 = df_test.copy()
o4.reset_index(drop=True, inplace=True)
```

```
## add the actual daily sale for the test dataset
o4['demand_RV_list'] = o4.apply(lambda x: df_sl.loc[x['item_sku_id'], \
                                    x['create_tm'].date():
 ↪x['next_complete_dt'].date()].values\
                          if x['item_sku_id'] in df_sl.index else [], axis=1)

## add the actual cumulative sale for the test dataset
o4['demand_RV_list_acm'] = o4['demand_RV_list'].apply(lambda x: np.cumsum(x))
```

E2E model

[282]:
```
o4_ = pd.concat([o4, pred_v5['E2E_MLP_pred'], pred_v6['E2E_RNN_pred']], axis=1)
```

[283]:
```
o4_['gbm_pred'] = out_gbm
```

Optimal order quantity

[284]:
```
o4_['OPT_pred'] = o4_[LABEL]
```

Normal benchmark

[285]:
```
Z90 = 1.64
o4_['Normal_pred'] = o4.apply(lambda x:␣
 ↪int(x['mean_140']*(x['review_period']+x['vendor_vlt_mean'])
                                   +Z90*np.
 ↪sqrt((x['review_period']+x['vendor_vlt_mean'])*x['std_140']**2
                                            + x['std_140']**2 *␣
 ↪x['vendor_vlt_std'])), axis=1)
```

Gamma benchmark

[286]:
```
def gamma_base(x):
    mean = x['mean_140']
    var = x['std_140']**2
    theta = var/(mean+1e-4)
    k = mean/(theta+1e-4)
    k_sum = int(x['review_period']+x['vendor_vlt_mean'])*k
    gamma_stock = gamma.ppf(0.9, a=k_sum, scale = theta)
    if(np.isnan(gamma_stock)):
        return 0
    else:
        return int(gamma_stock)
o4_['Gamma_pred'] = o4.apply(gamma_base, axis=1)
```

PTO benchmark1 and benchmark 2

[287]:
```
o4_['Bm1_pred'] = np.mean(sf_rnn[:,:,5], axis=1)
o4_['Bm1_pred'] = o4_['Bm1_pred'] * (o4['review_period'] + o4['vlt_actual']).
↪astype(int)
```

[288]:
```
b = 9
h = 1
def get_bm2(x):
    rl = x['review_period'] + x['vlt_actual']
    if rl <= b:
        days = int(rl)
    else:
        days = int(rl) - rl//(b+h)
    return x['Bm2_pred'] * days

o4_['Bm2_pred'] = np.mean(sf_rnn[:,:,5], axis=1)
o4_['Bm2_pred'] = o4_.apply(get_bm2, axis=1)
```

[289]:
```
o4_['vlt_forecast'] = vlt_rnn
```

[290]:
```
o4g = o4_.groupby('item_sku_id').agg(lambda x: x.tolist())
```

[291]:
```
o4g['Demand_agg_list'] = o4g.apply(lambda x: df_sl.loc[x.name, \
                                            x['create_tm'][0].date():
↪x['next_complete_dt'][-1].date()].values\
                                            , axis=1)
```

0.0.6 5. Sequential test model

[292]:
```
def get_agginv(x, name):
    inv1, inv2 = [x['initial_stock'][0]], []
    rd = len(x['pur_bill_id'])

    for r in range(rd):
        if r < rd - 1:
            len_day = len(x['demand_RV_list'][r])-1
        else:
            len_day = len(x['demand_RV_list'][r])
        for t in range(len_day):
            if t == 0:
                if r == 0:
                    replen = int(round(x[name+'_pred'][r] - inv1[0]))
                else:
                    try:
                        replen = int(round(x[name+'_pred'][r] -␣
↪inv1[-int(round(x['vlt_actual'][r]))-1]))
                    except:
```

```
                        replen = int(round(x[name+'_pred'][r] - inv1[1]))
            if t < int(round(x['vlt_actual'][r])):
                if r == 0:
                    inv1.append(inv1[-1] - x['demand_RV_list'][r][t])
            elif t == int(round(x['vlt_actual'][r])):
                if inv1[-1] >= 0:
                    inv_ = inv1[-1] + replen - x['demand_RV_list'][r][t]
                else:
                    inv_ = replen - x['demand_RV_list'][r][t]
                inv1.append(inv_)
                inv2.append(inv_)
            else:
                inv_ = inv1[-1] - x['demand_RV_list'][r][t]
                inv1.append(inv_)
                inv2.append(inv_)

    inv1 = inv1[1:]
    return [inv1, inv2]
```

[293]:
```
o4g['OPT_agginv_f'], o4g['OPT_agginv'] = zip(*o4g.apply(get_agginv, name='OPT',␣
↪ axis=1))
```

[294]:
```
o4g['E2E_MLP_agginv_f'], o4g['E2E_MLP_agginv'] = zip(*o4g.apply(get_agginv,␣
↪name='E2E_MLP',  axis=1))
```

[295]:
```
o4g['E2E_RNN_agginv_f'], o4g['E2E_RNN_agginv'] = zip(*o4g.apply(get_agginv,␣
↪name='E2E_RNN',  axis=1))
```

[296]:
```
o4g['Normal_agginv_f'], o4g['Normal_agginv'] = zip(*o4g.apply(get_agginv,␣
↪name='Normal',  axis=1))
```

[297]:
```
o4g['Gamma_agginv_f'], o4g['Gamma_agginv'] = zip(*o4g.apply(get_agginv,␣
↪name='Gamma',  axis=1))
```

[298]:
```
o4g['gbm_agginv_f'], o4g['gbm_agginv'] = zip(*o4g.apply(get_agginv, name='gbm',␣
↪ axis=1))
```

[299]:
```
o4g['Bm1_agginv_f'], o4g['Bm1_agginv'] = zip(*o4g.apply(get_agginv, name='Bm1',␣
↪ axis=1))
o4g['Bm2_agginv_f'], o4g['Bm2_agginv'] = zip(*o4g.apply(get_agginv, name='Bm2',␣
↪ axis=1))
```

0.0.7 6. Calculate cost

[302]:
```
list_c = ['SKU_DC', 'OPT',
          'E2E_RNN',
          'E2E_GBM',
          'Normal', 'Gamma', 'Bm2', 'Bm1',
          'Ave_sales','Std_sales',
         ]


h = 1
b = 9
numberOfRows = len(o4g)
df_cost_agg = pd.DataFrame(index=np.arange(0, numberOfRows), columns=list_c)
df_holding_agg = pd.DataFrame(index=np.arange(0, numberOfRows), columns=list_c)
df_back_agg = pd.DataFrame(index=np.arange(0, numberOfRows), columns=list_c)
df_stockout_agg = pd.DataFrame(index=np.arange(0, numberOfRows), columns=list_c)
df_turnover_agg = pd.DataFrame(index=np.arange(0, numberOfRows), columns=list_c)

df_cost_agg['SKU_DC']=df_holding_agg['SKU_DC']=df_back_agg['SKU_DC']=df_stockout_agg['SKU_DC']
            =df_turnover_agg['SKU_DC']=o4g.index.values
df_cost_agg['Ave_sales']=df_holding_agg['Ave_sales']=df_back_agg['Ave_sales']=df_stockout_agg[
            =df_turnover_agg['Ave_sales']=o4g['mean_140'].apply(lambda x:x[0]).
↪values
df_cost_agg['Std_sales']=df_holding_agg['Std_sales']=df_back_agg['Std_sales']=df_stockout_agg[
            =df_turnover_agg['Std_sales']=o4g['std_140'].apply(lambda x:x[0]).
↪values
```

[303]:
```
str_list = ['OPT',
            'E2E_RNN',
            'gbm',
            'Normal', 'Gamma',
            'Bm2', 'Bm1']

o4g_ = o4g.reset_index(drop=True)
for str1 in str_list:
    str2 = str1 + '_agginv'
    df_holding_agg[str1] = o4g_[str2].apply(lambda x: h * sum([inv for inv in x␣
↪if inv>0]) )
    df_back_agg[str1] = o4g_[str2].apply(lambda x: b * -sum([inv for inv in x␣
↪if inv<0]) )
    df_stockout_agg[str1] = o4g_[str2].apply(lambda x: len([inv for inv in x if␣
↪inv<=0])/len(x) if len(x)>0 else 0 )
    df_turnover_agg[str1] = o4g_.apply(lambda x: np.mean([max(i,0) for i in␣
↪x[str2]]) / x['mean_28'][0]
                                  if np.mean(x['mean_28'][0]) >0 else np.
↪mean(x[str2]), axis=1).fillna(7)
    df_cost_agg[str1] = df_holding_agg[str1] + df_back_agg[str1]
```

[304]:
```python
df_aggcom = pd.DataFrame({'Total cost': df_cost_agg[str_list].mean(),
             'Holding cost': df_holding_agg[str_list].mean(),
             'Stockout cost': df_back_agg[str_list].mean(),
             'Stockout rate': df_stockout_agg[str_list].mean(),
             'Turnover rate': df_turnover_agg[str_list].multiply(np.
 ↪sqrt(df_turnover_agg['Ave_sales']), axis="index").mean()/np.
 ↪sqrt(df_turnover_agg['Ave_sales']).mean()*1.2
             }).T
df_aggcom
```

[304]:
```
                       OPT       E2E_RNN           gbm        Normal  \
Total cost     3022.602661   3766.520059   4017.820936   4576.049171
Holding cost   1925.696476   2689.021730   2109.932511   3369.207968
Stockout cost  1096.906185   1077.498328   1907.888425   1206.841204
Stockout rate     0.095281      0.091799      0.071810      0.065366
Turnover rate     9.183903     13.457294     13.703757     18.810001

                     Gamma           Bm2           Bm1
Total cost     4476.170706   4157.030506   4207.088313
Holding cost   2821.866277   2254.996657   2502.545480
Stockout cost  1654.304430   1902.033849   1704.542833
Stockout rate     0.097331      0.110435      0.093309
Turnover rate    15.515257     11.368371     12.761443
```

[305]:
```python
print(df_aggcom.to_latex(float_format=lambda x: '%.3f' % x))
```

```
\begin{tabular}{lrrrrrrr}
\toprule
{} &      OPT &  E2E\_RNN &      gbm &   Normal &    Gamma &      Bm2 &      Bm1
\\
\midrule
Total cost    & 3022.603 & 3766.520 & 4017.821 & 4576.049 & 4476.171 & 4157.031
& 4207.088 \\
Holding cost  & 1925.696 & 2689.022 & 2109.933 & 3369.208 & 2821.866 & 2254.997
& 2502.545 \\
Stockout cost & 1096.906 & 1077.498 & 1907.888 & 1206.841 & 1654.304 & 1902.034
& 1704.543 \\
Stockout rate &    0.095 &    0.092 &    0.072 &    0.065 &    0.097 &    0.110
&    0.093 \\
Turnover rate &    9.184 &   13.457 &   13.704 &   18.810 &   15.515 &   11.368
&   12.761 \\
\bottomrule
\end{tabular}
```