3.Feature Engineering

101]:

```
#shift the time since it is a forecast problem
train_monthly['item_cnt_month'] = train_monthly.sort_values('date_block_num').groupby(['shop_id', 'item_id'])['item_cnt'].shift(-1)
train_monthly
```

[10…

	date_block_num	shop_id	item_id	item_category_id	item_price	mean_item_price	item_cnt	mean_item_cnt	transactions	year	month	item_cnt_month
0	0	2	5572	2.0	10730.00	1532.857143	9.0	1.285714	7.0	2013	0	1.0
1	0	2	5643	2.0	4775.21	2387.605000	0.0	0.000000	2.0	2013	0	0.0
2	0	2	5583	5.0	1188.30	594.150000	2.0	1.000000	2.0	2013	0	1.0
3	0	2	7893	6.0	5970.00	1990.000000	3.0	1.000000	3.0	2013	0	2.0
4	0	2	7894	6.0	1490.00	1490.000000	1.0	1.000000	1.0	2013	0	2.0
...	...	...	...	...	...	...	...	...	...	...	...	...
6734443	33	36	9103	0.0	0.00	0.000000	0.0	0.000000	0.0	2015	9	NaN
6734444	33	36	9107	0.0	0.00	0.000000	0.0	0.000000	0.0	2015	9	NaN
6734445	33	36	5704	0.0	0.00	0.000000	0.0	0.000000	0.0	2015	9	NaN
6734446	33	36	12733	0.0	0.00	0.000000	0.0	0.000000	0.0	2015	9	NaN
6734447	33	36	15925	0.0	0.00	0.000000	0.0	0.000000	0.0	2015	9	NaN

6732967 rows × 12 columns

+ Code + Markdown

[102]:

```
#unify the price mark
train_monthly['item_price_unit'] = train_monthly['item_price'] // train_monthly['item_cnt']
train_monthly['item_price_unit'].fillna(0, inplace=True)
```

[103]:

```
#grouping data with item_id
gp_item_price = train_monthly.sort_values('date_block_num').groupby(['item_id'], as_index=False).agg({'item_price':[np.min, np.max]})
gp_item_price.columns = ['item_id', 'hist_min_item_price', 'hist_max_item_price']

train_monthly = pd.merge(train_monthly, gp_item_price, on='item_id', how='left')
```

[104]:

```
train_monthly['price_increase'] = train_monthly['item_price'] - train_monthly['hist_min_item_price']
train_monthly['price_decrease'] = train_monthly['hist_max_item_price'] - train_monthly['item_price']
```

Rolling Window

1. Creating a rolling window with a specified size and perform calculations on the data
2. A time series problem that recent lag values are more predictive than older lag values
3. rolling() functions->perform rolling window functions.Regard window size as a parameter to group values.

```
#make rolling functions' parameters with lambda function
#Min value
f_min = lambda x: x.rolling(window=3, min_periods=1).min()
#Max value
f_max = lambda x: x.rolling(window=3, min_periods=1).max()
#Mean value
f_mean = lambda x: x.rolling(window=3, min_periods=1).mean()
#Standard deviation
f_std = lambda x: x.rolling(window=3, min_periods=1).std()

function_list = [f_min, f_max, f_mean, f_std]
function_name = ['min', 'max', 'mean', 'std']

for i in range(len(function_list)):
    train_monthly[('item_cnt_%s' % function_name[i])] = train_monthly.sort_values('date_block_num').groupby(['shop_id', 'item_category_id', 'item_id
#Fill the empty std features with 0
train_monthly['item_cnt_std'].fillna(0, inplace=True)
```

+ Code + Markdown

Lag Based Feature

1. A lag features is a variable which contains data from **prior time steps**.
2. Lag features are a classical way for transforming a time series forecasting problem into a **supervised learning problem**.
3. Let's say you are predicting the stock price for a company. So, the previous day's stock price is vital in making predictions right? So, this means that the value at time t is greatly affected by the value at time t-1. The past values are known as lags, so **t-1 is lag 1, t-2 is lag 2,** and so on.

[106]:
```python
#create the lag lists for lag1, lag2, lag3 with prior time data
lag_list = [1, 2, 3]
#name the lag list features
for lag in lag_list:
    ft_name = ('item_cnt_shifted%s' % lag)
    train_monthly[ft_name] = train_monthly.sort_values('date_block_num').groupby(['shop_id', 'item_category_id', 'item_id'])['item_cnt'].shift(lag)
    # Fill the empty shifted features with 0
    train_monthly[ft_name].fillna(0, inplace=True)
```

[107]:
```python
#generate monthly_trend variable
train_monthly['item_trend'] = train_monthly['item_cnt']

for lag in lag_list:
    ft_name = ('item_cnt_shifted%s' % lag)
    train_monthly['item_trend'] -= train_monthly[ft_name]

train_monthly['item_trend'] /= len(lag_list) + 1
train_monthly.head().T
```

[10…

	0	1	2	3	4
date_block_num	0.000000	0.000	0.00	0.00	0.00
shop_id	2.000000	2.000	2.00	2.00	2.00
item_id	5572.000000	5643.000	5583.00	7893.00	7894.00
item_category_id	2.000000	2.000	5.00	6.00	6.00
item_price	10730.000000	4775.210	1188.30	5970.00	1490.00
mean_item_price	1532.857143	2387.605	594.15	1990.00	1490.00
item_cnt	9.000000	0.000	2.00	3.00	1.00
mean_item_cnt	1.285714	0.000	1.00	1.00	1.00
transactions	7.000000	2.000	2.00	3.00	1.00
year	2013.000000	2013.000	2013.00	2013.00	2013.00
month	0.000000	0.000	0.00	0.00	0.00
item_cnt_month	1.000000	0.000	1.00	2.00	2.00
item_price_unit	1192.000000	inf	594.00	1990.00	1490.00
hist_min_item_price	0.000000	0.000	0.00	0.00	0.00
hist_max_item_price	18979.500000	35260.000	5592.00	42630.00	31290.00
price_increase	10730.000000	4775.210	1188.30	5970.00	1490.00
price_decrease	8249.500000	30484.790	4403.70	36660.00	29800.00
item_cnt_min	9.000000	0.000	2.00	3.00	1.00
item_cnt_max	9.000000	0.000	2.00	3.00	1.00
item_cnt_mean	9.000000	0.000	2.00	3.00	1.00
item_cnt_std	0.000000	0.000	0.00	0.00	0.00
item_cnt_shifted1	0.000000	0.000	0.00	0.00	0.00
item_cnt_shifted2	0.000000	0.000	0.00	0.00	0.00
item_cnt_shifted3	0.000000	0.000	0.00	0.00	0.00
item_trend	2.250000	0.000	0.50	0.75	0.25

We need to pre-define the train and test datasets for this time series data.

The test set is in the future, so we simulate the **same distribution** on our train/validation split.

Our **train set** will be the first 3-27 blocks, **validation** will be last 5 blocks (28-32) and **test** will be block 33.

(We can leave out the first 3 months because we use a 3 month window to generate features, so these first 3 month won't have really windowed useful features.)

+ Code + Markdown

[108]:
```python
#query the train,test, and validation sets with pre-defined blocks
train_set = train_monthly.query('date_block_num >= 3 and date_block_num < 28').copy()
validation_set = train_monthly.query('date_block_num >= 28 and date_block_num < 33').copy()
test_set = train_monthly.query('date_block_num == 33').copy()
#deal with missing data
train_set.dropna(subset=['item_cnt_month'], inplace=True)
validation_set.dropna(subset=['item_cnt_month'], inplace=True)

train_set.dropna(inplace=True)
validation_set.dropna(inplace=True)

print('Train set records:', train_set.shape[0])
print('Validation set records:', validation_set.shape[0])
print('Test set records:', test_set.shape[0])
```

```
Train set records: 4950670
Validation set records: 990025
Test set records: 198001
```

Mean encoding can be quite helpful for the model which we use (a gradient boosting model), and represent the features in a better way. Since we have a lot of different values for shop_ids, item_ids, this can also help in reducing cardinality.

[109]:

```python
# Shop mean encoding.
gp_shop_mean = train_set.groupby(['shop_id']).agg({'item_cnt_month': ['mean']})
gp_shop_mean.columns = ['shop_mean']
gp_shop_mean.reset_index(inplace=True)
# Item mean encoding.
gp_item_mean = train_set.groupby(['item_id']).agg({'item_cnt_month': ['mean']})
gp_item_mean.columns = ['item_mean']
gp_item_mean.reset_index(inplace=True)
# Shop with item mean encoding.
gp_shop_item_mean = train_set.groupby(['shop_id', 'item_id']).agg({'item_cnt_month': ['mean']})
gp_shop_item_mean.columns = ['shop_item_mean']
gp_shop_item_mean.reset_index(inplace=True)
# Year mean encoding.
gp_year_mean = train_set.groupby(['year']).agg({'item_cnt_month': ['mean']})
gp_year_mean.columns = ['year_mean']
gp_year_mean.reset_index(inplace=True)
# Month mean encoding.
gp_month_mean = train_set.groupby(['month']).agg({'item_cnt_month': ['mean']})
gp_month_mean.columns = ['month_mean']
gp_month_mean.reset_index(inplace=True)
```

[110]:

```python
# Add mean encoding features to train set.
train_set = pd.merge(train_set, gp_shop_mean, on=['shop_id'], how='left')
train_set = pd.merge(train_set, gp_item_mean, on=['item_id'], how='left')
train_set = pd.merge(train_set, gp_shop_item_mean, on=['shop_id', 'item_id'], how='left')
train_set = pd.merge(train_set, gp_year_mean, on=['year'], how='left')
train_set = pd.merge(train_set, gp_month_mean, on=['month'], how='left')
# Add mean encoding features to validation set.
validation_set = pd.merge(validation_set, gp_shop_mean, on=['shop_id'], how='left')
validation_set = pd.merge(validation_set, gp_item_mean, on=['item_id'], how='left')
validation_set = pd.merge(validation_set, gp_shop_item_mean, on=['shop_id', 'item_id'], how='left')
validation_set = pd.merge(validation_set, gp_year_mean, on=['year'], how='left')
validation_set = pd.merge(validation_set, gp_month_mean, on=['month'], how='left')
```

[111]:

```python
# Create train and validation sets and labels.
X_train = train_set.drop(['item_cnt_month', 'date_block_num'], axis=1)
Y_train = train_set['item_cnt_month'].astype(int)
X_validation = validation_set.drop(['item_cnt_month', 'date_block_num'], axis=1)
Y_validation = validation_set['item_cnt_month'].astype(int)
```

[116]:

```python
int_features = ['shop_id', 'item_id', 'year', 'month']

X_train[int_features] = X_train[int_features].astype('int32')
X_validation[int_features] = X_validation[int_features].astype('int32')
```

[112]:

```python
latest_records = pd.concat([train_set, validation_set]).drop_duplicates(subset=['shop_id', 'item_id'], keep='last')
X_test = pd.merge(test, latest_records, on=['shop_id', 'item_id'], how='left', suffixes=['', '_'])
X_test['year'] = 2015
X_test['month'] = 9
X_test.drop('item_cnt_month', axis=1, inplace=True)

X_test = X_test[X_train.columns]
```

[113]:

```python
sets = [X_train, X_validation, X_test]
# Replace missing values with the median of each shop.
for dataset in sets:
    for shop_id in dataset['shop_id'].unique():
        for column in dataset.columns:
            shop_median = dataset[(dataset['shop_id'] == shop_id)][column].median()
            dataset.loc[(dataset[column].isnull()) & (dataset['shop_id'] == shop_id), column] = shop_median

# Fill remaining missing values on test set with mean.
X_test.fillna(X_test.mean(), inplace=True)
```

[114]:

```python
# I'm dropping "item_category_id", we don't have it on test set and would be a little hard to create categories for items that exist only on test set.
X_train.drop(['item_category_id'], axis=1, inplace=True)
X_validation.drop(['item_category_id'], axis=1, inplace=True)
X_test.drop(['item_category_id'], axis=1, inplace=True)
```

+ Code + Markdown

[115]:

```
X_test.head().T
```

[11…

	0	1	2	3	4
shop_id	5.000000	5.000000	5.000000	5.000000	5.000000
item_id	5037.000000	5320.000000	5233.000000	5232.000000	5268.000000
item_price	749.500000	0.000000	2997.000000	0.000000	0.000000
mean_item_price	749.500000	0.000000	999.000000	0.000000	0.000000
item_cnt	1.000000	0.000000	3.000000	0.000000	0.000000
mean_item_cnt	1.000000	0.000000	1.000000	0.000000	0.000000
transactions	1.000000	0.000000	3.000000	0.000000	0.000000
year	2015.000000	2015.000000	2015.000000	2015.000000	2015.000000
month	9.000000	9.000000	9.000000	9.000000	9.000000
item_price_unit	749.000000	0.000000	999.000000	0.000000	0.000000
hist_min_item_price	0.000000	0.000000	0.000000	0.000000	0.000000
hist_max_item_price	25990.000000	2495.000000	7191.750000	4796.000000	2495.000000
price_increase	749.500000	0.000000	2997.000000	0.000000	0.000000
price_decrease	25240.500000	2398.000000	4194.750000	4796.000000	2398.000000
item_cnt_min	1.000000	0.000000	1.000000	0.000000	0.000000
item_cnt_max	3.000000	0.000000	3.000000	0.000000	0.000000
item_cnt_mean	1.666667	0.000000	2.000000	0.000000	0.000000
item_cnt_std	1.154701	0.000000	1.000000	0.000000	0.000000
item_cnt_shifted1	3.000000	0.000000	1.000000	0.000000	0.000000
item_cnt_shifted2	1.000000	0.000000	2.000000	0.000000	0.000000
item_cnt_shifted3	1.000000	0.000000	3.000000	0.000000	0.000000
item_trend	-1.000000	0.000000	-0.750000	0.000000	0.000000
shop_mean	0.158739	0.158739	0.158739	0.158739	0.158739
item_mean	0.703527	0.056190	0.071429	0.000000	0.056190
shop_item_mean	0.280000	0.000000	0.120000	0.000000	0.000000
year_mean	0.277102	0.277102	0.277102	0.277102	0.277102
month_mean	0.219153	0.219153	0.219153	0.219153	0.219153