Submission
✔ Ran successfully
Submitted by AshishPatel 7 months ago

Private Score	Public Score
13.06690	13.95666

Light GBM demand-forecasting

Python notebook using data from multiple data sources · 3,548 views · multiple data sources

30 Fork 64

Version 10
10 commits

Notebook
Data
Output
Log
Comments

Table of contents

Introduction

Kernel for the demand forecasting (https://www.kaggle.com/c/demand-forecasting-kernels-only) Kaggle competition.

Answer some of the questions posed:

- What's the best way to deal with seasonality?
- Should stores be modeled separately, or can you pool them together?
- Does deep learning work better than ARIMA?
- Can either beat xgboost?

Preparation

Dependencies

In [1]:

```
import pandas as pd
import numpy as np
from datetime import datetime
import matplotlib.pyplot as plt
import seaborn as sns
plt.style.use('fivethirtyeight')
sns.set()
%matplotlib inline
import plotly.offline as py
```

```
py.init_notebook_mode(connected=True)
import plotly.graph_objs as go
import statsmodels.api as sm
import xgboost as xgb
import lightgbm as lgb
from sklearn.model_selection import train_test_split

import warnings
# import the_module_that_warns

warnings.filterwarnings("ignore")

from fbprophet import Prophet
```

```
/opt/conda/lib/python3.6/site-packages/statsmodels/compat/pandas.p
y:56: FutureWarning:

The pandas.core.datetools module is deprecated and will be removed
in a future version. Please use the pandas.tseries module instead.
```

Load the datasets

In [2]:

```
# Input data files are available in the "../input/" directory.
# First let us load the datasets into different Dataframes
def load_data(datapath):
    data = pd.read_csv(datapath)
   # Dimensions
    print('Shape:', data.shape)
    # Set of features we have are: date, store, and item
    display(data.sample(10))
    return data


train_df = load_data('../input/demand-forecasting-kernels-only/tra
in.csv')
test_df = load_data('../input/demand-forecasting-kernels-only/tes
t.csv')
sample_df = load_data('../input/demand-forecasting-kernels-only/sa
mple_submission.csv')
```

```
Shape: (913000, 4)
```

	date	store	item	sales
431889	2015-08-12	7	24	49
393805	2016-04-30	6	22	96
553144	2017-08-20	3	31	113
10319	2016-04-04	6	1	18
472061	2015-08-12	9	26	59
573654	2013-10-18	5	32	26
808104	2015-10-10	3	45	110
678768	2016-08-15	2	38	94

540056	2016-10-18	6	30	31
456322	2017-07-07	10	25	153

```
Shape: (45000, 4)
```

	id	date	store	item
379	379	2018-01-20	5	1
3960	3960	2018-01-01	5	5
21450	21450	2018-01-31	9	24
33666	33666	2018-01-07	5	38
28309	28309	2018-02-19	5	32
7511	7511	2018-02-11	4	9
2003	2003	2018-01-24	3	3
22559	22559	2018-03-01	1	26
881	881	2018-03-13	10	1
27878	27878	2018-03-10	10	31

```
Shape: (45000, 2)
```

	id	sales
19444	19444	52
28912	28912	52
10073	10073	52
40548	40548	52
4555	4555	52
42578	42578	52
8988	8988	52
32348	32348	52
17238	17238	52
39203	39203	52

Time series data exploration

(This portion was forked (https://www.kaggle.com/danofer/getting-started-with-time-series-features).)

The goal of this kernel is data exploration of a time-series sales data of store items.
The tools `pandas` , `matplotlib` and, `plotly` are used for slicing & dicing the data and visualizations.

Distribution of sales

Now let us understand how the sales varies across all the items in all the stores

Now let us understand how the sales varies across all the items in all the stores

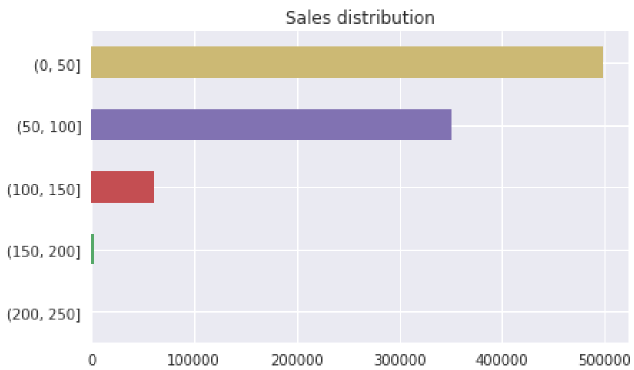
In [3]:

```
# Sales distribution across the train data
def sales_dist(data):
    """
        Sales_dist used for Checing Sales Distribution.
        data :  contain data frame which contain sales data
    """
    sales_df = data.copy(deep=True)
    sales_df['sales_bins'] = pd.cut(sales_df.sales, [0, 50, 100, 1
50, 200, 250])
    print('Max sale:', sales_df.sales.max())
    print('Min sale:', sales_df.sales.min())
    print('Avg sale:', sales_df.sales.mean())
    print()
    return sales_df

sales_df = sales_dist(train_df)

# Total number of data points
total_points = pd.value_counts(sales_df.sales_bins).sum()
print('Sales bucket v/s Total percentage:')
display(pd.value_counts(sales_df.sales_bins).apply(lambda s: (s/to
tal_points)*100))
```

```
Max sale: 231
Min sale: 0
Avg sale: 52.250286966046005

Sales bucket v/s Total percentage:


(0, 50]       54.591407
(50, 100]     38.388322
(100, 150]     6.709974
(150, 200]     0.308544
(200, 250]     0.001752
Name: sales_bins, dtype: float64
```

In [4]:

```
# Let us visualize the same
sales_count = pd.value_counts(sales_df.sales_bins)
sales_count.sort_values(ascending=True).plot(kind='barh', title='S
ales distribution', );
# sns.countplot(sales_count)
```



As we can see, almost 92% of sales are less than 100. Max, min and average sales are 231, 0

and 52.25 respectively.
So any prediction model has to deal with the skewness in the data appropriately.

How does sales vary across stores

Let us get a overview of sales distribution in the whole data.

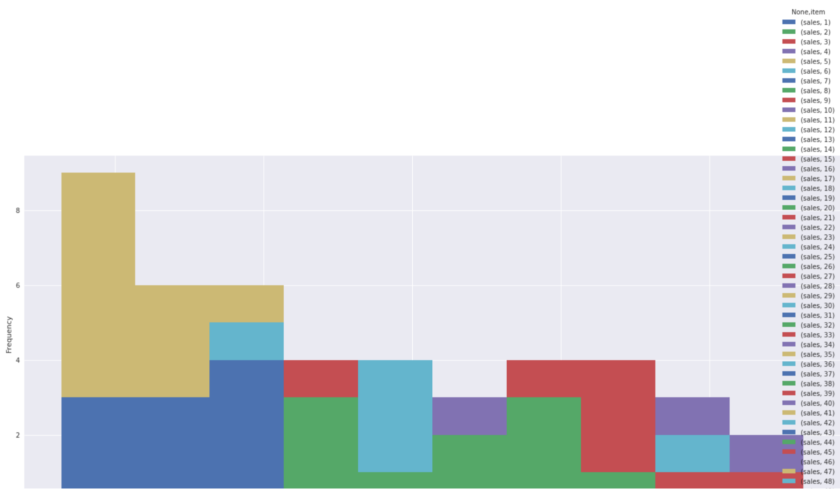
In [5]:

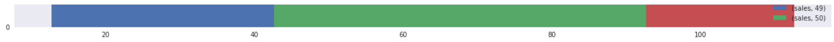
```
# Let us understand the sales data distribution across the stores
def sales_data_understanding(data):
    store_df = data.copy()
    plt.figure(figsize=(20,10))
    sales_pivoted_df = pd.pivot_table(store_df, index='store', val
ues=['sales','date'], columns='item', aggfunc=np.mean)
    sales_pivoted_df.plot(kind="hist",figsize=(20,10))
    # Pivoted dataframe
    display(sales_pivoted_df)
    return (store_df,sales_pivoted_df)

store_df,sales_pivoted_df = sales_data_understanding(train_df)
```

	sales						
item	1	2	3	4	5	6	7
store							
1	19.971522	53.148959	33.208105	19.956188	16.612815	53.060789	52.7836
2	28.173604	75.316539	46.992333	28.234940	23.540526	74.945235	75.0585
3	25.070099	66.804491	41.771084	25.116101	20.857612	67.007119	66.6478
4	22.938664	61.715225	38.548193	23.086528	19.525192	61.270537	61.6254
5	16.739321	44.488499	27.835706	16.776561	14.086528	44.564622	44.5355
6	16.717963	44.533954	27.811062	16.754107	13.893209	44.503834	44.5991
7	15.159365	40.717963	25.531216	15.358160	12.733844	40.703724	40.7097
8	26.983571	71.656627	45.076123	26.948521	22.427711	71.958379	71.7305
9	23.325849	61.792442	38.535049	23.150055	19.272180	61.412377	61.8121
10	24.736035	65.566813	41.113363	24.721249	20.637459	65.612267	65.8077

10 rows × 50 columns

```
<matplotlib.figure.Figure at 0x7f30f9671c18>
```





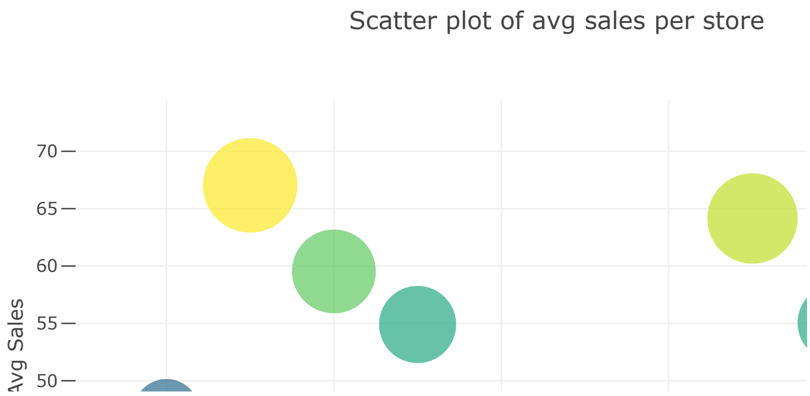
This pivoted dataframe has average sales per each store per each item.
Let use this dataframe and produce some interesting visualizations!

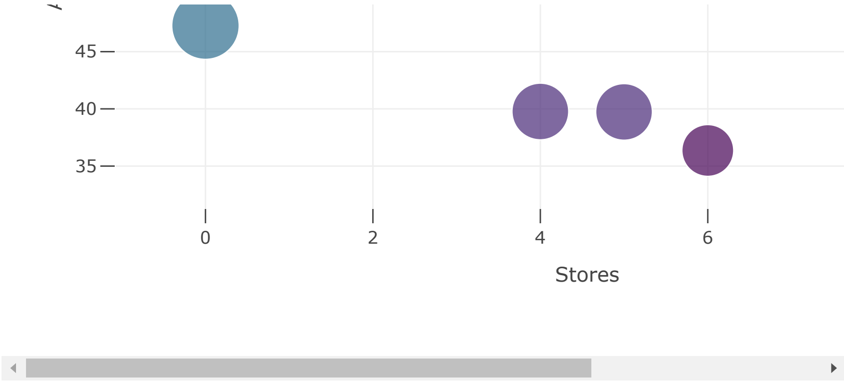
In [6]:

```
# Let us calculate the average sales of all the items by each store
sales_across_store_df = sales_pivoted_df.copy()
sales_across_store_df['avg_sale'] = sales_across_store_df.apply(la
mbda r: r.mean(), axis=1)
```

In [7]:

```
# Scatter plot of average sales per store
sales_store_data = go.Scatter(
    y = sales_across_store_df.avg_sale.values,
    mode='markers',
    marker=dict(
        size = sales_across_store_df.avg_sale.values,
        color = sales_across_store_df.avg_sale.values,
        colorscale='Viridis',
        showscale=True
    ),
    text = sales_across_store_df.index.values
)
data = [sales_store_data]

sales_store_layout = go.Layout(
    autosize= True,
    title= 'Scatter plot of avg sales per store',
    hovermode= 'closest',
    xaxis= dict(
        title= 'Stores',
        ticklen= 10,
        zeroline= False,
        gridwidth= 1,
    ),
    yaxis=dict(
        title= 'Avg Sales',
        ticklen= 10,
        zeroline= False,
        gridwidth= 1,
    ),
    showlegend= False
)
fig = go.Figure(data=data, layout=sales_store_layout)
py.iplot(fig,filename='scatter_sales_store')
```





From the visualization, it is clear that the stores with ID 2 and 8 have higher average sales than the remaining stores and is a clear indication that they are doing good money!

Whereas store with ID 7 has very poor performance in terms of average sales.

How does sales vary across items

In [8]:

```
def sales_insight(sales_pivoted_df):
    # Let us calculate the average sales of each of the item across
 all the stores
    sales_across_item_df = sales_pivoted_df.copy()
    # Aggregate the sales per item and add it as a new row in the sa
me dataframe
    sales_across_item_df.loc[11] = sales_across_item_df.apply(lamb
da r: r.mean(), axis=0)
    # Note the 11th index row, which is the average sale of each of
 the item across all the stores
    #display(sales_across_item_df.loc[11:])
    avg_sales_per_item_across_stores_df = pd.DataFrame(data=[[i+1,
a] for i,a in enumerate(sales_across_item_df.loc[11:].values[0])],
columns=['item', 'avg_sale'])
    # And finally, sort by avg sale
    avg_sales_per_item_across_stores_df.sort_values(by='avg_sale',
ascending=False, inplace=True)
    # Display the top 10 rows
    display(avg_sales_per_item_across_stores_df.head())
    return (sales_across_item_df,avg_sales_per_item_across_stores_
df)

sales_across_item_df,avg_sales_per_item_across_stores_df = sales_i
nsight(sales_pivoted_df)
```

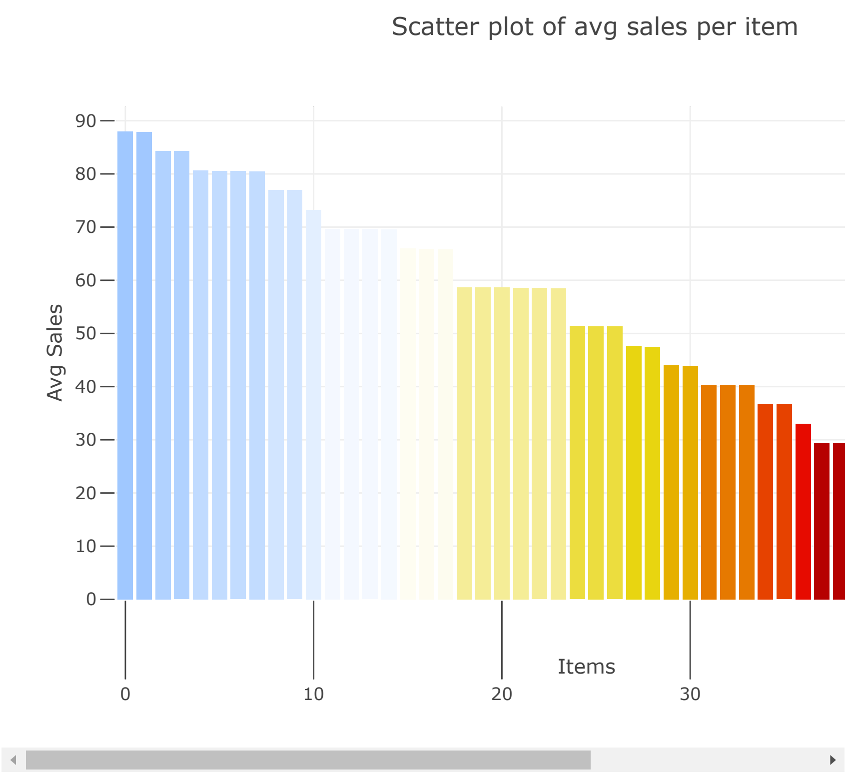
	item	avg_sale
14	15	88.030778
27	28	87.881325
12	13	84.316594
17	18	84.275794
24	25	80.686418

Great! Let us visualize these average sales per item!

In [9]:

```
avg_sales_per_item_across_stores_sorted = avg_sales_per_item_acros
```

```
s_stores_df.avg_sale.values
# Scatter plot of average sales per item
sales_item_data = go.Bar(
    x=[i for i in range(0, 50)],
    y=avg_sales_per_item_across_stores_sorted,
    marker=dict(
        color=avg_sales_per_item_across_stores_sorted,
        colorscale='Blackbody',
        showscale=True
    ),
    text = avg_sales_per_item_across_stores_df.item.values
)
data = [sales_item_data]

sales_item_layout = go.Layout(
    autosize= True,
    title= 'Scatter plot of avg sales per item',
    hovermode= 'closest',
    xaxis= dict(
        title= 'Items',
        ticklen= 55,
        zeroline= False,
        gridwidth= 1,
    ),
    yaxis=dict(
        title= 'Avg Sales',
        ticklen= 10,
        zeroline= False,
        gridwidth= 1,
    ),
    showlegend= False
)
fig = go.Figure(data=data, layout=sales_item_layout)
py.iplot(fig,filename='scatter_sales_item')
```



Amazing! The sales is uniformly distributed across all the items.
Top items with highest average sale are 15, 28, 13, 18 and with least average sales are 5, 1

Top items with highest average sale are 15, 28, 13, 18 and with least average sales are 5, 1, 41 and so on.

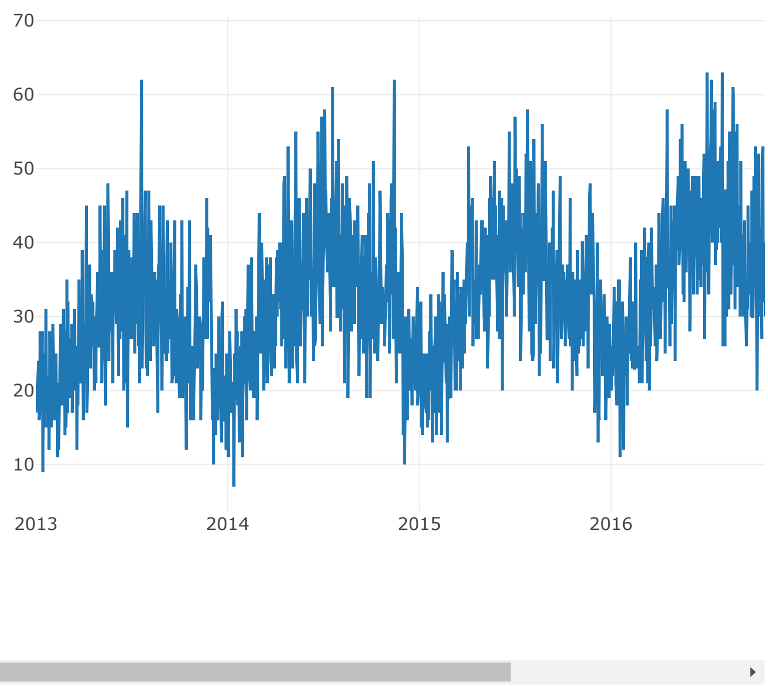
Time-series visualization of the sales

Let us see how sales of a given item in a given store varies in a span of 5 years.

In [10]:

```
def Time_visualization(data):
    store_item_df = data.copy()
    # First, let us filterout the required data
    store_id = 10   # Some store
    item_id = 40    # Some item
    print('Before filter:', store_item_df.shape)
    store_item_df = store_item_df[store_item_df.store == store_id]
    store_item_df = store_item_df[store_item_df.item == item_id]
    print('After filter:', store_item_df.shape)
    #display(store_item_df.head())

    # Let us plot this now
    store_item_ts_data = [go.Scatter(
        x=store_item_df.date,
        y=store_item_df.sales)]
    py.iplot(store_item_ts_data)
    return store_item_df

store_item_df = Time_visualization(train_df)
```

```
Before filter: (913000, 4)
After filter: (1826, 4)
```



Woww! Clearly there is a pattern here! Feel free to play around with different store and item IDs.
Almost all the items and store combination has this pattern!

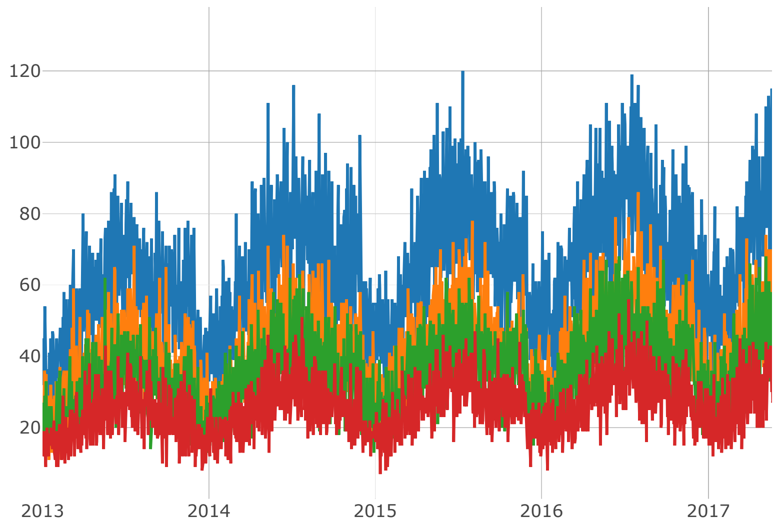
The sales go high in June, July and August months. The sales will be lowest in December, January and February months. That's something!!

Let us make it more interesting. What if we aggregate the sales on a montly basis and compare different items and stores.
This should help us understand how different item sales behave at a high level.

In [11]:

```
def sales_monthly(data):
    multi_store_item_df = data.copy()
    # First, let us filterout the required data
    store_ids = [1, 1, 1, 1]   # Some stores
    item_ids = [10, 20, 30, 40]    # Some items
    print('Before filter:', multi_store_item_df.shape)
    multi_store_item_df = multi_store_item_df[multi_store_item_df.
store.isin(store_ids)]
    multi_store_item_df = multi_store_item_df[multi_store_item_df.
item.isin(item_ids)]
    print('After filter:', multi_store_item_df.shape)
    #display(multi_store_item_df)
    # TODO Monthly avg sales

    # Let us plot this now
    multi_store_item_ts_data = []
    for st,it in zip(store_ids, item_ids):
        flt = multi_store_item_df[multi_store_item_df.store == st]
        flt = flt[flt.item == it]
        multi_store_item_ts_data.append(go.Scatter(x=flt.date, y=f
lt.sales, name = "Store:" + str(st) + ",Item:" + str(it)))
    py.iplot(multi_store_item_ts_data)
    return (multi_store_item_df)

multi_store_item_df = sales_monthly(train_df)
```

```
Before filter: (913000, 4)
After filter: (7304, 4)
```



Interesting!!

Though the pattern remains same across different stores and items combinations, the **actual sale value consitently varies with the same scale**.

As we can see in the visualization, item 10 has consistently highest sales through out the span of 5 years!
This is an interesting behaviour that can be seen across almost all the items.

ARIMA

ARIMA is Autoregressive Integrated Moving Average Model, which is a component of SARIMAX, i.e. Seasonal ARIMA with eXogenous regressors.

(sources: 1 (https://machinelearningmastery.com/arima-for-time-series-forecasting-with-python/), 2 (https://www.digitalocean.com/community/tutorials/a-guide-to-time-series-forecasting-with-arima-in-python-3), 3 (http://pandas.pydata.org/pandas-docs/stable/timeseries.html#offset-aliases))

http://pandas.pydata.org/pandas-docs/stable/timeseries.html#offset-aliases (http://pandas.pydata.org/pandas-docs/stable/timeseries.html#offset-aliases)

LIGHTGBM

In [12]:

```
def split_data(train_data,test_data):
    train_data['date'] = pd.to_datetime(train_data['date'])
    test_data['date'] = pd.to_datetime(test_data['date'])

    train_data['month'] = train_data['date'].dt.month
    train_data['day'] = train_data['date'].dt.dayofweek
    train_data['year'] = train_data['date'].dt.year

    test_data['month'] = test_data['date'].dt.month
    test_data['day'] = test_data['date'].dt.dayofweek
    test_data['year'] = test_data['date'].dt.year

    col = [i for i in test_data.columns if i not in ['date','id']]
    y = 'sales'
    train_x, test_x, train_y, test_y = train_test_split(train_data
[col],train_data[y], test_size=0.2, random_state=2018)
    return (train_x, test_x, train_y, test_y,col)

train_x, test_x, train_y, test_y,col = split_data(train_df,test_df
)
```

In [13]:

```
train_x.shape,train_y.shape,test_x.shape
```

Out[13]:

```
((730400, 5), (730400,), (182600, 5))
```

In [14]:

```
# from bayes_opt import BayesianOptimization
# def bayes_parameter_opt_lgb(X, y, init_round=15, opt_round=25, n_f
olds=5, random_seed=6, n_estimators=10000, learning_rate=0.02, outpu
t_process=False):
#     # prepare data
#     train_data = lgb.Dataset(data=X, label=y)
#     # parameters
#     def lgb_eval(num_leaves, feature_fraction, bagging_fraction, m
ax_depth, lambda_l1, lambda_l2, min_split_gain, min_child_weight):
```

```
#         params = {'application':'regression_l1','num_iterations':
 n_estimators, 'learning_rate':learning_rate, 'early_stopping_roun
d':100, 'metric':'auc'}
#         params["num_leaves"] = int(round(num_leaves))
#         params['feature_fraction'] = max(min(feature_fraction, 1),
0)
#         params['bagging_fraction'] = max(min(bagging_fraction, 1),
0)
#         params['max_depth'] = int(round(max_depth))
#         params['lambda_l1'] = max(lambda_l1, 0)
#         params['lambda_l2'] = max(lambda_l2, 0)
#         params['min_split_gain'] = min_split_gain
#         params['min_child_weight'] = min_child_weight
#         cv_result = lgb.cv(params, train_data, nfold=n_folds, seed
=random_seed, stratified=True, verbose_eval =200, metrics=['auc'])
#         return max(cv_result['auc-mean'])
#     # range
#     lgbBO = BayesianOptimization(lgb_eval, {'num_leaves': (24, 4
5),
#                                             'feature_fraction':
 (0.1, 0.9),
#                                             'bagging_fraction':
 (0.8, 1),
#                                             'max_depth': (5, 8.9
9),
#                                             'lambda_l1': (0, 5),
#                                             'lambda_l2': (0, 3),
#                                             'min_split_gain': (0.0
01, 0.1),
#                                             'min_child_weight':
 (5, 50)}, random_state=0)
#     # optimize
#     lgbBO.maximize(init_points=init_round, n_iter=opt_round)

#     # output optimization process
#     if output_process==True: lgbBO.points_to_csv("bayes_opt_resul
t.csv")

#     # return best parameters
#     return lgbBO.res['max']['max_params']

# opt_params = bayes_parameter_opt_lgb(train_x, train_y, init_round=
5, opt_round=10, n_folds=3, random_seed=6, n_estimators=100, learnin
g_rate=0.02)
```

In [15]:

```
# opt_params
```

In [16]:

```
%%time

def model(train_x,train_y,test_x,test_y,col):
    params = {
        'nthread': 10,
         'max_depth': 5,
#          'max_depth': 9,
        'task': 'train',
        'boosting_type': 'gbdt',
        'objective': 'regression_l1',
        'metric': 'mape', # this is abs(a-e)/max(1,a)
#          'num_leaves': 39,
        'num_leaves': 64,
        'learning_rate': 0.2,
```

```
        'feature_fraction': 0.9,
#          'feature_fraction': 0.8108472661400657,
#          'bagging_fraction': 0.9837558288375402,
        'bagging_fraction': 0.8,
         'bagging_freq': 5,
         'lambda_l1': 3.097758978478437,
         'lambda_l2': 2.9482537987198496,
#        'lambda_l1': 0.06,
#        'lambda_l2': 0.1,
         'verbose': 1,
         'min_child_weight': 6.996211413900573,
         'min_split_gain': 0.037310344962162616,
         }
    lgb_train = lgb.Dataset(train_x,train_y)
    lgb_valid = lgb.Dataset(test_x,test_y)
    model = lgb.train(params, lgb_train, 3000, valid_sets=[lgb_tra
```


Data

Data Sources

- [Private Dataset]
- Store Item Demand F...
  - sample_submission....
  - test.csv
  - train.csv

[Private Dataset]

Access to this Dataset is restricted.
You are seeing this placeholder because you have access to the Kernel.

Output Files

New Dataset | New Kernel | Download All

Output Files

- lgb_bayasian_param.csv
- Submission.csv

About this file

This file was created from a Kernel, it does not have a description.

lgb_bayasian_param.csv

1	id	sales
2	0	12.096060960947812
3	1	13.483620195751687
4	2	14.59761864377739
5	3	14.638576084571595
6	4	15.24021595349735
7	5	16.171887857432512
8	6	18.40053453614795
9	7	12.096060960947812
10	8	13.483620195751687

11	9	14.5976186 4377739
12	10	14.6385760 84571595
13	11	15.2402159 5349735
14	12	16.1718878 57432512
15	13	18.4005345 3614795
16	14	12.0960609 60947812
17	15	13.4836201 95751687
18	16	14.5976186 4377739
19	17	14.6385760 84571595
20	18	15.2402159 5349735

Run Info

Succeeded	True	Run Time	153.7 seconds
Exit Code	0	Queue Time	0 seconds
Docker Image Name	/python(Dockerfile)	Output Size	0
Timeout Exceeded	False	Used All Space	False
Failure Message			

Log

Download Log

```
Time   Line #  Log Message
3.9s        1  [NbConvertApp] Converting notebook script.ipynb to html
4.0s        2  [NbConvertApp] Executing notebook with kernel: python3
13.2s       3  [LightGBM] [Info] Total Bins 88
13.2s       4  [LightGBM] [Info] Number of data: 730400, number of used features:
               5
13.3s       5  [LightGBM] [Info] Start training from score 47.000000
13.3s       6  [LightGBM] [Warning] No further splits with positive gain, best
               gain: -inf
13.4s       7  [LightGBM] [Warning] No further splits with positive gain, best
               gain: -inf
13.5s       8  [LightGBM] [Warning] No further splits with positive gain, best
               gain: -inf
13.7s       9  [LightGBM] [Warning] No further splits with positive gain, best
               gain: -inf
13.8s      10  [LightGBM] [Warning] No further splits with positive gain, best
               gain: -inf
13.9s      11  [LightGBM] [Warning] No further splits with positive gain, best
               gain: -inf
14.0s      12  [LightGBM] [Warning] No further splits with positive gain, best
               gain: -inf
14.1s      13  [LightGBM] [Warning] No further splits with positive gain, best
               gain: -inf
14.2s      14  [LightGBM] [Warning] No further splits with positive gain, best
               gain: -inf
14.3s      15  [LightGBM] [Warning] No further splits with positive gain, best
               gain: -inf
14.4s      16  [LightGBM] [Warning] No further splits with positive gain, best
               gain: -inf
14.5s      17  [LightGBM] [Warning] No further splits with positive gain, best
               gain: -inf
14.6s      18  [LightGBM] [Warning] No further splits with positive gain, best
               gain: -inf
14.7s      19  [LightGBM] [Warning] No further splits with positive gain, best
               gain: -inf
14.8s      20  [LightGBM] [Warning] No further splits with positive gain, best
               gain: -inf
14.9s      21  [LightGBM] [Warning] No further splits with positive gain, best
               gain: -inf
15.0s      22  [LightGBM] [Warning] No further splits with positive gain, best
               gain: -inf
```

```
15.1s    23  [LightGBM] [Warning] No further splits with positive gain, best gain: -inf
15.1s    24  [LightGBM] [Warning] No further splits with positive gain, best gain: -inf
15.2s    25  [LightGBM] [Warning] No further splits with positive gain, best gain: -inf
15.3s    26  [LightGBM] [Warning] No further splits with positive gain, best gain: -inf
15.4s    27  [LightGBM] [Warning] No further splits with positive gain, best gain: -inf
15.5s    28  [LightGBM] [Warning] No further splits with positive gain, best gain: -inf
15.6s    29  [LightGBM] [Warning] No further splits with positive gain, best gain: -inf
15.7s    30  [LightGBM] [Warning] No further splits with positive gain, best gain: -inf
15.8s    31  [LightGBM] [Warning] No further splits with positive gain, best gain: -inf
15.9s    32  [LightGBM] [Warning] No further splits with positive gain, best gain: -inf
16.0s    33  [LightGBM] [Warning] No further splits with positive gain, best gain: -inf
16.1s    34  [LightGBM] [Warning] No further splits with positive gain, best gain: -inf
16.1s    35  [LightGBM] [Warning] No further splits with positive gain, best gain: -inf
16.2s    36  [LightGBM] [Warning] No further splits with positive gain, best gain: -inf
16.3s    37  [LightGBM] [Warning] No further splits with positive gain, best gain: -inf
16.4s    38  [LightGBM] [Warning] No further splits with positive gain, best gain: -inf
16.5s    39  [LightGBM] [Warning] No further splits with positive gain, best gain: -inf
16.7s    40  [LightGBM] [Warning] No further splits with positive gain, best gain: -inf
16.8s    41  [LightGBM] [Warning] No further splits with positive gain, best gain: -inf
16.9s    42  [LightGBM] [Warning] No further splits with positive gain, best gain: -inf
17.0s    43  [LightGBM] [Warning] No further splits with positive gain, best gain: -inf
17.1s    44  [LightGBM] [Warning] No further splits with positive gain, best gain: -inf
17.1s    45  [LightGBM] [Warning] No further splits with positive gain, best gain: -inf
17.2s    46  [LightGBM] [Warning] No further splits with positive gain, best gain: -inf
17.3s    47  [LightGBM] [Warning] No further splits with positive gain, best gain: -inf
17.4s    48  [LightGBM] [Warning] No further splits with positive gain, best gain: -inf
17.5s    49  [LightGBM] [Warning] No further splits with positive gain, best gain: -inf
17.6s    50  [LightGBM] [Warning] No further splits with positive gain, best gain: -inf
17.7s    51  [LightGBM] [Warning] No further splits with positive gain, best gain: -inf
17.8s    52  [LightGBM] [Warning] No further splits with positive gain, best gain: -inf
17.9s    53  [LightGBM] [Warning] No further splits with positive gain, best gain: -inf
18.0s    54  [LightGBM] [Warning] No further splits with positive gain, best gain: -inf
18.1s    55  [LightGBM] [Warning] No further splits with positive gain, best gain: -inf
18.2s    56  [LightGBM] [Warning] No further splits with positive gain, best gain: -inf
18.3s    57  [LightGBM] [Warning] No further splits with positive gain, best gain: -inf
18.4s    58  [LightGBM] [Warning] No further splits with positive gain, best gain: -inf
18.5s    59  [LightGBM] [Warning] No further splits with positive gain, best gain: -inf
18.6s    60  [LightGBM] [Warning] No further splits with positive gain, best gain: -inf
18.7s    61  [LightGBM] [Warning] No further splits with positive gain, best gain: -inf
18.8s    62  [LightGBM] [Warning] No further splits with positive gain, best gain: -inf
18.9s    63  [LightGBM] [Warning] No further splits with positive gain, best gain: -inf
19.0s    64  [LightGBM] [Warning] No further splits with positive gain, best gain: -inf
19.0s    65  [LightGBM] [Warning] No further splits with positive gain, best gain: -inf
19.2s    66  [LightGBM] [Warning] No further splits with positive gain, best gain: -inf
```

```
19.3s     67  [LightGBM] [Warning] No further splits with positive gain, best
              gain: -inf
19.3s     68  [LightGBM] [Warning] No further splits with positive gain, best
              gain: -inf
19.4s     69  [LightGBM] [Warning] No further splits with positive gain, best
              gain: -inf
19.5s     70  [LightGBM] [Warning] No further splits with positive gain, best
              gain: -inf
19.6s     71  [LightGBM] [Warning] No further splits with positive gain, best
              gain: -inf
19.7s     72  [LightGBM] [Warning] No further splits with positive gain, best
              gain: -inf
19.8s     73  [LightGBM] [Warning] No further splits with positive gain, best
              gain: -inf
19.9s     74  [LightGBM] [Warning] No further splits with positive gain, best
              gain: -inf
20.0s     75  [LightGBM] [Warning] No further splits with positive gain, best
              gain: -inf
20.1s     76  [LightGBM] [Warning] No further splits with positive gain, best
              gain: -inf
20.2s     77  [LightGBM] [Warning] No further splits with positive gain, best
              gain: -inf
20.3s     78  [LightGBM] [Warning] No further splits with positive gain, best
              gain: -inf
20.4s     79  [LightGBM] [Warning] No further splits with positive gain, best
              gain: -inf
20.5s     80  [LightGBM] [Warning] No further splits with positive gain, best
              gain: -inf
20.6s     81  [LightGBM] [Warning] No further splits with positive gain, best
              gain: -inf
20.7s     82  [LightGBM] [Warning] No further splits with positive gain, best
              gain: -inf
20.8s     83  [LightGBM] [Warning] No further splits with positive gain, best
              gain: -inf
20.8s     84  [LightGBM] [Warning] No further splits with positive gain, best
              gain: -inf
20.9s     85  [LightGBM] [Warning] No further splits with positive gain, best
              gain: -inf
21.0s     86  [LightGBM] [Warning] No further splits with positive gain, best
              gain: -inf
21.1s     87  [LightGBM] [Warning] No further splits with positive gain, best
              gain: -inf
21.2s     88  [LightGBM] [Warning] No further splits with positive gain, best
              gain: -inf
21.3s     89  [LightGBM] [Warning] No further splits with positive gain, best
              gain: -inf
21.4s     90  [LightGBM] [Warning] No further splits with positive gain, best
              gain: -inf
21.5s     91  [LightGBM] [Warning] No further splits with positive gain, best
              gain: -inf
21.6s     92  [LightGBM] [Warning] No further splits with positive gain, best
              gain: -inf
21.7s     93  [LightGBM] [Warning] No further splits with positive gain, best
              gain: -inf
21.8s     94  [LightGBM] [Warning] No further splits with positive gain, best
              gain: -inf
21.9s     95  [LightGBM] [Warning] No further splits with positive gain, best
              gain: -inf
22.0s     96  [LightGBM] [Warning] No further splits with positive gain, best
              gain: -inf
22.1s     97  [LightGBM] [Warning] No further splits with positive gain, best
              gain: -inf
22.2s     98  [LightGBM] [Warning] No further splits with positive gain, best
              gain: -inf
22.3s     99  [LightGBM] [Warning] No further splits with positive gain, best
              gain: -inf
22.4s    100  [LightGBM] [Warning] No further splits with positive gain, best
              gain: -inf
153.2s  1348
153.2s  1349  ...
153.2s  1350  Complete. Exited with code 0.
```

Comments (8)

Sort by

All Comments | Hotness

Click here to enter a comment...

Eric Hamers • Posted on Latest Version • 7 months ago • Options • Reply 1

This is looking good, when are you going to add the ARIMA and DL models?

Ashish Pat... Kernel Author • Posted on Latest Version • 7 months ago • Options • Reply 0

Sure Soon I will apply it.

Anirudh That... • Posted on Latest Version • 8 months ago • Options • Reply 1

Thanks for this outstanding post and the explanation.

Cauveri • Posted on Latest Version • 8 months ago • Options • Reply 1

why you are using this file ."../input/private/sub_val-0.132358565029612.csv"..can you explain about this file?

Ashish Pat... Kernel Author • Posted on Latest Version • 8 months ago • Options • Reply 1

I had just blend with that file.

nagin • Posted on Latest Version • 7 months ago • Options • Reply 1

How can I get ./ input / private / sub_val-0.132358565029612.csv file?
I cannot access your private data folder

Aditya Soni • Posted on Latest Version • 8 months ago • Options • Reply 1

Great Kernel And Plots !!!!

NikitPatel • Posted on Version 9 • 8 months ago • Options • Reply 2

Nice Kernel!!!!