2017.9.15更新

- 在第二轮的特征处理中，添加pairplot用以特征交互，使用了Holoviews库
- 用多个交互式直方图 Plotly库 来检验距离特征
- 用关联矩阵热点图 Seaborn库 来检验关联变量
- 用 Seaborn库 检验测试集和有效数据

# 序文

所有的python可视化库

| No. | 包 | kernel中用到的图像 | 备注 | 图像性质 |
|---|---|---|---|---|
| 1 | Matplotlib | vendor_id直方图；<br>store_and_fwd_flag直方图 | 简单快速首选 | |
| 2 | Seaborn | 乘客人数vs duration的小提琴图；<br>weekday vs duration的箱型图；<br>hours,weekday vs 平均duration 的tsplot时序图；<br>distplot of latlong and duration? | | |
| 3 | pandas | 对比坐标（对特征群） | | |
| 4 | Bokeh | 时间序列图（day of the year vs avg duration) | | 交互式 |
| 5 | Folium | 曼哈顿的接乘客地点；<br>（美国）群的地点<br>（曼哈顿）群的地点 | | 交互式 |
| 6 | Pygmaps | 地点可视化<br>集群可视化 | | 交互式 |
| 7 | Plotly | bubble图 | | 交互式 |
| 8 | Gmaps | | | 交互式 |
| 9 | Ggplot2 | 给定时期的NYC的天气图 | | 交互式 |
| 10 | Basemaps | 无 | | |
| 11 | No package | NYC出租车交通热图 | | |
| 12 | Datashader | 地点热点图 | | 交互式 |
| 13 | Holoviews | pairplot for 特征交互 | | 交互式 |

*在这个记事本里包括了一个动画制作，是使用了matplotlib的动画制作来展示一天内不同时刻的接送热点图，比如随时间而变化的不同地点的接送密度*

以上有一些包不被kaggle kernels支持，必须下载notebook让他们工作

完成所有这些可视化后，使用额外的特征来构建XGBoost model来做预测

## 关于比赛

使用的数据集：train-test，NYC OSRM 数据集， 天气数据

### 引入用作分析的包

```
import pandas as pd  #pandas for using dataframe and reading csv
import numpy as np   #numpy for vector operations and basic maths
#import simplejson    #getting JSON in simplified format
import urllib        #for url stuff
#import gmaps       #for using google maps to visulalize places on maps
import re            #for processing regular expressions
import datetime      #for datetime operations
import calendar      #for calendar for datetime operations
import time          #to get the system time
import scipy         #for other dependancies
from sklearn.cluster import KMeans # for doing K-means clustering
from haversine import haversine # for calculating haversine distance
import math          #for basic maths operations
import seaborn as sns #for making plots
import matplotlib.pyplot as plt # for plotting
import os  # for os commands
from scipy.misc import imread, imresize, imsave  # for plots
import plotly.plotly as py
import plotly.graph_objs as go
import plotly
from bokeh.palettes import Spectral4
from bokeh.plotting import figure, output_notebook, show
from IPython.display import HTML
from matplotlib.pyplot import *
from matplotlib import cm
from matplotlib import animation
import io
import base64
output_notebook()
plotly.offline.init_notebook_mode() # run at the start of every ipython notebookimport pandas as
pd
...

output_notebook()
plotly.offline.init_notebook_mode() # run at the start of every ipython notebook ?
```

### 读入并检查训练集和OSRM fastest route dataset的head

training data是比赛一开始就给出的。两个礼拜后，oscarleo在Kaggle datasets上传了这个数据集（OSRM）。这个数据集是从Open Source Routing Machine生成的。这和Gmaps很像但这个是开源的，所以一个用户可以任意次地使用这个引擎进行查询（不像Gmaps，一天的免费查询数限制到2000）尽管我已经比较了OSRM和Gmaps下的结果，Gmaps的结果相当不一样，几乎在同一个部分下对time duration的所有结果都不一样。（？？）所以训练任何回归量，都会对我们去预测duration非常有帮助。

```
#导入训练集和fastest_routes_train集，合并表，得到train_df
s = time.time()
train_fr_1 = pd.read_csv('../input/new-york-city-taxi-with-
osrm/fastest_routes_train_part_1.csv')
train_fr_2 = pd.read_csv('../input/new-york-city-taxi-with-
osrm/fastest_routes_train_part_2.csv')
train_fr = pd.concat([train_fr_1, train_fr_2])#concat合并表 fastest_routes_train
train_fr_new = train_fr[['id', 'total_distance', 'total_travel_time', 'number_of_steps']]   #从
train_fr中提取4个字段形成新的表
train_df = pd.read_csv('../input/new-york-city-taxi-with-osrm/train.csv')
train = pd.merge(train_df, train_fr_new, on = 'id', how = 'left') #left意味着根据train_df中的项寻
找train_fr_new中的项，加进去，没有的项用NaN表示
train_df = train.copy()
end = time.time()
print("Time taken by above cell is {}.".format((end-s)))
train_df.head() #列出前五条项
```

| id | vendor_id | pickup_datetime | dropoff_datetime | passenger_count | pickup_longitude | pickup_latitude | dropoff_longitude | dropoff_latitude | store_and_fwd_flag | trip_duration | total_distance | to |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | id2875421 | 2 | 2016-03-14 17:24:55 | 2016-03-14 17:32:30 | 1 | -73.982155 | 40.767937 | -73.964630 | 40.765602 | N | 455 | 20 |
| 1 | id2377394 | 1 | 2016-06-12 00:43:35 | 2016-06-12 00:54:38 | 1 | -73.980415 | 40.738564 | -73.999481 | 40.731152 | N | 663 | 25 |
| 2 | id3858529 | 2 | 2016-01-19 11:35:24 | 2016-01-19 12:10:48 | 1 | -73.979027 | 40.763939 | -74.005333 | 40.710087 | N | 2124 | 11 |
| 3 | id3504673 | 2 | 2016-04-06 19:32:31 | 2016-04-06 19:39:40 | 1 | -74.010040 | 40.719971 | -74.012268 | 40.706718 | N | 429 | 17 |
| 4 | id2181028 | 2 | 2016-03-26 13:30:55 | 2016-03-26 13:38:10 | 1 | -73.973053 | 40.793209 | -73.972923 | 40.782520 | N | 435 | 16 |

```
# checking if Ids are unique
start = time.time()
train_data = train_df.copy()
start = time.time()
print("Number of columns and rows and columns are {} and {}
respectively.".format(train_data.shape[1], train_data.shape[0]))    #shape为数据表的维度
if train_data.id.nunique() == train_data.shape[0]:  #nunqiue统计多少个不同的值
    print("Train ids are unique")
print("Number of Nulls - {}.".format(train_data.isnull().sum().sum()))
end = time.time()
print("Time taken by above cell is {}.".format(end-start))
```

```
Number of columns and rows and columns are 14 and 1458644 respectively.
Train ids are unique
Number of Nulls - 3.
Time taken by above cell is 1.069046974182129.
```

我们看到训练集里包括了14个列和大约1.4百万个数据记录，没有空记录

## 用sns库里的log-scale distplot来将duration可视化

首先用平轴来绘制duration，但由于有一些记录数值过大，其他部分记录不可见。然后使用了log-scale

另一个使用原因是这个比赛使用的是rmsle matrix（rmsle评估算法）所以用log scale来可视化目标变量 是有意义的

```
%matplotlib inline  #在ipython或jupyter notebook里使用，plot()绘图会直接在console里出现，其他ide直接注释即可
start = time.time()
sns.set(style="white", palette="muted", color_codes=True)#图像设置，palette调色板
f, axes = plt.subplots(1, 1, figsize=(11, 7), sharex=True)#f? subplots?
sns.despine(left=True)#despine删除上方和右方的框框
sns.distplot(np.log(train_df['trip_duration'].values+1), axlabel = 'Log(trip_duration)', label = 'log(trip_duration)', bins = 50, color="r")
plt.setp(axes, yticks=[])#setp?
plt.tight_layout()#tight_layout调整
end = time.time()
print("Time taken by above cell is {}.".format((end-start)))
plt.show()
```

图像Log(trip_duration)

**发现** 类似高斯分布 很少有大的数据（100h) 绝大多数是 e^4 = 1 minute to e^8 ~ 60 minutes.？？

下面检验经纬分布

```
start = time.time()
sns.set(style="white", palette="muted", color_codes=True)
f, axes = plt.subplots(2,2,figsize=(10, 10), sharex=False, sharey = False)#sharex?sharey
sns.despine(left=True)
sns.distplot(train_df['pickup_latitude'].values, label = 'pickup_latitude',color="m",bins = 100, ax=axes[0,0])#bins 矩形图数量；  ax为子图在整张图的坐标
sns.distplot(train_df['pickup_longitude'].values, label = 'pickup_longitude',color="m",bins =100, ax=axes[0,1])
sns.distplot(train_df['dropoff_latitude'].values, label = 'dropoff_latitude',color="m",bins =100, ax=axes[1, 0])
sns.distplot(train_df['dropoff_longitude'].values, label = 'dropoff_longitude',color="m",bins =100, ax=axes[1, 1])
plt.setp(axes, yticks=[])
plt.tight_layout()
end = time.time()
print("Time taken by above cell is {}.".format((end-start)))
plt.show()
```

**发现** pick 和 drop 纬度 集中在40 ~ 41附近，经度为-74 ~ -73

由于sns库的distplot方程被一些彼此相距非常远的车程（比如纬度32到44）所影响，当我们绘制经纬图时我们没有得到任何的直方图。这些车程花费非常长的时间，使得图像表现的像个长长的钉子一样。

我们对经纬做限制，去掉那些经纬度偏差较大的duration trip

```
start = time.time()
df = train_df.loc[(train_df.pickup_latitude > 40.6) & (train_df.pickup_latitude < 40.9)]
df = df.loc[(df.dropoff_latitude > 40.6) & (df.dropoff_latitude < 40.9)]
df = df.loc[(df.dropoff_longitude > -74.05) & (df.dropoff_longitude < -73.7)]
df = df.loc[(df.pickup_longitude > -74.05) & (df.pickup_longitude < -73.7)]
train_data_new = df.copy()
sns.set(style="white", palette="muted", color_codes=True)
```

```
f, axes = plt.subplots(2,2,figsize=(12, 12), sharex=False, sharey = False)#
sns.despine(left=True)
sns.distplot(train_data_new['pickup_latitude'].values, label = 'pickup_latitude',color="m",bins
= 100, ax=axes[0,0])
sns.distplot(train_data_new['pickup_longitude'].values, label =
'pickup_longitude',color="g",bins =100, ax=axes[0,1])
sns.distplot(train_data_new['dropoff_latitude'].values, label =
'dropoff_latitude',color="m",bins =100, ax=axes[1, 0])
sns.distplot(train_data_new['dropoff_longitude'].values, label =
'dropoff_longitude',color="g",bins =100, ax=axes[1, 1])
plt.setp(axes, yticks=[])
plt.tight_layout()
end = time.time()
print("Time taken by above cell is {}.".format((end-start)))
print(df.shape[0], train_data.shape[0])
plt.show()
```

**发现**

我们对经纬作以下限制

- 纬度 40.6 ~ 40.9
- 经度 -74.05 ~ -73.70

  我们得到了像长针一样的分布图（distplot在seaborn包里是直方图），我们可以看到大多数车程是集中在这些经纬段中的。由于我们不能在kaggle kernel上使用gmpas和folium来进行可视化，我们要在一张空图上绘制他们，并检验我们得到的是什么样的城市地图。**更新** - 当时kaggle python docker上不能用folium，现在我们可以用了并在kernel中包括了漂亮的folium可视图。

```
start = time.time()
temp = train_data.copy()
train_data['pickup_datetime'] = pd.to_datetime(train_data.pickup_datetime)
train_data.loc[:, 'pick_date'] = train_data['pickup_datetime'].dt.date
train_data.head()

ts_v1 = pd.DataFrame(train_data.loc[train_data['vendor_id']==1].groupby('pick_date')
['trip_duration'].mean())
ts_v1.reset_index(inplace = True)
ts_v2 = pd.DataFrame(train_data.loc[train_data.vendor_id==2].groupby('pick_date')
['trip_duration'].mean())
ts_v2.reset_index(inplace = True)

from bokeh.palettes import Spectral4
from bokeh.plotting import figure, output_notebook, show
#from bokeh.sampledata.stocks import AAPL, IBM, MSFT, GOOG
output_notebook()
p = figure(plot_width=800, plot_height=250, x_axis_type="datetime")
p.title.text = 'Click on legend entries to hide the corresponding lines'

for data, name, color in zip([ts_v1, ts_v2], ["vendor 1", "vendor 2"], Spectral4):
    df = data

    p.line(df['pick_date'], df['trip_duration'], line_width=2, color=color, alpha=0.8,
```

```
legend=name)

p.legend.location = "top_left"
p.legend.click_policy="hide"
show(p)
end = time.time()
train_data = temp
print("Time Taken by above cell is {}.".format(end - start))
```

## 坐标热点图

一些基础的图像处理

经纬度点的可视化

kernel不支持gmaps和folium，当时也未用“datashader“包，使用了一些基本的图像处理技术来完成图像

```
start = time.time()
rgb = np.zeros((3000, 3500, 3), dtype=np.uint8)
rgb[..., 0] = 0
rgb[..., 1] = 0
rgb[..., 2] = 0
train_data_new['pick_lat_new'] = list(map(int, (train_data_new['pickup_latitude'] -
(40.6000))*10000))
train_data_new['drop_lat_new'] = list(map(int, (train_data_new['dropoff_latitude'] -
(40.6000))*10000))
train_data_new['pick_lon_new'] = list(map(int, (train_data_new['pickup_longitude'] -
(-74.050))*10000))
train_data_new['drop_lon_new'] = list(map(int,(train_data_new['dropoff_longitude'] -
(-74.050))*10000))

summary_plot = pd.DataFrame(train_data_new.groupby(['pick_lat_new', 'pick_lon_new'])
['id'].count())

summary_plot.reset_index(inplace = True)
summary_plot.head(120)
lat_list = summary_plot['pick_lat_new'].unique()
for i in lat_list:
    lon_list = summary_plot.loc[summary_plot['pick_lat_new']==i]['pick_lon_new'].tolist()
    unit = summary_plot.loc[summary_plot['pick_lat_new']==i]['id'].tolist()
    for j in lon_list:
        a = unit[lon_list.index(j)]
        if (a//50) >0:
            rgb[i][j][0] = 255
            rgb[i,j, 1] = 0
            rgb[i,j, 2] = 255
        elif (a//10)>0:
            rgb[i,j, 0] = 0
            rgb[i,j, 1] = 255
            rgb[i,j, 2] = 0
        else:
            rgb[i,j, 0] = 255
            rgb[i,j, 1] = 0
```

```
            rgb[i,j, 2] = 0
fig, ax = plt.subplots(nrows=1,ncols=1,figsize=(14,20))
end = time.time()
print("Time taken by above cell is {}.".format((end-start)))
ax.imshow(rgb, cmap = 'hot')
ax.set_axis_off()
```

**发现**

- 红点 1-10个车程 该店为接客点
- 绿点 10-50
- 黄点 50+

整个曼哈顿是黄色的并只有少数绿点，表面曼哈顿是大多数车程的起点

这是不用任何包来绘制大量地理数据的基本方法

如果不喜欢图像处理，可以用datashader

我会在下个部分里展示在pygmaps上的1000个样例数据的图，他会生成一个HTML，用户需要在浏览器中打开它（？？）

# 定义一些方程来表明所给数据的特征

我们需要做一些特征工程，并找到哪些特征是会影响duration的。

下面正式开始特征提取。

```
start = time.time()
def haversine_(lat1, lng1, lat2, lng2):
    """function to calculate haversine distance between two co-ordinates"""
    lat1, lng1, lat2, lng2 = map(np.radians, (lat1, lng1, lat2, lng2))
    AVG_EARTH_RADIUS = 6371  # in km
    lat = lat2 - lat1
    lng = lng2 - lng1
    d = np.sin(lat * 0.5) ** 2 + np.cos(lat1) * np.cos(lat2) * np.sin(lng * 0.5) ** 2
    h = 2 * AVG_EARTH_RADIUS * np.arcsin(np.sqrt(d))
    return(h)

def manhattan_distance_pd(lat1, lng1, lat2, lng2):
    """function to calculate manhatten distance between pick_drop"""
    a = haversine_(lat1, lng1, lat1, lng2)
    b = haversine_(lat1, lng1, lat2, lng1)
    return a + b

import math
def bearing_array(lat1, lng1, lat2, lng2):
    """ function was taken from beluga's notebook as this function works on array
    while my function used to work on individual elements and was noticably slow"""
    AVG_EARTH_RADIUS = 6371  # in km
    lng_delta_rad = np.radians(lng2 - lng1)
    lat1, lng1, lat2, lng2 = map(np.radians, (lat1, lng1, lat2, lng2))
    y = np.sin(lng_delta_rad) * np.cos(lat2)
    x = np.cos(lat1) * np.sin(lat2) - np.sin(lat1) * np.cos(lat2) * np.cos(lng_delta_rad)
```

```
    return np.degrees(np.arctan2(y, x))

end = time.time()
print("Time taken by above cell is {}.".format((end-start)))
```

## 特征提取

```
start = time.time()
train_data = temp.copy()
train_data['pickup_datetime'] = pd.to_datetime(train_data.pickup_datetime)
train_data.loc[:, 'pick_month'] = train_data['pickup_datetime'].dt.month
train_data.loc[:, 'hour'] = train_data['pickup_datetime'].dt.hour
train_data.loc[:, 'week_of_year'] = train_data['pickup_datetime'].dt.weekofyear
train_data.loc[:, 'day_of_year'] = train_data['pickup_datetime'].dt.dayofyear
train_data.loc[:, 'day_of_week'] = train_data['pickup_datetime'].dt.dayofweek
train_data.loc[:,'hvsine_pick_drop'] = haversine_(train_data['pickup_latitude'].values,
train_data['pickup_longitude'].values, train_data['dropoff_latitude'].values,
train_data['dropoff_longitude'].values)
train_data.loc[:,'manhtn_pick_drop'] =
manhattan_distance_pd(train_data['pickup_latitude'].values,
train_data['pickup_longitude'].values, train_data['dropoff_latitude'].values,
train_data['dropoff_longitude'].values)
train_data.loc[:,'bearing'] = bearing_array(train_data['pickup_latitude'].values,
train_data['pickup_longitude'].values, train_data['dropoff_latitude'].values,
train_data['dropoff_longitude'].values)

end = time.time()
print("Time taken by above cell is {}.".format(end-start))
```

### 动画

提取日期时间特征后，我们可以看到接客时间对于duration的影响

比如在一些时刻中，大量的接客数量发生变化，这意味着那里发生了一些交通事故并且duration会变得长一些（？？）

绘制热点图并制作动画，对应于接客时间如何伴随交通变化

这里所做的是：对不同的接客时间，制作接客地点的热点图，然后生成一个gif

```
start = time.time()
def color(hour):
    """function for color change in animation"""
    return(10*hour)

def Animation(hour, temp, rgb):
    """Function to generate return a pic of plotings"""
    #ax.clear()
    train_data_new = temp.loc[temp['hour'] == hour]
    start = time.time()
    rgb = np.zeros((3000, 3500, 3), dtype=np.uint8)

    rgb[..., 0] = 0
```

```
    rgb[..., 1] = 0
    rgb[..., 2] = 0
    train_data_new['pick_lat_new'] = list(map(int, (train_data_new['pickup_latitude'] -
(40.6000))*10000))
    train_data_new['drop_lat_new'] = list(map(int, (train_data_new['dropoff_latitude'] -
(40.6000))*10000))
    train_data_new['pick_lon_new'] = list(map(int, (train_data_new['pickup_longitude'] -
(-74.050))*10000))
    train_data_new['drop_lon_new'] = list(map(int,(train_data_new['dropoff_longitude'] -
(-74.050))*10000))

    summary_plot = pd.DataFrame(train_data_new.groupby(['pick_lat_new', 'pick_lon_new'])
['id'].count())

    summary_plot.reset_index(inplace = True)
    summary_plot.head(120)
    lat_list = summary_plot['pick_lat_new'].unique()
    for i in lat_list:
        #print(i)
        lon_list = summary_plot.loc[summary_plot['pick_lat_new']==i]['pick_lon_new'].tolist()
        unit = summary_plot.loc[summary_plot['pick_lat_new']==i]['id'].tolist()
        for j in lon_list:
            #j = int(j)
            a = unit[lon_list.index(j)]
            #print(a)
            if (a//50) >0:
                rgb[i][j][0] = 255 - color(hour)
                rgb[i,j, 1] = 255 - color(hour)
                rgb[i,j, 2] = 0 + color(hour)
            elif (a//10)>0:
                rgb[i,j, 0] = 0 + color(hour)
                rgb[i,j, 1] = 255 - color(hour)
                rgb[i,j, 2] = 0 + color(hour)
            else:
                rgb[i,j, 0] = 255 - color(hour)
                rgb[i,j, 1] = 0 + color(hour)
                rgb[i,j, 2] = 0 + color(hour)
    #fig, ax = plt.subplots(nrows=1,ncols=1,figsize=(14,20))
    end = time.time()
    print("Time taken by above cell is {} for {}.".format((end-start), hour))
    return(rgb)
end = time.time()
print("Time taken by above cell is {}.".format(end -start))
```

```
start = time.time()
images_list=[]
train_data_new['pickup_datetime'] = pd.to_datetime(train_data_new.pickup_datetime)
train_data_new.loc[:, 'hour'] = train_data_new['pickup_datetime'].dt.hour

for i in list(range(0, 24)):
    im = Animation(i, train_data_new, rgb.copy())
    images_list.append(im)
end = time.time()
print("Time taken by above cell is {}.".format(end -start))
```

```
start = time.time()
def build_gif(imgs = images_list, show_gif=False, save_gif=True, title=''):
    """function to create a gif of heatmaps"""
    fig, ax = plt.subplots(nrows=1,ncols=1,figsize=(10,10))
    ax.set_axis_off()
    hr_range = list(range(0,24))
    def show_im(pairs):
        ax.clear()
        ax.set_title('Absolute Traffic - Hour ' + str(int(pairs[0])) + ':00')
        ax.imshow(pairs[1])
        ax.set_axis_off()
    pairs = list(zip(hr_range, imgs))
    #ims = map(lambda x: (ax.imshow(x), ax.set_title(title)), imgs)
    im_ani = animation.FuncAnimation(fig, show_im, pairs,interval=500, repeat_delay=0,
blit=False)
    plt.cla()
    if save_gif:
        im_ani.save('animation.gif', writer='imagemagick') #, writer='imagemagick'
    if show_gif:
        plt.show()
    return
end = time.time()
print("Time taken by above cell is {}".format(end-start))
```

```
start = time.time()
build_gif()
end = time.time()
print(end-start)
```

```
filename = 'animation.gif'
video = io.open(filename, 'r+b').read()
encoded = base64.b64encode(video)
HTML(data='''<img src="{0}" type="gif"
/>'''.format(encoded.decode('ascii')))
```

**发现** 2.am-7.am需求最少；7.am-4.pm，需求数量缓；5.pm-1.pm需求数量巨大

右上角那是JFK机场，分布一样，但2.am-6.am人更少