MLPractice-01 神舟优车订单预测

讲义

1. 任务描述

在网约车业务中，由于系统中的专车资源是有限的，而乘客资源也是有限的，但是这两类有限的资源是各自散布在一个城市的若干区域中，并且由于城市的不同区域的功能不同，乘客资源散布在各个区域的密度有所差别，如何使专车的散布区域密度尽量符合乘客散布密度是其中很重要的一个需求。因此我们提出这样的需求，预测某个时间段内从区域A出发到区域B的订单数量。

现在给定2017年7月份和8月份的部分订单数据作为训练数据，预测8月份规定时间段内的数据。

2. 数据集

2.1. 训练集

我们以七月份的订单数据作为训练集，其中包含的各个字段的含义如下：

字段名称	字段含义	数据示例
id	订单id的hash值	583411b46a31bcc5d12d4402c928a146
driver_id	司机id的hash值（未出行成功则为一个特殊司机编号）	3e69e17a6e5a726fe44d71896bee4f32
menber_id	乘客id的hash值	6b4d6e4992191fe96b9f27921520d551
create_date	订单创建日期	2017-07-01

字段名称	字段含义	数据示例
create_hour	订单创建时间（0-23小时）	00
status	订单状态：0是未预约成功，1是预约取消，2是出行成功	2
estimate_money	预估行程金额（元）	140.00
estimate_distance	预估行程距离（米）	20099.00
estimate_term	预估行程时间（分钟）	18.00
start_geo_id	起点区域id的hash值	6d7827e8dcfa09497954a31e6f7e6ee6
end_geo_id	终点区域id的hash值	85e49ded1fa70a7bfa01ab0212a6e538

见附件：train_July.csv

2.2. 测试集

我们以八月份第一周内随机抽取的若干条数据作为测试集。测试集字段含义如下所示：

字段名称	字段含义	数据示例
test_id	该条测试用例的id	1
start_geo_id	起点区域id的hash值	6d7827e8dcfa09497954a31e6f7e6ee6
end_geo_id	终点区域id的hash值	85e49ded1fa70a7bfa01ab0212a6e538
create_date	订单创建日期	2017-08-01
create_hour	订单创建时间（0-23小时）	01

见附件：test_Aug.csv

2.3. 其他数据

2.3.1. 区域内设施数据

包含某一区域内公共设施的数量描述，其中第一列为区域id的hash值（与训练及测试数据中的区域对应），后面分别为各类设施的类型及数量，例如：

```
3d99665144344fc090b5b7450ffe72f5,加油站,4,超市,43,住宅区,151,地铁站,4,公交站,36,咖啡厅,22,中餐厅,597,ATM,59,写字楼,47,酒店,45
```

表示该区域内有加油站4个，超市43个，等等。

见附件：poi.csv

2.3.2. 天气数据

训练集与测试集所涉及的时间段内的天气情况（有极少部分缺失）。各个字段含义如下：

字段名称	字段含义	数据示例
date	日期（精确至半小时）	2017-7-10:30
text	天气现象文字	晴
code	天气现象代码	1
temperature	温度	29
feels_like	体感温度	28
pressure	气压	998
humidity	相对湿度	62
visibility	能见度	9.3
wind_direction	风向文字	南
wind_direction_degree	风向角度，范围0-360，0为正北，90为正东，180为正南，270为正西	200
wind_speed	风速	8
wind_scale	风力等级	2

见附件：weather.csv

3. 解决过程

在尝试解决一个问题之前，首先需要明确我们有哪些资源，然后考虑如何利用这些资源解决问题，因此第一步是将我现有的数据，特别是各个表头，表之间的关系梳理一下，然后再考虑要解决什么问题。

3.1. 目标是什么？

首先，该问题是一个回归问题，即根据测试集给定的字段，包括起点、重点、日期、小时来预测对应的订单数量。初一看应该是以这四个字段作为输入，作为预测的条件。那么对应的训练数据也应该与这四个字段对应。因此我们到训练集中去找到这四个对应的字段，看看有没有。

我们可以很容易的就在七月份的订单表中找到如下的几个字段，与测试集的输入相对应：

id	create_date	create_hour	start_geo_id	end_geo_id
d91bda...	2017-08-01	12	25998a0...	4176d25...
844444...	2017-08-01	12	d42290f...	7469adf...
67eb5b...	2017-08-01	14	2da36a4...	2da36a4...
28cb07...	2017-08-01	14	56b85fc...	56b85fc...

但是，有一个问题。我们知道回归问题作为监督学习的一种，训练集数据除了有输入特征之外，还要有标签，对应到这个问题就应该是特定的时间段内，对应的一组起点和终点的订单数量。但是在这个测试集中，我们并没有找到一个字段是用来描述某个时间段内的订单数量的字段，那怎么办呢？

考虑下，这个数据集中记录的是七月份所有的订单，每个订单一条数据，也就是说这个标签需要我们自己统计一下某个时间段内，从区域A到区域B的订单数量。

根据这个思路我们应该已经能够获取到训练集了。

3.2. 真正参与计算的特征？

当我们拿到看似完整的训练集之后，开始着手训练模型时，会发现一个问题，训练集中的

这四个特征，与我们需要计算的目标之间存在什么直接的关联么？

比如起点这个字段，是一个地区的编号，应该与订单数量是没有直接的关联的，而是这个地区背后所表示的含义与订单的数量有关系。比如说这个地区内是否有很多住宅区？是否有很多写字楼？是否有很多商业区等等。

比如日期这个字段，日期直接与订单数量存在什么必然的联系么？一般不会，而是日期所表示的各方面含义与订单数量有关系。比如某个日期对应的天气状况。

我们需要将这几个表面特征转换为其背后所表示的各方面数据，然后分析这些数据与订单数量之间的关系，以此来训练模型。

3.2.1. 地区所表示的含义

根据之前的思路，现在要做的是将训练集中的地区对应的特征数据表示出来，此处我们结合MySQL来实现。首先我们创建一张训练集表，将训练集数据导入该表中，然后将对应的公共设施数据导入一张天气表中，最后利用SQL语句连接数据，形成新表。

3.2.1.1. 创建训练集表

```
CREATE TABLE `t_train` (
  `order_id` varchar(45) NOT NULL,
  `driver_id` varchar(45) DEFAULT NULL,
  `member_id` varchar(45) DEFAULT NULL,
  `create_date` varchar(45) DEFAULT NULL,
  `create_hour` int(11) DEFAULT NULL,
  `status` int(11) DEFAULT NULL,
  `estimate_money` decimal(12,4) DEFAULT NULL,
  `estimate_distance` decimal(12,4) DEFAULT NULL,
  `estimate_term` decimal(12,4) DEFAULT NULL,
  `start_geo_id` varchar(45) DEFAULT NULL,
  `end_geo_id` varchar(45) DEFAULT NULL,
  PRIMARY KEY (`order_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

3.2.1.2. 导入训练集

首先利用`csv`模块将数据从CSV文件中读取出来：

```
import csv

def TRAIN_read(fileName):
    csv_reader = csv.reader(open(fileName, encoding='utf-8'))
    trainSet = []
```

```
    for row in csv_reader:
        item = {}
        item["order_id"] = row[0]
        item["driver_id"] = row[1]
        item["member_id"] = row[2]
        item["create_date"] = row[3]
        item["create_hour"] = row[4]
        item["status"] = row[5]
        item["estimate_money"] = row[6]
        item["estimate_distance"] = row[7]
        item["estimate_term"] = row[8]
        item["start_geo_id"] = row[9]
        item["end_geo_id"] = row[10]
        trainSet.append(item)

    trainSet.pop(0)
    return trainSet
```

这个函数接受一个参数是训练集所在的文件名称。

利用`PyMySQL`模块连接数据库，并将数据导入数据库：

```
import pymysql

def getConnection():
    conn = pymysql.connect(
        host='127.0.0.1',
        port=3306,
        user='root',
        passwd='1234',
        db='szyc',
        charset='utf8')
    return conn

def Train_import(data):
    conn = getConnection()
    cursor = conn.cursor()

    count = 0

    for item in data:
        sql = 'insert into t_train (order_id, driver_id, member_id, cre
ate_date, ' \
```

```
                'create_hour, status, estimate_money, estimate_distance, estimate_term, ' \
                'start_geo_id, end_geo_id) values ' \
                '(\'%s\', \'%s\', \'%s\', \'%s\', %s, %s, %s, %s, %s, \'%s\', \'%s\')' \
                % (item["order_id"], item["driver_id"], item["member_id"],
                   item["create_date"], item["create_hour"], item["status"],
                   item["estimate_money"], item["estimate_distance"], item["estimate_term"],
                   item["start_geo_id"], item["end_geo_id"])

        count += 1
        cursor.execute(sql)

        if count % 500 == 0:
            conn.commit()

    conn.commit()
    cursor.close()
    conn.close()
```

然后导入数据：

3.2.1.3. 导入公共设施数据

同样的思路创建表并导入：

```
CREATE TABLE `t_poi` (
  `rid` varchar(45) NOT NULL,
  `petrol` int(11) NOT NULL DEFAULT '0',
  `market` int(11) NOT NULL DEFAULT '0',
  `uptown` int(11) NOT NULL DEFAULT '0',
  `metro` int(11) NOT NULL DEFAULT '0',
  `bus` int(11) NOT NULL DEFAULT '0',
  `cafe` int(11) NOT NULL DEFAULT '0',
  `restruant` int(11) NOT NULL DEFAULT '0',
  `atm` int(11) NOT NULL DEFAULT '0',
  `office` int(11) NOT NULL DEFAULT '0',
  `hotel` int(11) NOT NULL DEFAULT '0',
  PRIMARY KEY (`rid`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

3.2.1.4. 连接数据

利用SQL中的连接查询，将地区与对应的设施连接起来：

```
select d.*,
    p.petrol s_petrol, p.market s_market, p.uptown s_uptown, p.metro s_metro,
    p.bus s_bus, p.cafe s_cafe, p.restruant s_restruant, p.atm s_atm,
    p.office s_office, p.hotel s_hotel
from t_train d, t_poi p
where d.start_geo_id = p.rid
```

3.2.1.5. 以同样的方式对目的地的数据进行同样的连接

```
select temp1.*,
    p2.petrol e_petrol, p2.market e_market, p2.uptown e_uptown, p2.metro e_metro, p2.bus e_bus, p2.cafe e_cafe,
    p2.restruant e_restruant, p2.atm e_atm, p2.office e_office, p2.hotel e_hotel
from t_poi p2,
    (select d.*,
        p.petrol s_petrol, p.market s_market, p.uptown s_uptown, p.metro s_metro, p.bus s_bus, p.cafe s_cafe,
        p.restruant s_restruant, p.atm s_atm, p.office s_office, p.hotel s_hotel
    from t_train d, t_poi p
    where d.start_geo_id = p.rid) as temp1
where p2.rid = temp1.end_geo_id
```

3.2.2. 日期所表示的天气

根据之前的思路，现在要做的是将训练集中的日期对应的天气数据表示出来，此处我们结合MySQL来实现。首先我们创建一张训练集表，将训练集数据导入该表中，然后将对应的天气数据导入一张天气表中，最后利用SQL语句连接数据，形成新表。

3.2.2.1. 导入天气数据

以同样的方式导入天气数据。首先创建天气表：

```
CREATE TABLE `t_weather` (
  `wid` varchar(45) NOT NULL,
  `date` varchar(12) DEFAULT NULL,
  `hour` int(11) DEFAULT '0',
```

```
  `min` int(11) DEFAULT '0',
  `text` varchar(45) DEFAULT NULL,
  `code` int(11) DEFAULT '0',
  `temperature` int(11) DEFAULT '0',
  `feels_like` int(11) DEFAULT '0',
  `pressure` int(11) DEFAULT '0',
  `humidity` int(11) DEFAULT '0',
  `visibility` decimal(10,2) DEFAULT '0.00',
  `wind_direction` varchar(45) DEFAULT NULL,
  `wind_direction_degree` int(11) DEFAULT '0',
  `wind_speed` decimal(10,2) DEFAULT '0.00',
  `wind_scale` int(11) DEFAULT '0',
  PRIMARY KEY (`wid`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

利用与导入训练集相似的方法将天气数据导入数据库：

wid	date	hour	min	text	code	temperature	feels_like	pressure	humidity	visibility	wind_direction	wind_direction_degree	wind_speed	wind_scale
0fbd1b9a-0ae3-11e8-9c6a-3c970e0...	2017-7-1	0	30	晴	1	29	28	998	62	9.30	南	200	9.00	2
0fbd42a8-0ae3-11e8-b862-3c970e0...	2017-7-1	1	0	晴	1	28	28	998	63	7.70	西南	233	7.92	2
0fbd42a9-0ae3-11e8-939f-3c970e0...	2017-7-1	1	30	晴	1	28	28	998	65	7.70	南	197	7.20	2
0fbd42aa-0ae3-11e8-9049-3c970e0...	2017-7-1	2	0	晴	1	28	27	998	66	7.70	西南	233	9.00	2
0fbd42ab-0ae3-11e8-9948-3c970e0...	2017-7-1	2	30	晴	1	28	27	998	67	6.60	南	180	7.20	2
0fbd42ac-0ae3-11e8-89d0-3c970e0...	2017-7-1	3	0	晴	1	28	27	998	67	6.60	南	197	10.08	2
0fbd42ad-0ae3-11e8-b4cf-3c970e00...	2017-7-1	3	30	晴	1	28	27	998	67	6.60	南	188	7.20	2
0fbd42ae-0ae3-11e8-9eec-3c970e0...	2017-7-1	4	0	晴	1	27	27	998	67	6.00	西南	239	9.00	2
0fbd42af-0ae3-11e8-8c3e-3c970e00...	2017-7-1	4	30	晴	1	27	27	998	67	6.00	西南	239	9.00	2
0fbd42b0-0ae3-11e8-8ad6-3c970e0...	2017-7-1	5	0	晴	1	27	26	998	72	5.50	南	180	5.76	2
0fbd42b1-0ae3-11e8-b52b-3c970e0...	2017-7-1	5	30	晴	0	27	26	998	70	4.30	南	194	6.84	2
0fbd42b2-0ae3-11e8-9680-3c970e0...	2017-7-1	6	30	晴	0	26	26	999	71	3.80	南	160	5.40	1
0fbd42b3-0ae3-11e8-8243-3c970e0...	2017-7-1	7	0	晴	0	26	26	999	72	3.80	东南	149	3.96	1

3.2.1.4. 连接数据

利用SQL中的连接查，然后将这个最终合并的结果形成一个新表：

```
create table t_train_combine
select tt.*,
    tw.temperature w_temperature, tw.feels_like w_feels_like, tw.pressure w_pressure, tw.humidity w_humidity,
    tw.visibility w_visibility, tw.wind_direction_degree w_wind_direction_degree, tw.wind_speed w_wind_speed
from
(
    select date, hour, avg(temperature) temperature, avg(feels_like) feels_like, avg(pressure) pressure,
    avg(humidity) humidity, avg(visibility) visibility, avg(wind_direction_degree) wind_direction_degree,
    avg(wind_speed) wind_speed
    from t_weather group by date, hour
```

```
) as tw,
(
    select temp1.*,
        p2.petrol e_petrol, p2.market e_market, p2.uptown e_uptown, p2.metro e_metro, p2.bus e_bus, p2.cafe e_cafe,
        p2.restruant e_restruant, p2.atm e_atm, p2.office e_office, p2.hotel e_hotel
    from t_poi p2,
        (select d.*,
            p.petrol s_petrol, p.market s_market, p.uptown s_uptown, p.metro s_metro, p.bus s_bus, p.cafe s_cafe,
            p.restruant s_restruant, p.atm s_atm, p.office s_office, p.hotel s_hotel
        from t_train d, t_poi p
        where d.start_geo_id = p.rid) as temp1
    where p2.rid = temp1.end_geo_id
) as tt
where date_format(tt.create_date, '%Y-%m-%d') = date_format(tw.date, '%Y-%m-%d')
and tt.create_hour = tw.hour
```

3.3. 如何得到标签？

之前的思路中，我们是统计给定时间段内从某个区域到某个区域的订单数量。但是现在我们将时间以及区域分别表示成其背后对应的含义了，因此，此时我们重新进行统计时，统计的数量应该是基于天气状况与地区的各类特征的组合进行统计。

```
create table t_train_real
SELECT s_petrol, s_market, s_uptown, s_metro, s_bus,
    s_cafe, s_restruant, s_atm, s_office, s_hotel,
    e_petrol, e_market, e_uptown, e_metro, e_bus,
    e_cafe, e_restruant, e_atm, e_office, e_hotel,
    w_temperature, w_feels_like, w_pressure, w_humidity,
    w_visibility, w_wind_direction_degree, w_wind_speed,
    count(order_id) order_count
FROM t_train_combine
group by s_petrol, s_market, s_uptown, s_metro, s_bus, s_cafe,
    s_restruant, s_atm, s_office, s_hotel,
    e_petrol, e_market, e_uptown, e_metro, e_bus,
    e_cafe, e_restruant, e_atm, e_office, e_hotel,
    w_temperature, w_feels_like, w_pressure, w_humidity,
    w_visibility, w_wind_direction_degree, w_wind_speed
```

3.4. 训练模型

如此，经过上面一系列的步骤，我们最终得到了可以用来训练模型的训练集，接下来利用sklearn训练模型。首先可以利用`Pandas`直接从数据库加载数据：

```
import pymysql
import pandas as pd

def getConnection():
    conn = pymysql.connect(
        host='127.0.0.1',
        port=3306,
        user='root',
        passwd='1234',
        db='szyc',
        charset='utf8')
    return conn

train = pd.read_sql(sql="select * from t_train_real",
con=getConnection())
```

拆分训练集，将特征和标签进行拆分：

```
x = train.drop(labels="order_count", axis=1)
y = train[["order_count"]]
```

使用sklearn训练模型：

```
from sklearn.linear_model import LinearRegression

lr = LinearRegression()
lr.fit(x, y)
```

3.5. 预测验证

在利用训练好的模型对测试数据进行预测时，需要将测试数据进行相同的处理，即将日期和地区转换为其背后的含义。

3.5.1. 导入测试数据

首先将测试数据导入数据库：

```
CREATE TABLE `t_aug_test` (
  `test_id` INT NOT NULL,
  `start_geo_id` VARCHAR(45) NULL,
  `end_geo_id` VARCHAR(45) NULL,
  `create_date` VARCHAR(45) NULL,
  `create_hour` INT NULL,
  `count` INT,
  PRIMARY KEY (`test_id`));
```

3.5.2. 合并测试数据

以同样的方式将日期背后的天气、地区背后的公共设施数据合并成一张表，不过需要注意此处使用左连接，这样可以确保有些地区或者日期对应的公共设施或天气不存在时，不会丢失测试数据。

```
create table t_aug_test_real
select tt.*,
    tw.temperature w_temperature, tw.feels_like w_feels_like, tw.pressure w_pressure, tw.humidity w_humidity,
    tw.visibility w_visibility, tw.wind_direction_degree w_wind_direction_degree, tw.wind_speed w_wind_speed
from
(
    select temp1.*,
        p2.petrol e_petrol, p2.market e_market, p2.uptown e_uptown, p2.metro e_metro, p2.bus e_bus, p2.cafe e_cafe,
        p2.restruant e_restruant, p2.atm e_atm, p2.office e_office, p2.hotel e_hotel
    from
        (select d.*,
            p.petrol s_petrol, p.market s_market, p.uptown s_uptown, p.metro s_metro, p.bus s_bus, p.cafe s_cafe,
            p.restruant s_restruant, p.atm s_atm, p.office s_office, p.hotel s_hotel
        from t_aug_test d
        left join t_poi p
        on d.start_geo_id = p.rid) as temp1
    left join t_poi p2
    on p2.rid = temp1.end_geo_id
```

```
) as tt
left join
(
    select date, hour, avg(temperature) temperature, avg(feels_like) fe
els_like, avg(pressure) pressure,
    avg(humidity) humidity, avg(visibility) visibility,
avg(wind_direction_degree) wind_direction_degree,
    avg(wind_speed) wind_speed
    from t_weather group by date, hour
) as tw
on date_format(tt.create_date, '%Y-%m-%d') = date_format(tw.date, '%Y-
%m-%d')
and tt.create_hour = tw.hour
```

3.5.3. 验证

读取测试数据进行验证：

```
test = pd.read_sql(sql="select * from t_aug_test_real",
con=getConnection())
x_test = test[["s_petrol", "s_market", "s_uptown", "s_metro", "s_bus",
"s_cafe",
               "s_restruant", "s_atm", "s_office", "s_hotel",
"e_petrol", "e_market",
               "e_uptown", "e_metro", "e_bus", "e_cafe", "e_restruant",
"e_atm",
               "e_office", "e_hotel", "w_temperature", "w_feels_like",
"w_pressure",
               "w_humidity", "w_visibility", "w_wind_direction_degree",
"w_wind_speed"]]

y_test = test[["count"]]

print(lr.score(x_test, y_test))
```