机器学习纳米学位

毕业项目

1. 问题的定义

1.1 项目概述

这个项目是2015年由Rossmann药店在Kaggle上给出的赛题，要求对旗下的1115家药店的日销售量进行预测。所提供的数据集包括`train.csv`，`test.csv`，`store.csv`，其中：

- `train.csv`中包含以下信息：
  - `Store`：药店编号
  - `DayOfWeek`：该日期对应的星期
  - `Date`：日期
  - `Sales`：日销售量 （`test.csv`中需要预测的标签）
  - `Customers`：日客流数
  - `Open`：该天是否营业 （0=关店，1=开店）
  - `Promo`：该天该门店是否正在促销活动
  - `StateHoliday`：该天是否为法定假日（分为：a, b, c, 0）
  - `SchoolHoliday`：改天是否为学校放假日
- `store.csv`中包含以下信息：
  - `Store`：药店编号
  - `StoreType`：药店类型（分为：a, b, c, d）
  - `Assortment`：商品组合（分为：a, b, c）
  - `CompetitionDistance`：与竞争对手的距离
  - `CompetitionOpenSinceMonth`：竞争对手开店的月份
  - `CompetitionOpenSinceYear`：竞争对手开店的年份
  - `Promo2` 该点是否正在参与优惠券促销活动（即每个季度该店会向附近居民邮寄优惠券，一般都是全品打折，有效期是3个月）
  - `Promo2SinceWeek`：连续促销活动从那周开始
  - `Promo2SinceYear`：促销活动开始的年份
  - `PromoInterval`：促销活动间隔周期
- `test.csv`不包含`train.csv`中的`Sales`,`Customers`两列，其余都相同

1.2 问题陈述

这个项目的问题是一个典型的监督学习中的回归问题，通过算法将已知的信息与标签（Sales）进行映射，使得我们可以通过知识的信息就可以去预测标签（Sales）

对此这个项目使用以下策略：

- 对数据进行清理
- 对数据进行探索性分析，并进行初步的特征选择，特征重构

- 使用合适的算法进行训练，优化
- 将训练好的模型去预测`test.csv`中的Sales并将其上传至Kaggle查看得分

我们希望上传到Kaggle的得分，对于RMSPE评分标准是得分越低，模型越优秀。

1.3 评价指标

这里采用与该比赛项目一致的评价指标（Root Mean Square Percentage Error），以便我们之后将`test.csv`上传至Kaggle：

$$\mathbf{RMSPE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(\frac{y_i - \hat{y}_i}{y_i}\right)^2},$$

y_i 是标签（Sales），$\hat{y}_i$ 是我们通过已知信息预测的标签，采用RMSPE的好处：

- 在RMSE的基础上，采用 $\frac{y_i-\hat{y}_i}{y_i}$ （Percentage）可以将每个数据所产生的误差去刚量化，即每个数据的误差都具有一样的话语权。因此这里无需对Outlier进行特殊处理，因为在拟合过程中Outlier对Loss function的影响和其他值的影响是一样的。

采用该评价指标等于间接希望模型能对所有数值的Sales亦有较好的拟合。

2. 分析

2.1 概览

`train.csv`中数据信息：

```
Store            1017209 non-null int64
DayOfWeek        1017209 non-null int64
Date             1017209 non-null object
Sales            1017209 non-null int64
Customers        1017209 non-null int64
Open             1017209 non-null int64
Promo            1017209 non-null int64
StateHoliday     1017209 non-null object
SchoolHoliday    1017209 non-null int64
```

在训练数据中是没有缺失值的，下面来看下`store.csv`中的数据信息：

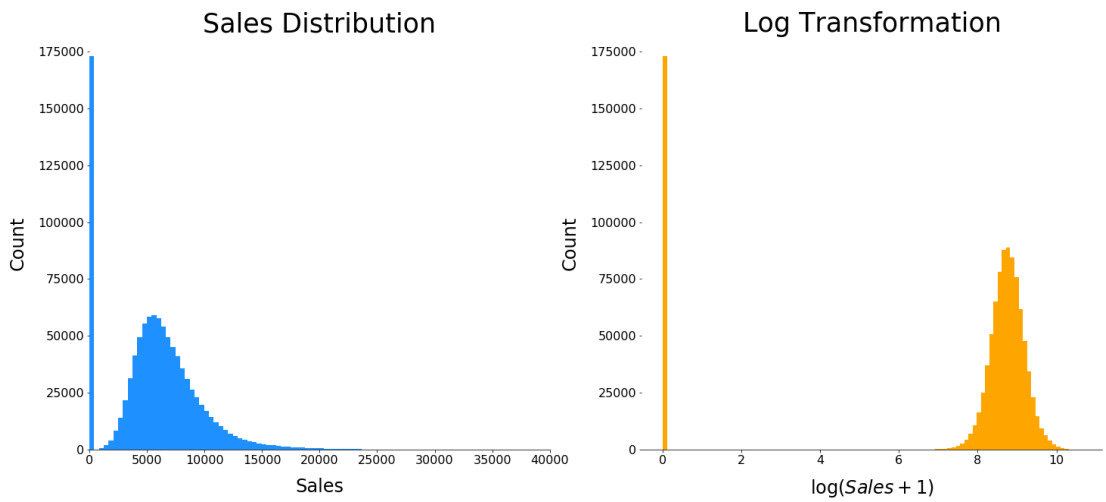
```
Store                        1115 non-null int64
StoreType                    1115 non-null object
Assortment                   1115 non-null object
CompetitionDistance          1112 non-null float64
CompetitionOpenSinceMonth    761 non-null float64
CompetitionOpenSinceYear     761 non-null float64
Promo2                       1115 non-null int64
Promo2SinceWeek              571 non-null float64
Promo2SinceYear              571 non-null float64
PromoInterval                571 non-null object
```

可以发现里面有几列信息存在缺失值，这也是后续我们需要对数据进行处理的主要方向

2.2 探索性可视化

2.2.1 Sales:

首先观察下我们需要拟合的标签（Sales）的分布情况：



- 左图是`train.csv`中Sales的分布情况，我们可以看出Sales在0处频数最多，若将0值的数据去除，分析其他数据的分布特征发现：

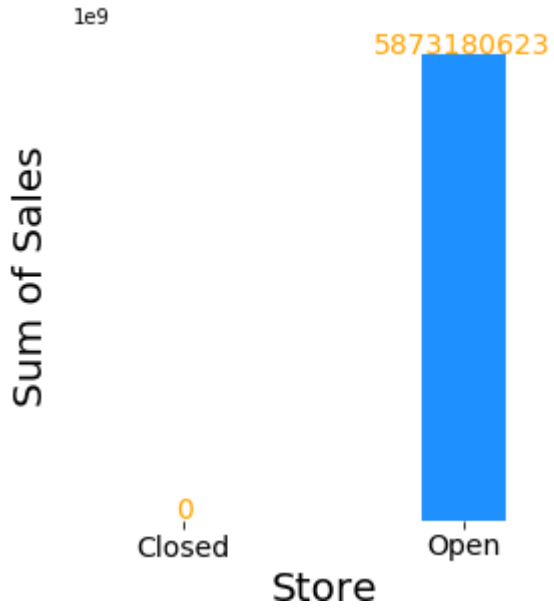
```
>>>trainset.loc[trainset['Sales']>0,'Sales'].skew()
1.5949288357537479 # Sales的偏度
>>>np.log(train_raw.loc[train_raw['Sales']>0,'Sales']).skew()
-0.10968460576537623 # log(Sales)的偏度
```

左图中发现，Sales在0处存在最多的频数，非0值的数据呈现右偏分布，通过计算发现Sales的偏度：1.59，为正偏分布，存在拖尾的特性。

右图为对Sales进行对数转换后的分布，可以发现其分布非常接近正态分布，通过计算发现其偏度：-0.11，其偏斜可以忽略近似看成正态分布，非常适合用线性回归进行拟合分析。

2.2.2 Open:

Open信息是一个非常重要的信息，如下图所示：



这里将所有`train.csv`中根据Open信息进行分类发现：只要门店关的就肯定没有日销售额，这点可以帮助我们快速的做出一个预测的决策：

1. 只要`Open=0`则`Sales=0`。
2. 在算法只对`Open=1`的数据进行训练，评估。
3. 在`test.csv`只针对`Open=1`门店的日销售额进行预测，对于`Open=0`的数据直接预测为**0**

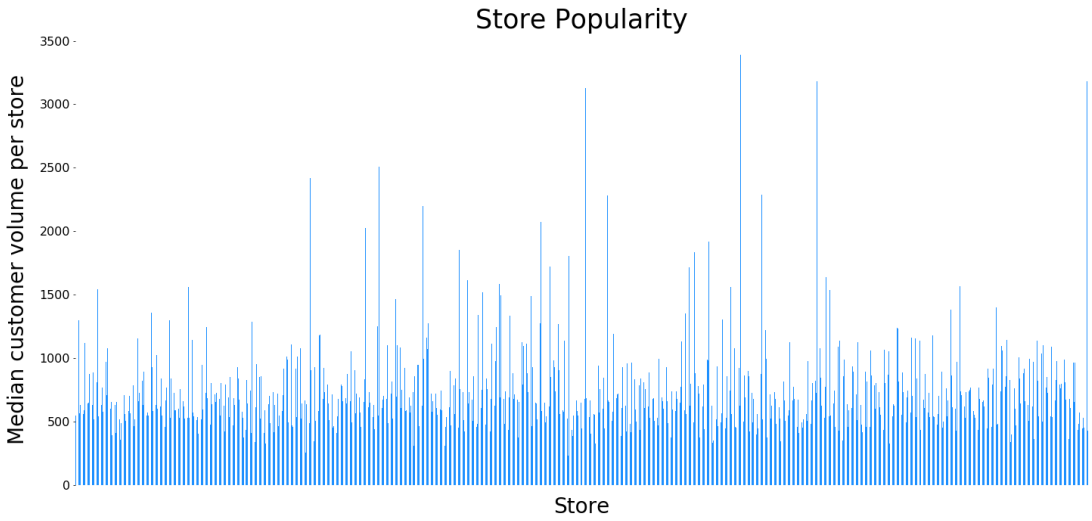
因此这里不将Open选为模型训练用的特征，即之后的模型只针对开店营业的信息进行训练，而对于`Open=1`的数据直接采用上述决策进行处理。

2.2.3 Store Popularity:

Store特征为门店编号，虽然能很好的将不同门店进行区分，以便更准确的预测，只是若是设置成哑变量则会生成**1115**个特征，这么高维的特征是非常不值得的，又不能直接以序列方式放入一个特征里（因为门店之间的差距不是等差的）。

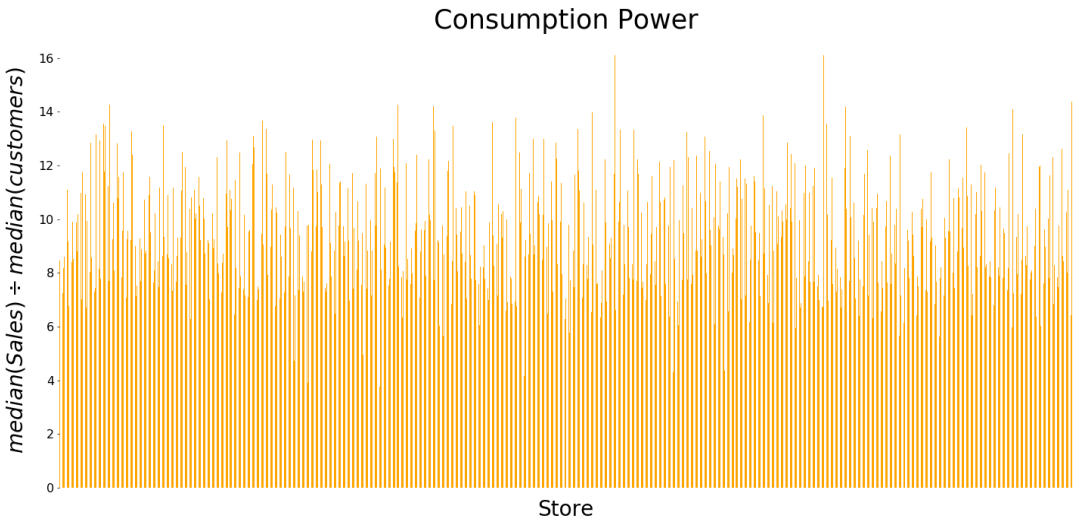
这里通过结合Customer和Store，去生成一个新的特征来试图捕捉这两个信息中包含类似门店“受欢迎度”的信息。操作如下：

- 在开店营业时`Open=1`的数据进行统计
- 将数据按门店分组，找出每个门店每日客户数的中间值（采用中间值可以防止该新特征受异常值影响而无法正确代表该门店的信息）
- 以这个中间值作为该门店的Store_popularity并合并进`store.csv`。



2.2.4 Consumption Power

这里希望收集每家店周边居民的信息，通过每家店的日销售中位数 / 日客流量中位数 --- 来创建消费者购买力(Consumption Power)，统计数据和Store Popularity相同只考虑开店营业的数据：



2.2.5 Sales数据中的异常值

当把关店数据拍出后发现，部分数据仍出现`Sales=0`的异常情况：

```
# trainset中开店营业数据中Sales为0的数据量
>>>train_merge_nozero[train_merge_nozero['Sales']==0].shape[0]
54
# trainset中所有开店营业的数据量
>>>train_merge_nozero.shape[0]
844392
```

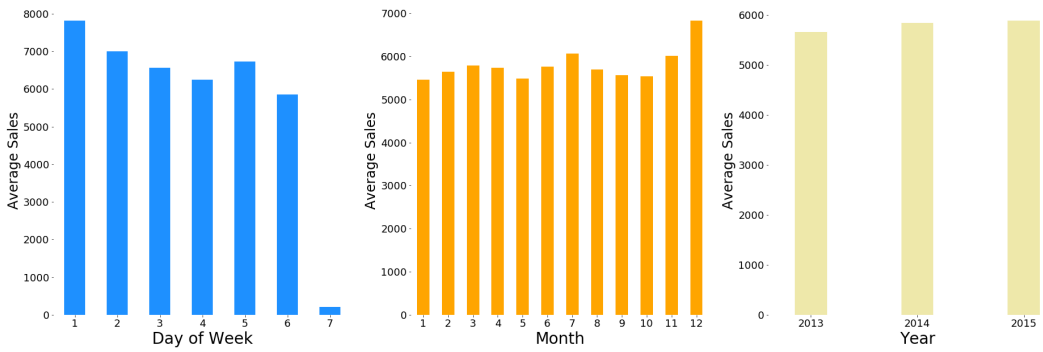
发现，这部分异常数据只占0.0064%，故在这里予以去除。

2.2.6 DayofWeek, Month, Year:

与日期相关，常规会有以下几种特性影响日销售量：

- 星期（DayofWeek）：一周中不同星期会对销售量产生影响
- 月份（Month）：一年中不同月份会对销售量产生影响
- 年份（Year）：不同年份会对销售量产生影响

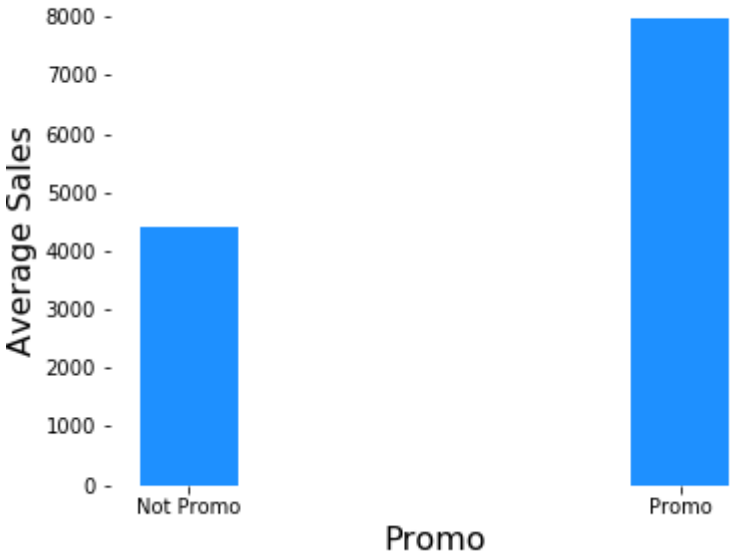
下图为集中特征的平均销售量：



Month，Year是从原先的Date中提取出来的，这里选择DayofWeek，Month，Year作为模型需要的特征

2.2.7 Promo

查看下在有无促销活动下，所有门店的平均销售量：



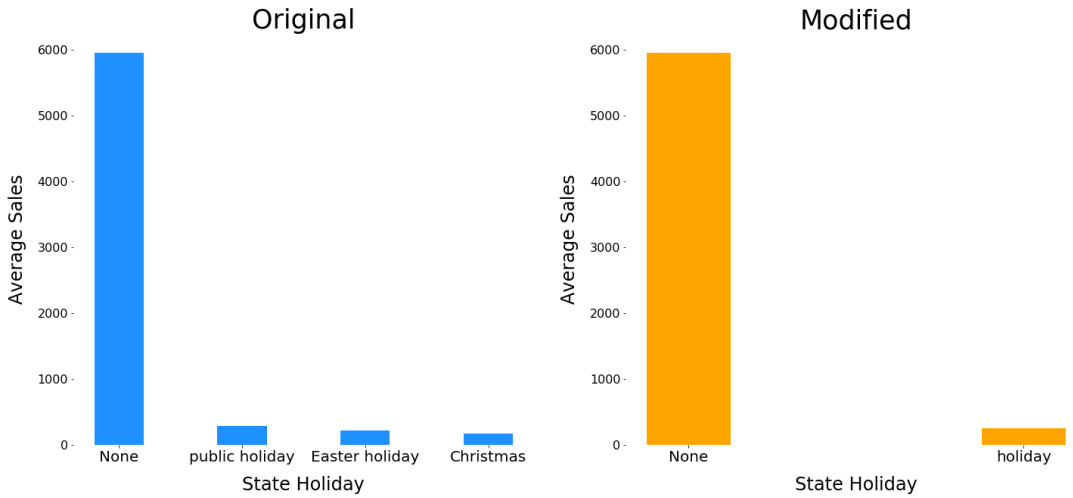
可以发现有无促销对销售量影响非常大，这里将Promo选为特征

2.2.8 StateHoliday

首先在对StateHoliday分组时发现该列数据需要清理：

```
>>>trainset['StateHoliday'].unique()
array(['0', 'a', 'b', 'c', 0], dtype=object)
```

发现StateHoliday特征中表示当天不是节日的**0**，在数据中对应这两个状态[**0** , `'0'`]，这里先对其进行简单清理后再查看其中各类对Sales的影响：



发现数据中提供了三种节日，其特点都十分相似，这里将这三个节日进行合并，把StateHoliday特征变成**(0 , 1)**分类特征。

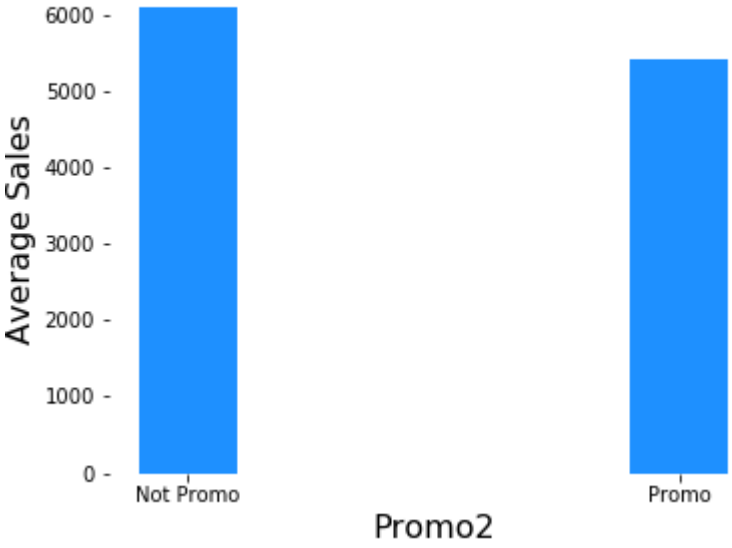
2.2.9 `train.csv`中的其他信息

SchoolHoliday也会对销售额产生影响，予以保留。Customers信息因在预测阶段无法获得故在此予以舍去。

2.2.10 `store.csv`中的信息：

我们发现`train.csv`和`store.csv`可以通过相同列Store将信息关联起来，从而使得`train.csv`中的信息更加完善，`store.csv`中的信息对于我们进行日销售量（Sales）的预测会非常有帮助，因为每家店的情况都不相同，在导入了`store.csv`中的信息后，我们可以的不同门店的特点有更好的把握。

- 选为特征的信息：
    - StoreType，Assortment作为门店特点的区分予以保留
    - CompetitionDistance显示每家门店竞争对手的信息，也会对该店的销售量产生影响，予以保留。（我们发现该特征有三个缺省值），这里因为后续选用的算法可以处理缺省值，故无需丢弃缺省值，或进行填充
    - Promo2可以提供该点是否参与连续的促销活动，我们从下图可以看出该特征的确对销售额产生了影响，需要保留。（可以发现长期促销对销售额反而会有负面影响）



- 选择舍弃的信息：
  - CompetitionOpenSinceMonth，CompetitionOpenSinceYear缺失值比例占31.75%，同时其信息只是作为对CompetitionDistance的补充，故不将其选为特征
  - Promo2SinceWeek，Promo2SinceYear，PromoInterval缺失值比例占48.79%,同时其信息只是作为对Promo2的补充，故不将其选为特征

最后选定的特征集为：`Promo`，`StateHoliday`，`SchoolHoliday`，`DayOfWeek`，`Month`，`Year`，`CompetitionDistance`，`Promo2`，`store_popularity`，`StoreType`，`Assortment`

2.3 算法和技术

在上述分析中发现特征中有较多的分类特征，这里可以使用$Gradient\ Boosting$构建K个回归树f_k（回归树能很好的处理分类变量与连续变量，又可以处理缺失值），回归方法使用$Linear\ Regression$，使用线性回归的依据以在Sales中阐明。

然后用$\hat{y_i} = \sum_{k=1}^{K} \alpha_k \cdot f_k(x_i)$来作为Sales的预测值，即用多个表现较弱的线性回归模型集成，以达到更好的预测效果。

2.4 基准模型

这里的基准模型设定为：若该问题用传统统计的广义线性回归模型处理的化，相同的特征空间它的表现可以作为一个基准。$Gradient\ Boosting$的表现应比它优异。

$$log(Sales) = \rho_1 Feature_1 + \rho_2 Feature_2 + \cdots + \rho_n Feature_n + \epsilon$$

具体步骤如下：

1. 对数据进行预处理：其处理方法和下面介绍$Gradient\ Boosting$的数据处理方法大致相同，只有2处略有区别：
   - 是否需要考虑多重共线性：

     基准模型会一次性将所有给定的特征进行拟合，因此对于哑变量需要去除多重共线性问题（即一个分类变量经过独热编码后产生N个哑变量，最后将$N-1$个哑变量引入线性模型中

*Gradient Boosting*本身采用弱学习器集成，同时一般会进行一定的剪枝，特征随机选择，因此无需考虑多重共线问题，直接将N个哑变量放入特征空间即可）。

- 是否需要对缺失值进行处理

  基准模型无法处理缺失值，因此对于特征中的缺失值采用该特征的均值进行填充。

  *Gradient Boosting*的回归树可以处理缺失值，因此无需对其进行缺失值处理

2. 对`trainset`里的选定特征和标签（Sales）进行拟合得到基准模型
3. 使用基准模型根据`testset`的选定特征进行标签（Sales）预测（对于`Open=0`数据直接预测为0，对于预测<0的值修正为0）

```
from sklearn.linear_model import LinearRegression
features_linear = features.copy()

# 消除多重共线问题
for x in ['StoreType_d','Assortment_c','DayOfWeek_7','Month_12']:
    features_linear.remove(x)

# 缺失值用均值替代
X_train_linear = X_train[features_linear].apply(lambda x: x.fillna(x.mean()),axis=0)
X_test_linear= testset[features_linear].apply(lambda x: x.fillna(x.mean()),axis=0)

# 线性回归，预测
linear_reg = LinearRegression()
linear_reg.fit(X_train_linear,Y_train)
testset["predict_linear"]=linear_reg.predict(X_test_linear)
```

4. 将预测结果上传至Kaggle，并得到如下得分：

Submission and Description	Private Score	Public Score
sample_submission_linear.csv just now by Peter Benchmark : Linear Regression	0.23865	0.23525

其中Private Score是同`testset`中的**61%**数据得出的，Public Score 是通过`testset`剩下的数据集得到的，则综合得分：$0.61 \times 0.23865 + 0.39 \times 0.23525 \approx 0.2373$

III. 方法

3.1 数据预处理

3.1.1构建store_popluarity , Consumption Power特征

首先构造store_popluarity 特征的数据依据只考虑`train.csv`中开店营业的数据（即`Open=1`），然后根据2.2.3，2.2.4 的策略进行构建，并将得到的store_popluarity 特征放入store数据集内：

```
# 只考虑开店营业的数据
>>>trainset_openstore = train_raw.query('Open!=0')
# 得到每家店每日客户数的中间值，并将其定义为store popularity
>>>store_popularity = pd.DataFrame(trainset_openstore.groupby(['Store']) \
                                  ['Customers'].median())
>>>store_popularity.columns = ['store_popularity']
#定义购买力 = median(Sales) / median(customers) per store
>>>consumption = pd.DataFrame(trainset_openstore.groupby(['Store'])['Sales'].median() /
                             trainset_openstore.groupby(['Store'])['Customers'].median())
>>>consumption.columns = ['consumption_power']

# 将store popularity,consumption power放入store信息内
>>>store= pd.merge(store,store_popularity,left_on='Store',right_index=True)
>>>store= pd.merge(store,consumption,left_on='Store',right_index=True)
```

3.1.2 去除开店营业数据中的Sales异常值

这里将2.2.5中发现的异常值从trainset中去除：

```
>>>trainset = trainset.query('Sales!=0')
```

3.1.3 `store` 数据与 `train`， `test` 数据合并

在进步数据处理前，我们需要将 `store.csv` 中的门店数据合并到训练集，测试集中：

```
>>>trainset = pd.merge(train_raw,store,on='Store')
>>>testset = pd.merge(test_raw,store,on='Store')
```

3.1.4 缺失值占比较大的特征处理

对于2.2.10中描述的缺失值占比较大，信息有重叠的特征予以去除，不纳入模型考虑的特征集：

```
# 去除缺失值占比大的特征
>>>dataset = dataset.drop(['CompetitionOpenSinceMonth','CompetitionOpenSinceYear',
                          'Promo2SinceWeek','PromoInterval','Promo2SinceYear'],axis=1)
```

3.1.5 从Date中创建Month ， Year特征

将数据导入后发现Date初始为'Object'对象，先将其转换会'datetime'对象后，再从中创建Month ， Year特征：

```
# 将Date转换为datetime对象
>>>dataset['Date']=pd.to_datetime(dataset['Date'])

# 创建'Month','Year'特征
>>>dataset['Month'] = dataset['Date'].map(lambda x: x.month)
>>>dataset['Year'] = dataset['Date'].map(lambda x: x.year)
```

3.1.6 StateHoliday 错误更正 + 特征重构

在2.2.8中发现一个typo，因此对于StateHoliday 先将这个错误更正，之后在根据2.2.8中的策略对StateHoliday特征进行重构：

```
# StateHoliday中的typo进行修正，并将当日不是节日设置为0
>>>dataset.loc[dataset['StateHoliday']=='0','StateHoliday']=0

# 将当日是节日，则设置为1，不管是什么节日
>>>dataset.loc[dataset['StateHoliday']!=0,'StateHoliday']=1
>>>dataset['StateHoliday']=dataset['StateHoliday'].astype('uint8')
```

3.1.7 多分类变量 - 独热编码

在选定的特征集中，StoreType，Assortment，DayOfWeek ， Month为多分类变量，不适合用数值去表示特征中各类别的区别大小，故使用独热编码，让每个特征的各个类别独立的去体现对Sales的影响程度（Year宜将其看成时序变量，因为Year无重复性：如今年是2017年，之后无论过多少年，2017都不会在之后的年份中出现。而无论哪年都有12个月，Month有重复性，所以将Month设置成多分类变量有其意义）。

同时发现`testset`中Month，只包含8，9月份因此在独热编码时，对于缺失的月份用0填充：

```
# 进行独热编码
>>>for x in ['StoreType','Assortment','DayOfWeek','Month']:
       dataset = dataset.join(pd.get_dummies(dataset[x],prefix=x))

# 对testset中缺失的月份用0填充
>>>for month in range(1,13):
       head = 'Month_' + str(month)
       if head not in dataset.columns:
           dataset[head]=0
```

3.1.8 Open处理

对于Open进行两方面的处理：

1. 对于`trainset`进行筛选，只对开店营业的数据进行学习，并只对`testset`中`Open=1`的数据进行预测

```
# 只对开店营业的数据进行学习
>>>trainset=trainset.loc[trainset['Open']==1,:]
>>>trainset = trainset.drop(['Open'],axis=1)
```

2. 对于`testset`，若`Open=0`则直接预测`Sales=1`

```
# 对关店的数据直接预测Sales=0
>>>Open_0_indice=testset['Open']==0
>>>testset.loc[Open_0_indice,'predict']=0
```

3.1.9 标签（Sales）的对数转换

将`trainset`中的Sales单独设置为标签，并对其进行对数转换以便更好的模型训练：

```
# 将Sales设置为标签
>>>Y_train = trainset.pop("Sales")

# 对标签进行对数转换
>>>Y_train = np.log(Y_train+1)
```

3.1.10 缺失值处理

由于这里使用多个回归树集成的$GradientBoosting$方法，其回归树有很好的处理缺省值的方法，故无需对缺失值进行处理（$XGBoost$中每个节点的缺失值处理方法：将所有的缺失值分别放入左子节点、右子节点，并计算各自的Gain值，然后该节点处理缺失值的默认方向是指向Gain值大的子节点）

3.2 执行过程

3.2.1 算法选择

本问题使用 $XGBoost$ 包中的XGBRegressor，并将回归方法设置为线性回归：

```
>>>from xgboost.sklearn import XGBRegressor
>>>reg = XGBRegressor(objective='reg:linear',seed=seed)
```

3.2.2 自定义损失函数

由于我们的评价指标与Kaggle相同，为了得到最好的得分，我们在模型评价指标需要自定义， 这部分也是在模型搭建中较为困难的部分

- 一开始对于损失函数直接采用 $\frac{y_i-\hat{y}_i}{y_i}$ 方法进行设置，运行后直接报error，在查看日志后发现数值达到了infinity。后来发现部分y_i （Sales）有0的情况，则会引起计算错误。
- 之后改变思路，通过设置 $w_i=\frac{1}{y_i}$, $y_i\neq 0$，$w_i=0, y_i=0$，损失函数：$w_i\cdot(y_i-\hat{y}_i)$ 则使得损失函数顺利运行：

```
# 设置cross-validation用的loss function
>>>def rmspe_exp(y_true, y_predict):
       y_true = np.exp(y_true) - 1 # 将log（y+1）变换回原来y的度量
       y_predict = np.exp(y_predict) - 1
       w = np.zeros(y_true.shape, dtype=float)
       indice = y_true != 0 # 若y=0，则权重为0
       w[indice] = 1./(y_true[indice]**2) # 根据1/y**2 设置权重
       score = np.sqrt(np.mean( w * (y_true - y_predict)**2 )) # RMSPE
       return score

>>>rmspe_exp_error = make_scorer(rmspe_exp,greater_is_better=False) # 得分越小越好
```

3.2.3 交叉验证策略

因为`trainset`数据集是按门店数据一次排列，因此在数据分块时需要先随机打乱，这里采用4次抽样的ShuffleSplit交叉验证方法，，以避免出现类似train data包含了1-800家门店，为validation data包含的801-1115家门店的情况

```

>>>cv =ShuffleSplit(n_splits=4,test_size=0.2,random_state=seed)
```

3.3 完善

3.3.1 初始模型

在模型的完善阶段我们将trainset以3 : 1的比例分成development set，evaluation set：

```
>>>X_dev, X_eval, Y_dev, Y_eval = train_test_split(
    X_train,Y_train,test_size=0.33,shuffle=True,stratify=X_train['Store'],random_state=seed)
```

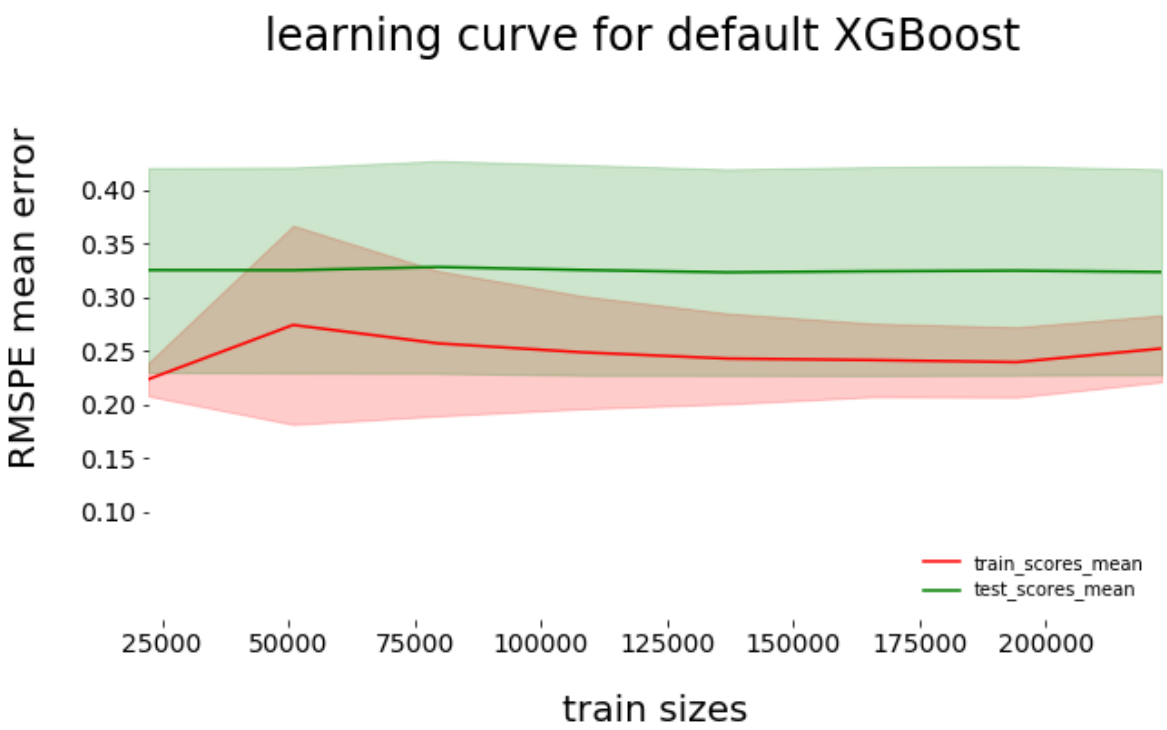
- development set : 用于parameter search使用的数据集
- evalution set : 用于的所得到的参数集进行评估的数据集

这么区分的好处是可以使得我们在参数搜索阶段的数据集不会泄露(linkage)到评估阶段，使得对模型的评估具有很好的代表性。

一开始使用*XGBoost* 初始设置的参数集，并在evaluation set绘制 learning curve 查看模型拟合情况：

> **default params:**
>
> objective : reg:linear
>
> max_depth : 3
>
> learning_rate : 0.1
>
> n_estimators : 100
>
> colsample_bytree : 1
>
> subsample : 1

```
>>>from sklearn.model_selection import learning_curve
>>>default_reg=XGBRegressor(objective='reg:linear',seed=seed)
# 初始XGBoost的learning curve
>>>train_default_sizes, train_default_scores, test_default_scores =learning_curve(
    default_reg,X_eval[features],Y_eval,train_sizes=np.linspace(0.1,1,8),
    cv=cv,scoring=rmspe_exp_error,verbose=100,shuffle=True,random_state=seed)
```



从学习曲线中发现，随着训练数据的增加，test scores 和 train scores 表现的都不好，这是非常典型的欠拟合的情况，因此这里在随后的模型修改中，可选参数集合中应包含比默认参数集更为激进的参数范围

3.3.2 RandomizedSearchCV 搜索优化参数组合

在3.31中得出应采用较为激进的参数范围进行优化，由于使用 ***XGBoost*** 有非常多的可以设置的参数，每个参数的可选范围又可以是无穷，因此在可选参数范围上引入领域知识从而缩小可选参数空间：

可选参数空间：

```
# 根据domain knowledge 设置参数范围
>>>params = {
    'n_estimators':np.arange(100,650,50),
    'max_depth':[4,5,6,7,8],
    'colsample_bytree':[0.5,0.6,0.7,0.8],
    'subsample':[0.5,0.6,0.7,0.8,],
    'learning_rate':[0.05,0.1,0.15,0.2,0.25,0.3]
   }
```

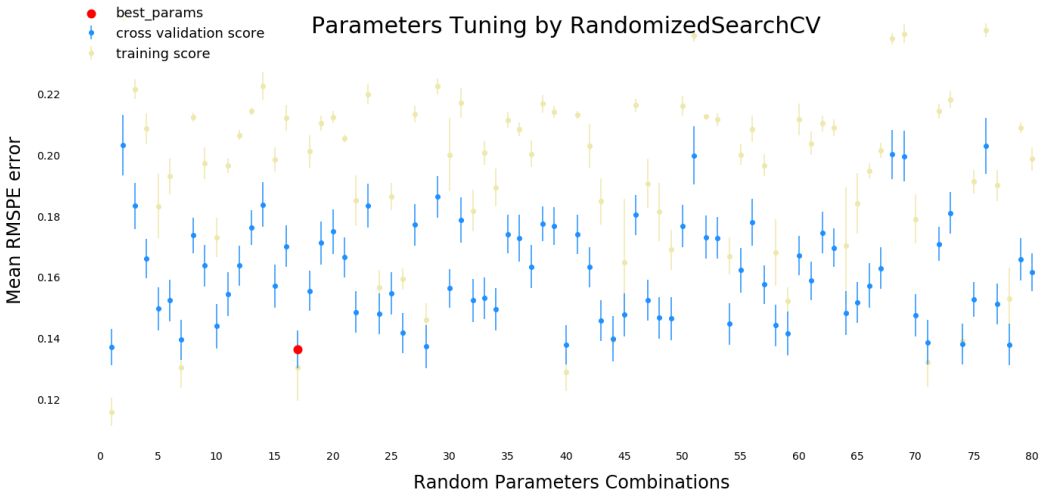
这里我们希望通过优化参数组合使得模型的表现比初始模型要优秀。

可选的参数组合数有：$11 \times 5 \times 4 \times 4 \times 6 = 5280$ 种组合。结合4次随机的交叉验证策略，若使用*GridSearchCV*则需要训练21120 模型次，这个在时间上的花销是巨大的。

因此这里采用*RandomizedSearchCV*的搜索方法，并将尝试次数设定为80次。这样所需训练的模型数：320 个。

```
>>>bst_grid = RandomizedSearchCV(estimator=reg,param_distributions=params,
                                  cv=cv,scoring=rmspe_exp_error,
                                 verbose=100,n_iter=80,random_state=seed)
>>>bst_grid.fit(X_train[features],Y_train)
```

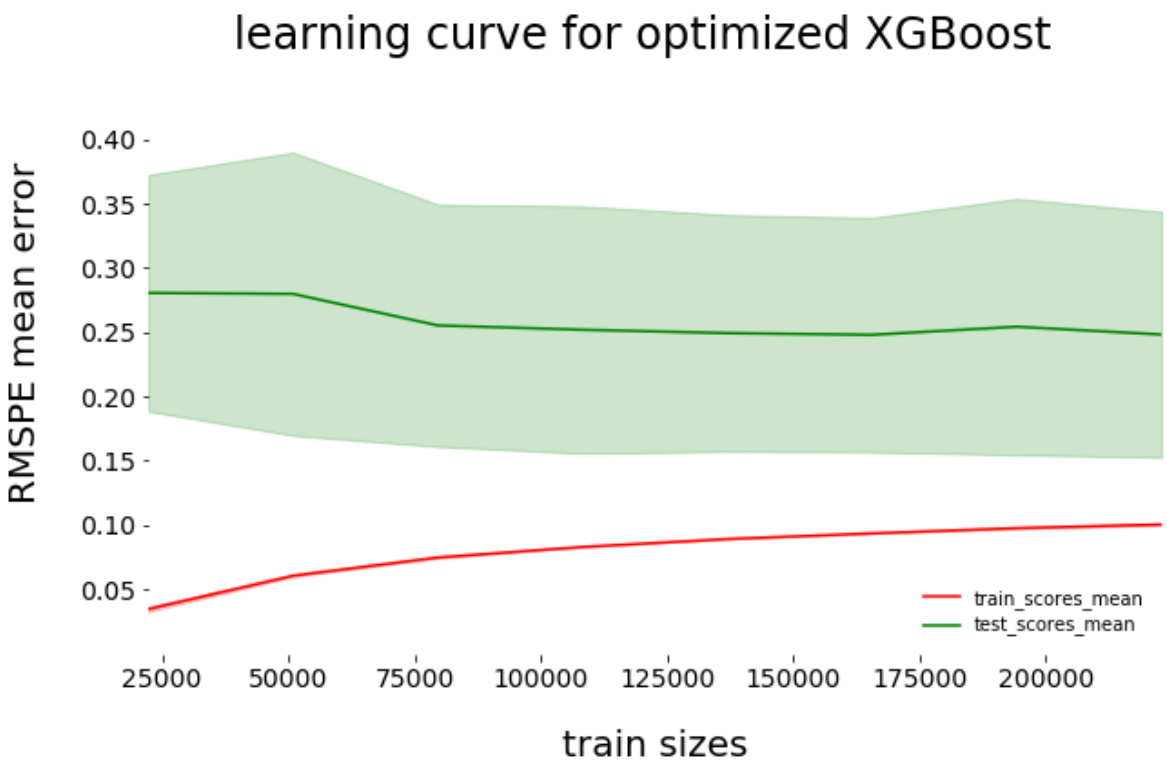
根据*RandomizedSearchCV*交叉验证的结果得到下图：（其中圆点为均值，线段为左右一个标准差的范围）



最佳参数组合为：

n_estimators	learning_rate	max_depth	colsample_bytree	subsample
550	0.25	8	0.8	0.8

将该参数的模型带入evaluation set并计算learning curve：



发现优化后的error明显比初始模型的要减小接近30%，同时随着训练数量的增加test error 逐渐下降，train error 逐渐增加，即增加训练集可以是两者越来越接近并与之交汇，可以期待当将整个`trainset`都用于训练，并用`testset`进行预测，上传至Kaggle的分数会比上图的error (≈ 0.25) 更低。

IV. 结果

4.1 模型的评价与验证

将最佳参数放入模型中，用完整的`trainset`进行训练，并用`testset`进行预测：

```
>>>import xgboost as xgb

>>>params_rcv = {
      'objective':'reg:linear',
      'colsample_bytree': 0.8,
      'subsample': 0.8,
      'eta': 0.25,
      'max_depth': 8
   }

>>>n_rounds = 550  # 同n_estimators
>>>dtrain = xgb.DMatrix(X_train[features],Y_train)
>>>dtest = xgb.DMatrix(testset[features])
>>>origin_reg = xgb.train(params_rcv,dtrain,n_rounds,feval=rmspe_xg) # 训练模型

>>>testset["predict"] = orgin_reg.predict(dtest) # 预测testset
```

将预测数据上传至Kaggle，模型得分为：

Submission and Description	Private Score	Public Score
sample_submission_optimized.csv a few seconds ago by Peter XGBoost optimized by RandomizedSearchCV	0.13058	0.11824

综合得分：$0.61 \times 0.13058 + 0.39 \times 0.11824 \approx 0.1257$。与之前的预期相符

该模型的设计合理，将弱学习器选为线性回归有助于集成学习的搭建，参数选择空间的设置是基于初始模型的拟合结果的情况下进步提出的，对问题有良好的改进。将优化后的模型最后的Kaggle得分较为满意，模型的结果可信度高。

从3.3.2中优化后模型的learning curve可以看出，evaluation set 中的error score随着数据量的变量呈现很稳定的趋势，模型鲁棒性很强（初始模型的learning curve则有较大的波动，缺乏鲁棒性）。

4.2 合理性分析

从4.1给出的结果，最终优化后的模型在Kaggle得分上的提升较基准模型提高了**47%**，表现改进了非常多

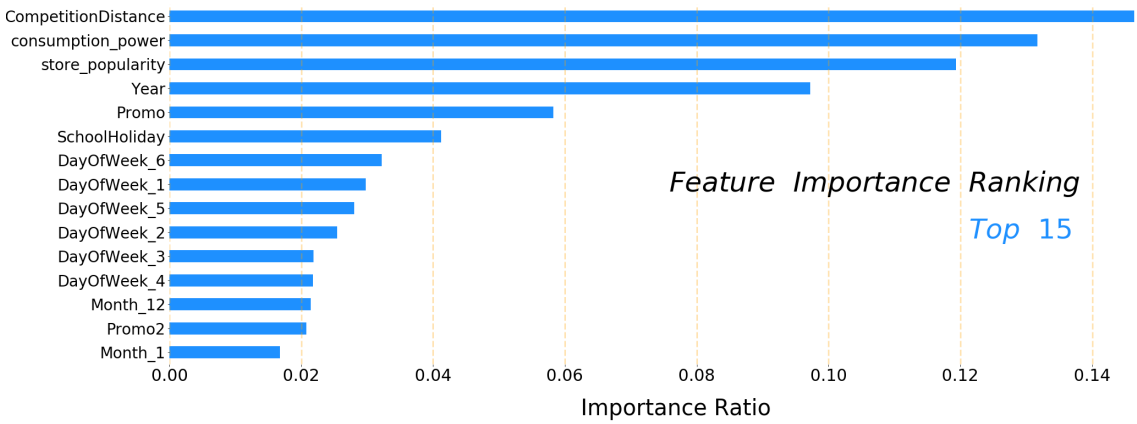
最终结果**0.1257**表明：优化后的模型每次预测会与实际值有大约**87.4%**的预测准确度。

毕竟预测未来本身就不可避免存在误差，若一个模型的预测准确率能有接近**90%**的预测准确率的化，其作为企业经营决策的会是一个非常大的帮助，因为其决策基于的预测数据与实际情况的误差并不大，不会对决策结果产生质的改变。

V. 项目结论

5.1 结果可视化

XGBoost 在训练模型同时可以同时得到各特征对于Sales预测的贡献（Gain），这也是回归树/决策树最受欢迎的技术之一，这里我们将每个特征看成个体,把特征的贡献归一化，并列出重要性前15的特征：



- **Competition Distance**对Sales的影响最大，即如新店选址时要充分考虑竞争对手的位置
- **Consumption Power**影响其次，表明地点是一方面，若周边居民是否有消费力也非常重要，如果客户每次来只买个泡腾片那也没啥钱途。又如开在医院旁边的药店销售额一般都比较好。
- **Store Popularity**对Sales的影响第三，这个特征隐含了地段优势，附近居民购买行为等信息。说明周边居民去附近药店购买欲望的大小。
- **Year**对Sales影响第四，表明预测未来销售量。则最近年份已有的数据会比较远年份的数据更有影响
- **Promo**对Sales影响第五，说明促销活动具有很明显的影响，这对于企业的经营策略部署提供有效建议和支持

5.2 对项目的思考

5.2.1 项目流程

本项目步骤共经历以下一个流程：数据可视化探索 --> 目标模型和基准模型选择 --> 数据预处理 --> 目标模型搭建 --> 参数调优 --> 评估预测 --> 结果展示

5.2.2 项目中有趣的部分

项目中最有趣的部分就是一开始的数据可视化探索，在将数据可视化后发现了很多一开始和自己感觉向违背后在仔细思考发现的确是这么回事的现象（如节假日的药店销售量都会少很多，不同门店的受欢迎程度也会相差很多等）

在整个项目中大约有50%的时间都在数据可视化探索上，20%在数据预处理，剩下的30%时间放在模型搭建和参数调优上。

5.2.3 项目中遇到的困难

1. 项目中遇到的最大的困难就是模型搭建中的自定义损失函数部分，其具体经过已经在3.2.2中具体描述。
2. 其次的困难就是在撰写报告制作这些可视化图形，这里使用 $Matplotlib$ 包进行绘图，其初始的图形非常粗糙，而 $Matplotlib$ 的特点是每次只修改一点，因此一个满意的可视化通常>10行代码，并且很多小改动的API都不了解，需要一一的谷歌查询解决方案，不可谓不困难。
3. 第三个困难就是$XGBoost$的学习，在做这个项目之前并未接触过所以学的过程，这里主要是通过Practical XGBoost in Python这个免费课程学习其原理和操作方法。

5.2.4 最终模型是否可以使用通用场景

最后模型的得分非常满意，同时对于类似的单一连续因变量的回归问题也可以用 $XGBoost$ 解决问题，可以在回归树的回归方法可以考虑选用其它的回归模型（reg : gamma，reg : tweedie）。对于集成学习最重要的防止弱学习器过拟合，因为集成学习是要的目标是最小化偏差（Bias），如果弱学习器导致了很大的方差（Variance），则集成学习对过拟合是没有修正和改善作用的，因此在选择回归模型时建议使用线性回归（reg : linear）这种不太容易过拟合的模型为宜。

5.3 需要作出的改进

在本项目中可以进步改进的地方，就是参数的设置还没有达到局部最优

这里可以采用SMO (Sequential Minimal Optimization) 算法进行优化，具体步骤：

```
for n in 迭代N次
    for param in parameter_sets:
      1.  固定其他参数（parameter_sets \ param）
      2.  对param参数范围内每一个选值训练模型，计算损失函数RMSPE大小
      3.  param = RMSPE最小的所对应的选值

得到局部最优的参数集
```

通过上述方法可以使得参数的设置更接近局部最佳状态。