Rossmann 销售预测

Machine Learning Engineer Nanodegree

Capstone Report

闵安

2018-07

## 定义

* 1. 项目概述

Rossmann是欧洲的一家连锁药店，有超过3000家店面，分布在欧洲7个不同的国家。Rossmann的管理者希望能够预测未来最长6周的每家店面每天的销售额。影响一个店面的销售额的因素有很多，包括竞争对手，是否有促销，是否是节假日，季节和位置等。

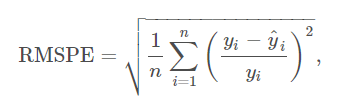
本项目来源于Rossmann公司在Kaggle上发布的竞赛“Rossmann Store Sale”**[1]**。我们需要根据Rossmann药妆店的信息（比如促销，竞争对手，节假日）以及在过去的销售情况，来预测Rossmann未来的销售额。

* 1. 问题说明

这个项目的要求是预测单个店面的销售额。而销售额是个连续的数值，所以这个项目是一个回归类的问题，而不是一个分类问题。所以本项目需要创建一个回归模型用于预测每个店面的销售额。预测值和实际值之间的误差要足够小。

* 1. 指标

根据Kaggle 的要求[4]，采用的指标Root Mean Square Percentage Error(RMSPE)，公式为：



其中y\_i表示的是单店当天的销售额，yhat\_i 表示的是对应的预测值。如果某店某天的销售额是0，则忽略这条记录。

根据项目要求，回归模型的预测结果的误差要能够达到Kaggle上该项目Private Leaderboard的前10%的结果。该项目，Kaggle上有3303个参赛者，前10%的误差是小于等于0.11773。即最终项目得到的RMSPE值要小于这个数值。

1. **分析**
   1. 数据研究

数据集包含在三个文件中：train.csv, test.csv, store.csv。下载路径为：https://www.kaggle.com/c/rossmannstore-sales/data

* train.csv 每家商店每天的销售相关的信息，例如是否营业，促销，节假日以及当天的顾客量和销售量等，用于训练。
* test.csv 与train.csv 的结构相似，但是不包括当天顾客量和销售量，用于测试。
* store.csv 每家商店的额外的信息，例如商店类型，等级，竞争者信息，以及长期的促销活动等。
* train.csv 有1017209 条记录，包括1115 家商店。记录了从2013 年1 月1 日至2015 年7 月31 日约2 年半的历史数据，但是有180 家商店缺少从2014 年7 月1 日至2014 年12 月31 日约半年的数据；
* test.csv 有41088 条记录，包括856 家商店从2015 年8 月1 日至2015 年9 月17 日约6 周的数据。
* store.csv有1115 条记录，每一条记录表示一家店面的的属性，如商店类型，竞争对手的信息，促销信息等。

表1：Train.csv字段说明

|  |  |  |  |
| --- | --- | --- | --- |
| 字段名 | 字段含义 | 字段取值范围 | 字段类型 |
| Store | 商店ID | 1-1115 | 离散 |
| DayofWeek | 星期几 | 1-7 | 离散 |
| Date | 日期，格式为YYYY-MM-DD | 2013 年1 月1 日至2015 年7 月31 日 | 离散 |
| Sales | 销售额 | 正整数 | 连续 |
| Customers | 用户数 | 正整数 | 连续 |
| Open | 是否开张 | 0 = closed, 1 = open | 离散 |
| Promo | 当天是否有促销 | 0=否，1=是 | 离散 |
| StateHoliday | 国家假日，通常，商店在国家假日不营业 | a = public holiday,  b = Easter holiday,  c = Christmas,  0 = None | 离散 |
| SchoolHoliday | 学校放假 | 0 = None  1 = 放假 | 离散 |

表2：Test.csv字段说明

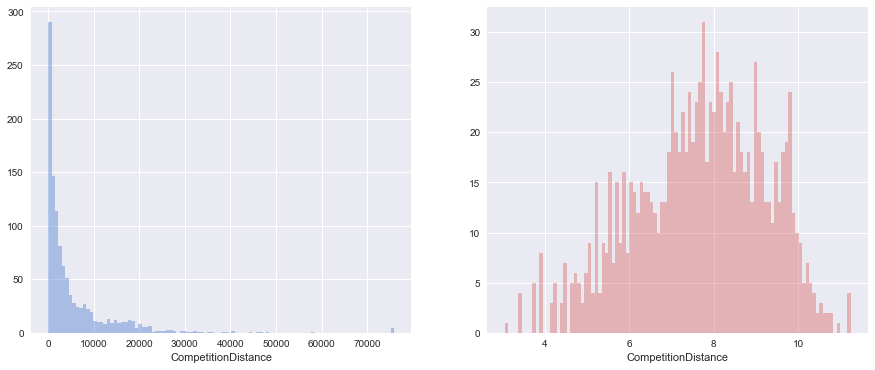
|  |  |  |  |
| --- | --- | --- | --- |
| 字段名 | 字段含义 | 字段取值范围 | 字段类型 |
| Id | 记录的序列号 | 1-41088 | 连续 |
| Store | 商店ID | 1-1115 | 离散 |
| DayofWeek | 星期几 | 1-7 | 离散 |
| Date | 日期，格式为YYYY-MM-DD | 2013 年1 月1 日至2015 年7 月31 日 | 离散 |
| Open | 是否开张 | 0 = closed, 1 = open | 离散 |
| Promo | 当天是否有促销 | 0=否，1=是 | 离散 |
| StateHoliday | 国家假日，通常，商店在国家假日不营业 | a = public holiday,  b = Easter holiday,  c = Christmas,  0 = None | 离散 |
| SchoolHoliday | 学校放假 | 0 = None  1 = 放假 | 离散 |

表3：Store.csv字段说明

|  |  |  |  |
| --- | --- | --- | --- |
| 字段名 | 字段含义 | 字段取值范围 | 字段类型 |
| Store | 商店ID | 1-1115 | 离散 |
| StoreType | 商店类型 | a, b, c, d | 离散 |
| Assortment | 商店分级 | a = basic,  b = extra,  c = extended | 离散 |
| CompetitionDistance | 最近的竞争对手的距离，单位是米 | 正整数 | 连续  有三条记录该字段为空 |
| CompetitionOpenSinceMonth | 竞争对手开张的月份 | 1-12 | 离散，有空记录 |
| CompetitionOpenSince  Year | 竞争对手开张的年份 | 1900. 1965,2015, etc | 离散，有空记录 |
| Promo | 当天是否有促销 | 0=否，1=是 | 离散 |
| Promo2 | 是否参与持续的、联系的促销 | 0=否，1=是 | 离散 |
| Promo2SinceWeek | 年内的第几周参与Promo2的促销 | 1-54，有空的记录 | 离散 |
| Promo2SinceYear | 某年参与Promo2促销 | 2009,2014，etc。有空的记录 | 离散 |
| PromoInterval | 每年Promo2 开始的月份 | "Jan,Apr,Jul,Oct",  "Feb,May,Aug,Nov" ,  "Mar,Jun,Sept,Dec"  有空的记录 | 离散 |

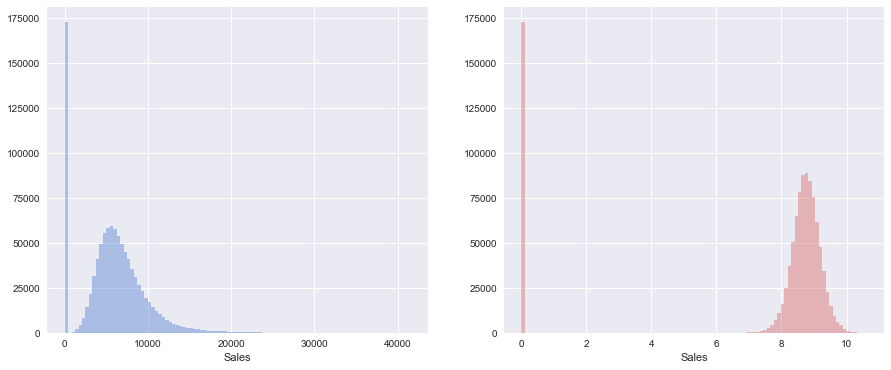
可以看出大部分字段是离散型的取值。只有这三个字段是连续的：Sales，Customers，和CompetitionDistance。现在观察一下这三个连续字段分布情况

1. Competition Distance的分布：



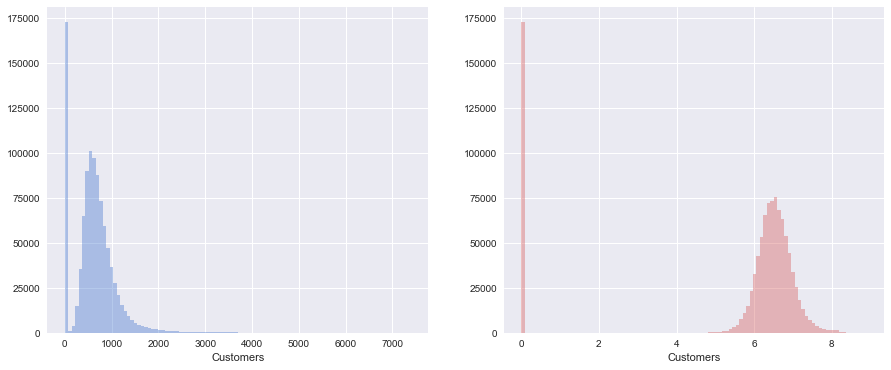
数据分布严重左倾，经log转换后，也不是正态分布

1. Sales的分布



Sales的分布，有大量数据为0，非0的数据的稍微左倾分布，log转化后，接近正态分布

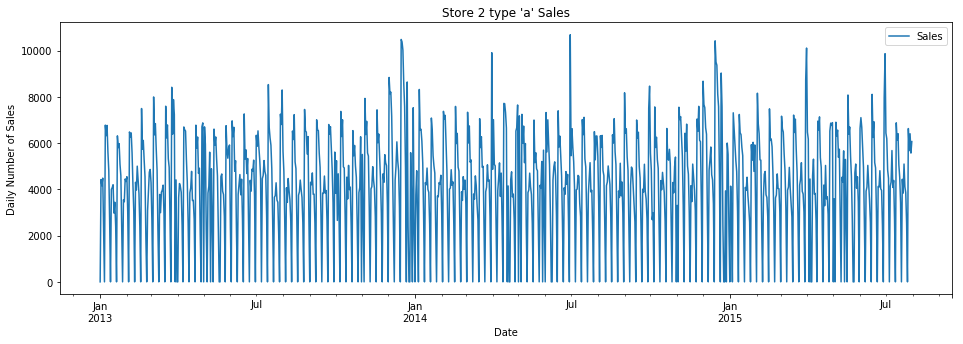
1. Customers的分布

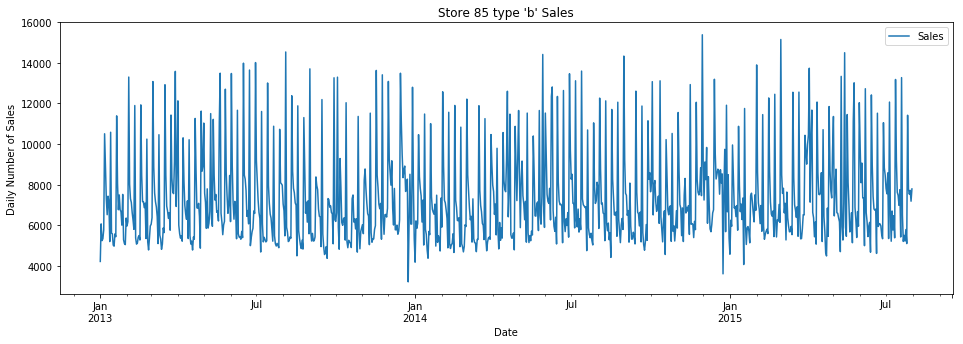


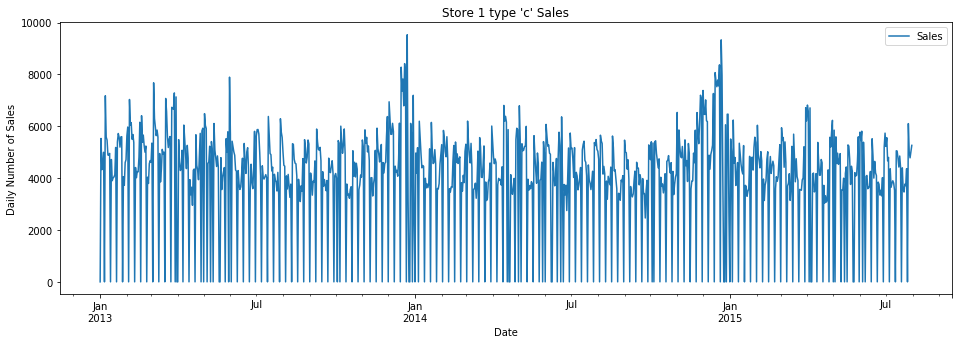
Customers的分布，有大量数据为0，非0的数据的左倾分布，log转化后，接近正态分布

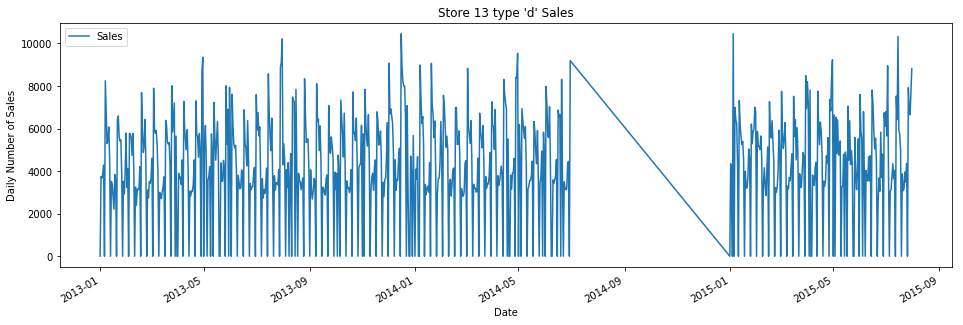
* 1. 探索数据
     1. 销量和时间的关系

这个选取几个商店的数据看一下Sales的情况。根据Store Type的值，分别取一个商店的数据。Store type 的值 a，b，c，d 分别对应Store为：（2, 85, 1, 13）







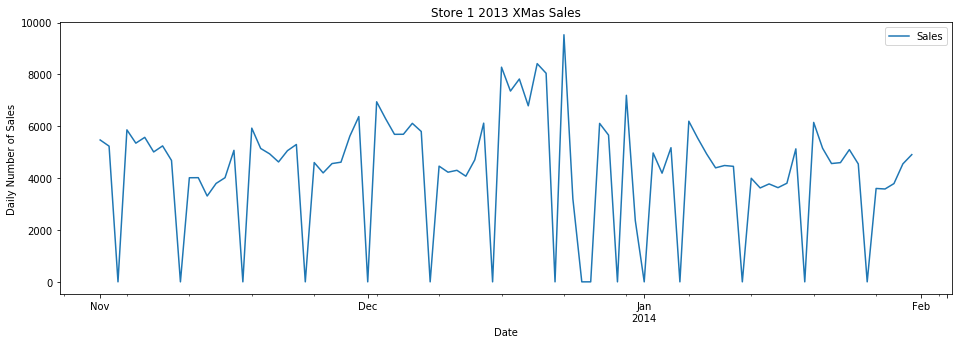


从2013-1-1到2015-7-31的销售数据来看，可以看出上述4个Store的数据的趋势为：

1. 年和年之间的销售数据是类似的
2. 年底的时候，销量有个高峰

从上面的数据可以看到，销售数据是周期性，大致一周为周期。一周中不同的天有不同的销量。

再取Store 1 2013 年底和2014年初的数据观察一下：

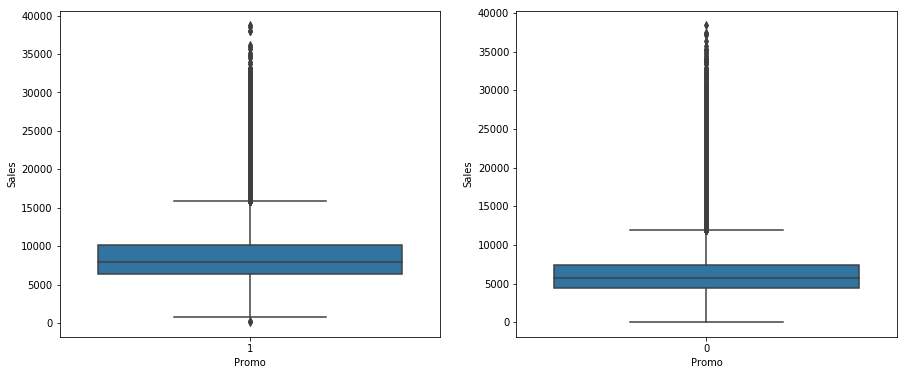


发现圣诞节前的销量明显有增高，而节后的销量则会回落。说明不同的月份以及不同的周会有不同的销量。

* + 1. Promo 和销量的关系

基于常理，可以假设Promo对销量有促进作用。取非Promo2的记录，再根据Promo字段的取值（0,1）来筛选出销售数据做对比：

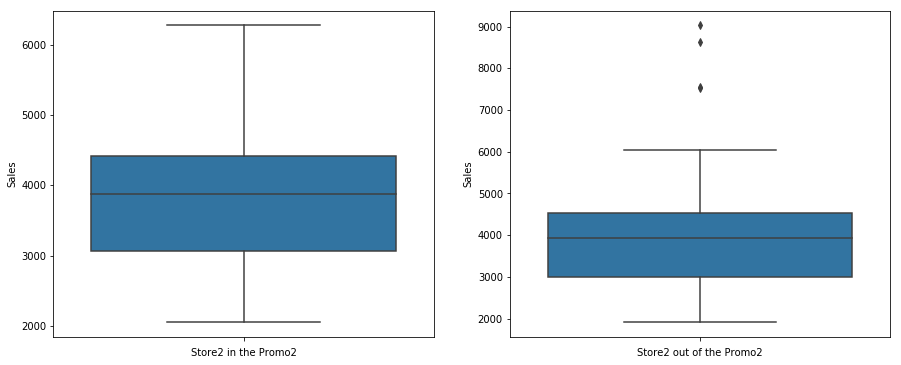
左侧的是Promo=1，右侧的是Promo=0



可以看出，左图的销售数据比右图要高出一个级别。

* + 1. Promo2 和销量的关系

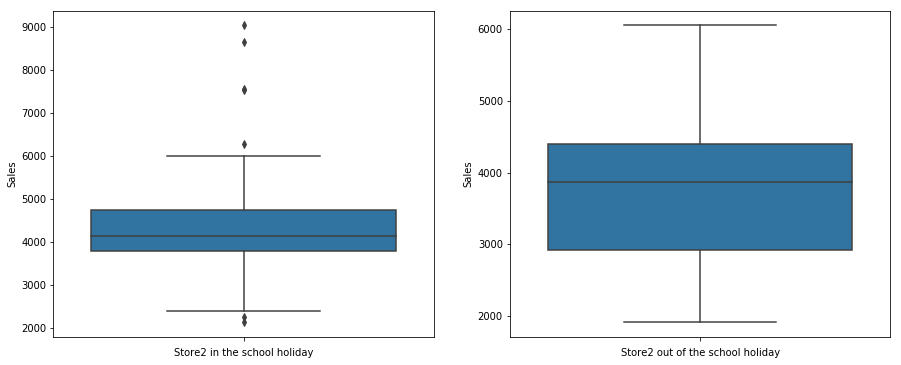
推测Promo2 对销量有促进作用。随机选取Store 2，Promo2的月份是1,4,7，10 这四个月份。然后筛选出没有Promo(Promo)的记录，把1,4,7,10这四个月的记录作为一份数据，其他的数据作为另一份记录，两者做个对比：



仔细观察发现Promo2对销量的影响似乎不大。

* + 1. School Holiday 对销量的影响

选取Store2，没有Promo的记录，对比Schoolholiday 期间和没有SchoolHoliday的销量数据，如下图：

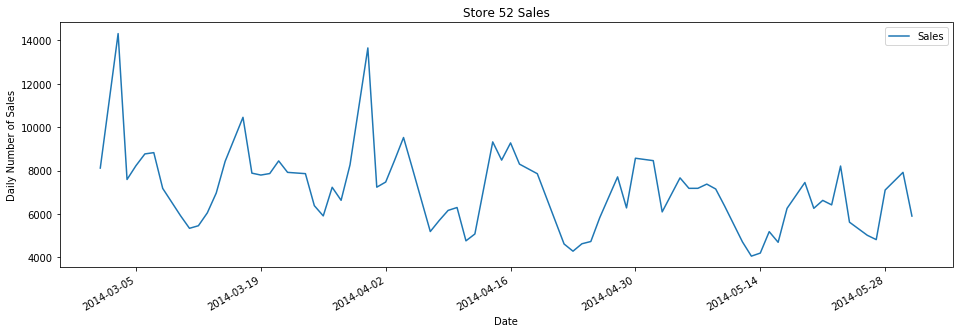


可以看出school holiday期间销量要高一些

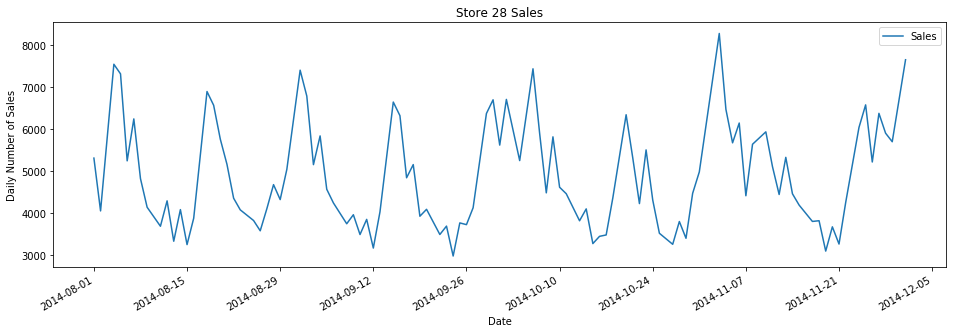
* + 1. Competition 商店开业的对销量的影响

基于常理推断，竞争商店的开业对销量有影响，如果本商店有大力促销，可能会促进销售；如果没有什么应对措施，可能会影响销售。竞争商店和本店的距离，也有影响，距离越近，则影响越大。这里选取两个商店来看一下对销售的影响：

* Store 52，竞争商店2014年4月开业，距离450m
* Store 28，竞争商店2014年10月开业，距离1200m



可以看出，store 52 4月第一周有个销售高峰（Promo等于1）。3月初的销售高峰是因为有Promo活动（Promo等于1）。



store 28 在9-29 ~ 10-2 有Promo 10-6~10-10 有Promo，这段时间销量其他周相对高一些。

* + 1. 总结

通过上述数据分析的结果，可以初步认为销量会受到如下因素的影响：

1. Month
2. Day of week
3. Week of year
4. Day of Month
5. Day of Year
6. Promo 促销
7. Competition Distance
8. Promo2 促销（目前看影响不明显）
9. State Holiday
10. School Holiday
    1. 算法和方法

本项目中，我采用Python做数据探索，用到了Pandas做数据处理，使用Seaborn（0.8.1）， Matplotlib做数据可视化。

根据Kaggle竞赛第一名的建议[2]，使用XGBoost (eXtreme Gradient Boosting )[3]作为回归算法。把Train.csv 和 store.csv 用Store ID 做关联，合并成一个数据文件，然后切割出最后的六周的数据作为验证集。前面的数据作为训练集。

* 1. 基准测试

根据项目要求，回归模型的预测结果的误差要能够达到Kaggle上该项目的前10%的结果（基于Private Leaderboard）。该项目，Kaggle上有3303个参赛者，前10%的误差是小于等于0.11773

1. **方法**

根据Kaggle Rossmann 项目的论坛上经验，基于xgboost需要构造一些额外的字段，即做特征工程。

* 1. 数据预处理
     1. 填充NaN值

Store.csv 中有缺失的数据，通过Pandas读取该文件后，存储在变量store中。Store中如下字段有NaN值：

StoreType 0

Assortment 0

CompetitionDistance 3

CompetitionOpenSinceMonth 354

CompetitionOpenSinceYear 354

Promo2 0

Promo2SinceWeek 544

Promo2SinceYear 544

PromoInterval 544

* CompetitionDistance字段，有3个NaN值。用最大值填充这三个字段。代码如下：

store = store.fillna(store.max()['CompetitionDistance':'CompetitionDistance'])

* CompetitionOpenSinceMonth和CompetitionOpenSinceYear 使用最早的时间来填充，即最小值。

store.fillna(store.min()['CompetitionOpenSinceMonth':'CompetitionOpenSinceYear'], inplace=True)

* Promo2SinceWeek， Promo2SinceYear， PromoInterval 这三个字段用0来填充缺失的值，并且缺失的值都是没有Promo2的Store。

store.fillna(0, inplace=True)

Train.csv中没有字段是NaN值，但其中有180 家商店缺失半年的记录。不过这些记录可以不用补齐。缺失的记录就是相当于少了一些训练的数据。

* + 1. 离散字段处理

对离散字段使用Label编码。

Store数据中，需要处理“StoreType”，“Assortment”这两个字段：

store['StoreType'] = store['StoreType'].map({'a':0, 'b':1, 'c':2, 'd':3})

store['Assortment'] = store['Assortment'].map({'a':0, 'b':1, 'c':2, 'd':3})

PromoInterval字段中的值是个字符串，表示哪几个月有促销。经检测，每个字符串中，都是有4个促销月份，比如“Jan,Apr,Jul,Oct”，把这个字段拆分成4个字段：

* Interval1
* Interval2
* Interval3
* Interval4

，每个字段保存一个月份，用整数来表示月份，从1~12 分别表示Jan 到 Dec。

Train数据中， 对StateHoliday字段做Label编码：

train['StateHoliday'] = train['StateHoliday'].astype(str)

train['StateHoliday'] = train['StateHoliday'].map({'0':0, 'a':1, 'b':2, 'c':3})

* 1. 特征工程

本节描述新增了哪些字段，用于训练xgBoost 模型。

* + 1. 对CompetitionDistance的处理

这个字段表示的是竞争商店的距离，常理推知，竞争商店离的越近，则对本店影响越大。尝试对这个字段做离散化处理，比如用1000米为单位设置成从1开始的整数；也尝试过，非等距的方式，比如500米内，设置成1,然后500~2000米设置成2, 2000~5000米设置成3，再远的距离设置成4。最终通过xgboost模型的运行，发现还是直接使用连续的CompetitionDistance值的效果最好，而且使用原始值比用 log() 处理后的值，效果还要稍好一些，所以最终这个字段采用的是原始值。

* + 1. 对Date 字段的处理

根据在第二节的分析，基于Date 字段，可以获取如下新增字段：

* Year
* Month
* Day： 这个是Day of Month，从1 开始
* Week of Year
* Day of Year

再加上已有的DayOfWeek 字段，时间相关的字段就完整了。

另外，为了方便后续的计算，我把每条记录的日期都转换成自2000年1月1日以来的天数，并新增一个字段“CurrentDay” 来记录这个天数。

* + 1. 对Promo2 相关字段的处理

部分商店从某个时间点开始，每年有4个月的促销期。基于经验，人们在促销快开始的时候，非急需的商品会等到促销期间买，这相当于延迟消费，使得商店的销售额有所下降。 而在促销期间，人们会提前购买一些未来需要的商品，会增加商店的消费额。并且，人们会集中在促销开始的期间进行抢购，在促销期的后期，优惠商品可能都卖完了，销售额会有所下降。促销结束后，人们因为提前购买了商品，也会导致销售额有所下降。基于这些经验，把Promo2 的开始时间以及每年的4个月，和Train 数据中的Date转换成 促销前，促销中，促销后的天数，新增三个字段。比如促销前1天，2天，100天等，促销中0天，1 天，10天等，促销后1天，2天，20天等。这三个字段的赋值方式如下表：

|  |  |  |
| --- | --- | --- |
| Before Promo2 | In the Promo2 | After Promo2 |
| 2 | -1 | 100（晚于上一个Promo2月份的结束时间的天数） |
| 1 | -1 | 101 |
| -1 | 0 | -1 |
| -1 | 1 | -1 |
| ... | ... | ... |
| 100 (早于下一个Promo2月份的时间) | -1 | 1 |
| 99 | -1 | 2 |

这三个字段事实上有一定的相关性，从时间轴上来看，是有连续性的。但用一个字段表示三种状态，不方便描述。而且对于树模型而言，更多的字段可以方便树来进行分叉操作。

* + 1. Competition Open time的处理

大部分竞争商店是在2013年之前开业的，但也有小部分是在2013, 2014, 2015年开业的，这对相应的商店的销售数据有直接的影响。

我尝试了两种方式。第一种是新增一个字段“CompetitionLevel”，用当天和竞争商店开业的那天进行相减，如果当天比开业的天要早，则为负数，否则为正数。

第二种方式，是新增两个字段，分别为“BeforeCompetition” 和 “AfterCompetition”。 然后赋予离散的值。开业的前后几天，对销量的影响最明显。时间离的越远，则影响越小。赋值方式如下：

|  |  |
| --- | --- |
| Before Competition | After Competition |
| 半年之前，值为1 | 0 |
| 一个月之前，值为10 | 0 |
| 一周之前，值为100 | 0 |
| 一周之内，值为1000 | 0 |
| 0 | 一周之内，值为1000 |
| 0 | 一个月之内，值为100 |
| 0 | 半年之内，值为10 |
| 0 | 超过半年，值为1 |

经过运行xgboost模型验证，采用方式二。

* + 1. 去除Sales的Outlier 值

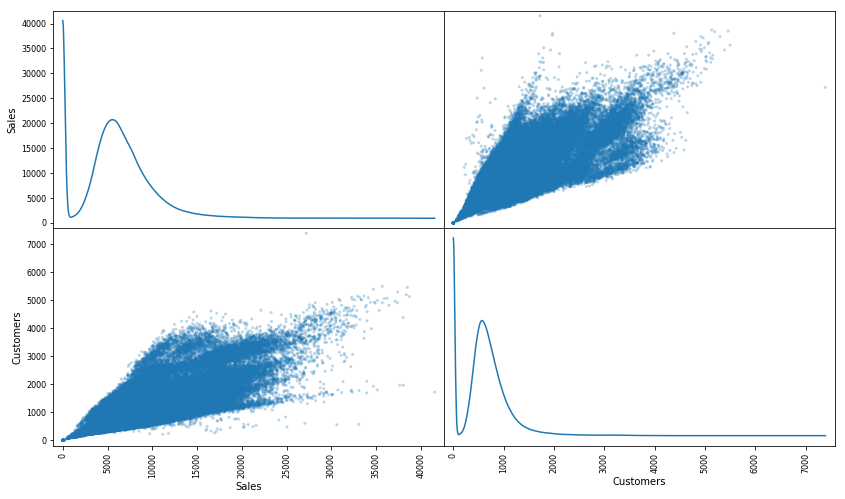
因为Sales的数据Log()处理后，符合正态分布，可以使用Tukey 方法去除Outlier数据，即1.5\*(Q3-Q1)范围之外的数据点。

* + 1. 利用Sales 和 Customers 数据

参考了[Johan Manders](https://www.kaggle.com/johanmanders) 的方案[5]，增加了如下三个字段：

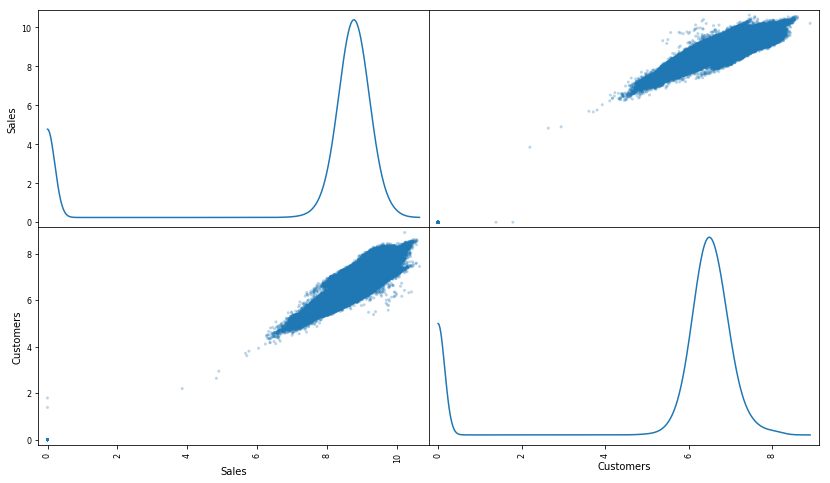
* 每个商店天平均销量：总销量除以总天数。Average Sales per Day per Store
* 每个商店天平均顾客数：总顾客数除以总天数。Average Customers per Day per Store
* 每个商店天平均单顾客销量：；上述两个值相除。Average Sales per Customer per Day per Store

观察一下Sales 和Customers 之间的相关性



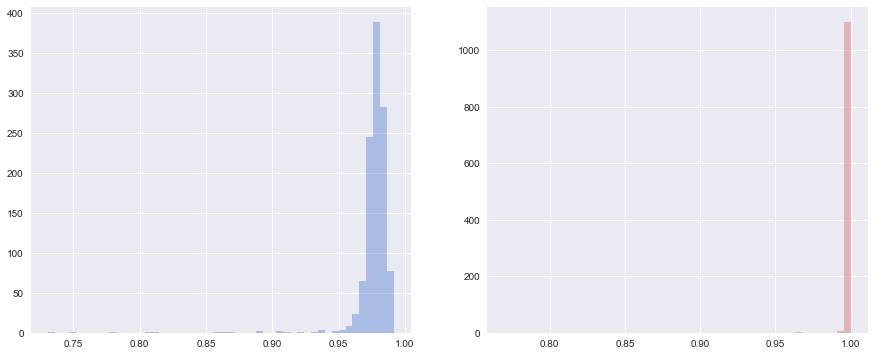
pearson 相关系数为：0.894711

对两个字段做log()处理后，再看相关性如下：



pearson 相关系数为0.996253

再按照每个Store来查看Sales和Customers 之间的相关系数分布：



log() 处理后，几乎所有Store的相关系数非常接近1，可以说明相关性非常高。

这里采用Sales，Customers的平均值和对应的每Customer销售均值，来作为新的特征。

* + 1. 对采用外部数据的考虑

在Kaggle 论坛上，很多人的方案都使用了外部获取的天气数据和商店的位置信息。我理解如果在实际中应用该模型来做预测，在网上是无法获取未来六周的天气信息的，所以我没有采用同样的方法来获取外部数据。

* 1. 实现

筛选的特征列表如下：

'Store', 'DayOfWeek', 'Promo', 'StateHoliday', 'SchoolHoliday', 'Year', 'Month', 'Day', 'WeekofYear', 'StoreType', 'Assortment', 'Promo2', 'CompetitionDistance', 'Competition\_level\_before', 'Competition\_level\_after', 'DaybeforePromo2', 'DayinPromo2',

'DayafterPromo2', 'SalesPerDay', 'CustomersPerDay', 'SalesPerCustomersPerDay'

**xgboost** 参数为：

params = {"objective": "reg:linear",

"booster" : "gbtree",

"eta": 0.01, #0.3, 0.1, 0.01

"max\_depth": 8, #10, 8

"subsample": 0.9, #0.9, 0.5

"colsample\_bytree": 0.7,

"silent": 1,

"seed": 42

}

num\_boost\_round = 36000

early stop = 300

1. **结果**

运行了23400轮左右停止了，最佳结果是：

[21300] train-rmse:0.069578 eval-rmse:0.110237 train-rmspe:0.073531 eval-rmspe:0.115447

[21600] train-rmse:0.06939 eval-rmse:0.110192 train-rmspe:0.073304 eval-rmspe:0.115394

[21900] train-rmse:0.069202 eval-rmse:0.110181 train-rmspe:0.073074 eval-rmspe:0.115378

[22200] train-rmse:0.069017 eval-rmse:0.110159 train-rmspe:0.072852 eval-rmspe:0.115361

[22500] train-rmse:0.068837 eval-rmse:0.110129 train-rmspe:0.072643 eval-rmspe:0.115335

[22800] train-rmse:0.068657 eval-rmse:0.110106 train-rmspe:0.072426 eval-rmspe:0.115305

[23100] train-rmse:0.06848 eval-rmse:0.110083 train-rmspe:0.072215 eval-rmspe:0.115269

Stopping. Best iteration:

[23095] train-rmse:0.068482 eval-rmse:0.110081 train-rmspe:0.072218 eval-rmspe:0.115268

train 过程数据如下，这里显示了头5条数据。train 和 eval 的数据很接近

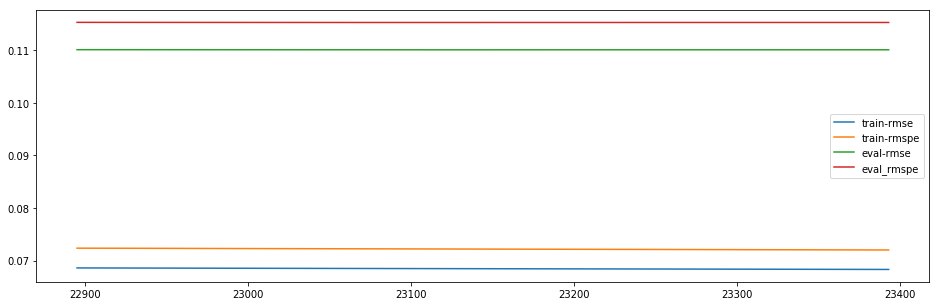
|  | train-rmse | train-rmspe | eval-rmse | eval\_rmspe |
| --- | --- | --- | --- | --- |
| 0 | 8.173599 | 0.999863 | 8.185439 | 0.999865 |
| 1 | 8.091918 | 0.999837 | 8.103895 | 0.999840 |
| 2 | 8.011169 | 0.999809 | 8.023099 | 0.999812 |
| 3 | 7.931110 | 0.999779 | 7.943172 | 0.999782 |
| 4 | 7.851860 | 0.999747 | 7.864011 | 0.999751 |

模型的训练曲线如下：



说明： 因为tain 和 eval 的值很接近，最后就只能看到一条曲线了。

取最后的5000条记录，观察一下：



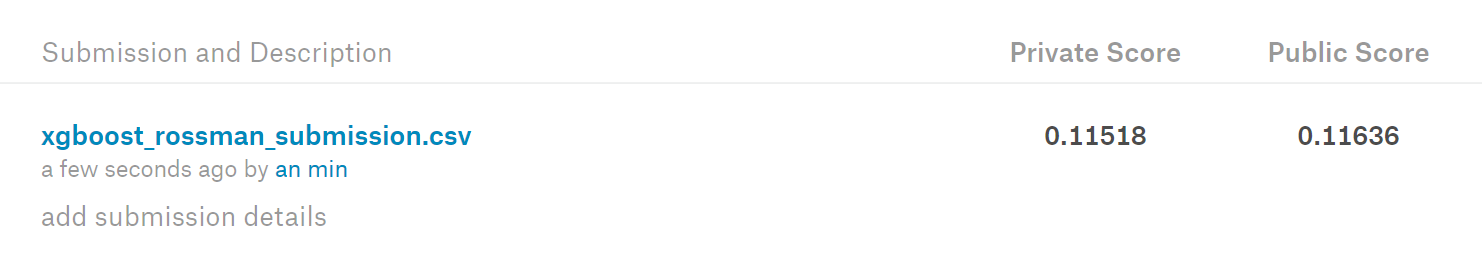
可以看出，斜率很小。因为Learning rate 是0.01所以loss 下降的慢，而且是越来越慢。

用最后六周的数据作为验证集，得到结果：

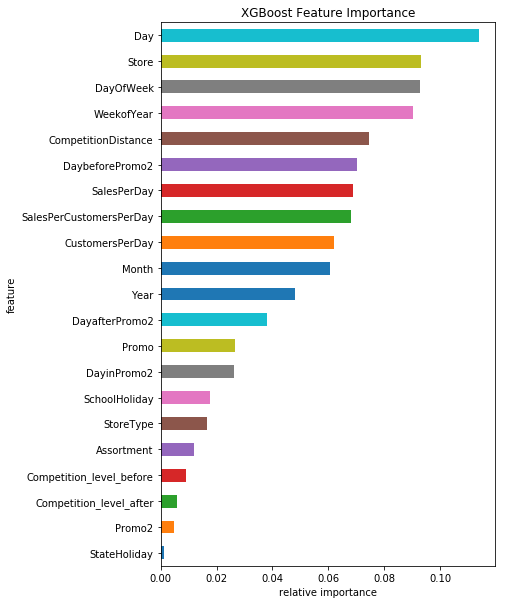
Validating

RMSPE: 0.109707

提交到Kaggle上得到如下分数：

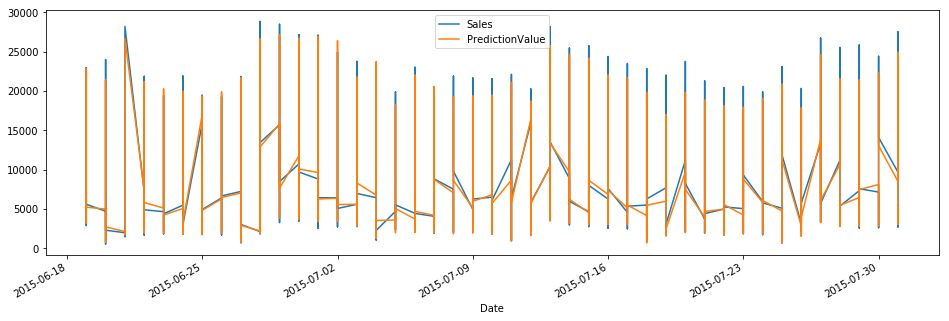


特征重要性如下图



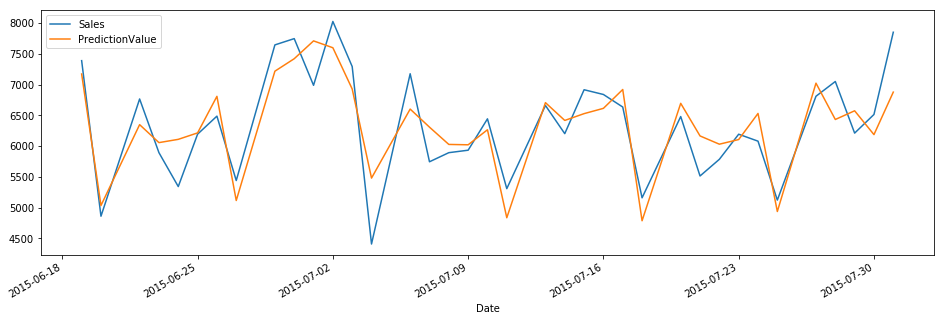
1. **结论**
   1. 可视化

对比Sales实际值和预测值，如下图：



从这个图上可以看出对峰顶的值预测的不够准确，预测值比最大的销售额要小。预测值比实际的最低值又要稍微大一些。

取一个Store的数据观察一下：



可以看出预测值比实际值的上下范围要小一些。

* 1. 思考

当前的模型使用的Learning Rate 是0.01，导致学习的速度很慢，在i5 四核的电脑上，要运行7-8个小时。虽然预测准确度达到了要求，但是模型运行效率还是太低了。

销售预测和“Time Series”[6] 有相关性，可以考虑后续通过使用ARMA[7]模型来构造更多和销量相关的特征，来提升xgboost 模型的训练速度和预测准确性。

还有，Cheng Guo 使用了深度学习模型：Entity‐Embedding 模型[8]，他在Rossman Store Sales 竞赛中以0.10583 的成绩取得了第三名。这是一种完全不同的思路。后面计划学习这种模型并验证预测的结果。

* 1. 后续改进
* 尝试使用 XGBoost 的GPU 模式提升训练速度；
* 使用模型集成技术，融合多个模型；
* 使用Time Serials 模型增加新的Feature；
* 尝试 Entity-Embedding 技术

1. **参考文献**

[1] https://www.kaggle.com/c/rossmann-store-sales

[2] http://blog.kaggle.com/2015/12/21/rossmann-store-sales-winners-interview-1st-place-gert/

[3] https://en.wikipedia.org/wiki/Xgboost

[4] <https://www.kaggle.com/c/rossmann-store-sales#evaluation>

[5] <https://www.kaggle.com/c/rossmann-store-sales/discussion/17979>

[6] <https://en.wikipedia.org/wiki/Time_series>

[7] [https://en.wikipedia.org/wiki/Autoregressive%E2%80%93moving-average\_model](https://en.wikipedia.org/wiki/Autoregressive–moving-average_model)

[8] Cheng Guo’s Entity‐Embedding 模型源代码: https://github.com/entron/entity‐embedding‐rossmann/tree/kaggle