# 机器学习纳米学位

## Rossmann Stores 预测营业额项目

2018 年 12 月 3 日

# I. 问题的定义

## 项目概述

本项目是来自 Kaggle 的一个竞赛项目[1]。Rossmann 药妆店在欧洲的 7 个国家里有 3000 多家连锁店。影响药妆店的营业额的因素很多，比如打折，附近的竞争者，学校假期（寒暑假），国家假期（圣诞节），季节性和本地因素，药店规模和类型，药店装修歇业等等。这些因素都让每家药店的营业额各不相同[2]。

为了药妆店更好的运营，Rossmann 试图找到影响药店营业额的多种因素的潜在模型，从而更好地预测药店的营业额。在本项目中，Rossmann 提供了数据集提供了全德国 1115 家店的 1017209 条数据，时间周期是从 2013 年 1 月 1 日到 2015 年 7 月 31 日止，这 31 个月的数据。要预测的对这 1115 家店，从 2015 年 8 月 1 日到 2015 年 9 月 17 日共六周的营业额。

竞赛的评估标准则是：竞赛者提交的预测营业额的结果，都会对其进行 Root Mean Square Percentage Error (RMSPE)的计算[3]。计算公式如下：

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

其中，$y_i$ 代表了单店的单日营业额，$\hat{y}_i$ 则代表了相应的预测营业额。当 $y_i$ =0，即单店单日营业额为零。

在计算 RMSPE 的过程中，这种数据会被自然忽略。

选取 RMSPE 作为评估度量的好处在于：它是完全独立于数据集的大小的，这样可以拿来比较评估不同规模的数据集。当然，也存在一个问题，那就是 RMSPE 永远为正，且没有上限，该值在实际应用中很容易形成 right-skewed 的情况[4]。

## 问题陈述

在本项目中，Rossmann 提供的训练集中包含全德国 1115 家店的 1017209 条数据，时间周期是从 2013 年 1 月 1 日到 2015 年 7 月 31 日止，共 31 个月的数据。需要预测的是对这 1115 家店，从 2015 年 8 月 1 日到 2015 年 9 月 17 日共六周的营业额。

由于我的电脑配置较低，为了减少运行和优化的时长，我只选取数据集中 1115 家店中的前 400 家店进行预测分析。这样就把训练集的数据减少到 303023 条，测试集的数据减少到 14736 条，约占到原数据集的 30%。

于是本项目的任务目标就是：根据这 400 家店在过去 31 个月的历史经营数据，以及每家店的额外补充数据，如何尽量准确地预测 2015 年 8 月开始的六周时间内的单日单店营业额，从而让 RMSPE 越低越好。不过，我本次并不在 kaggle 上提交我的预测数据集（但我会按照 kaggle 的格式要求，生成预测数据集），而这六周的营业额在 kaggle 上并没有公布，因此我无法根据营业额的预测值和实际值来计算 rmspe。为了通过 rmspe 来评估本模型，我需提前从 train 数据集中切分出一部分数据来 （2015 年 7 月 1 日-2015 年 7 月 31 日的整个月数据），用于后续的 rmspe 计算，模型验证和评估。

上述任务目标，就变成了一个有监督的回归问题，而回归是通过 XGBOOST 的算法来建模优化的。

## 评价指标

在项目概述中已经提到，竞赛的评估标准则是：竞赛者提交的预测营业额的结果，都会对其进行 Root Mean Square Percentage Error (RMSPE)的计算[3]。计算公式如下：

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

其中， $y_i$ 代表了单店的单日营业额， $\hat{y}_i$ 则代表了相应的预测营业额。当 $y_i$ =0，即单店单日营业额为零。

在计算 RMSPE 的过程中，这种数据会被自然忽略。

选取 RMSPE 作为评估度量的合理性和好处在于：它是完全独立于数据集的大小的，这样可以拿来比较评估不同规模的数据集。因此虽然我选取的数据集只是原来数据集的一部分（原来的 30%左右），同样可以用 RMSPE 来评估。

在 Kaggle 上的此次竞赛，最终吸引了 3303 名竞赛者提交了结果。其中第 330 名（10% percentile）的 RMSPE 得分为 0.11773。我会尽量在我的小数据集上跑出接近这样的分数，此为本项目的目标基准。

# II. 分析

## 数据的探索

### 1. 数据集信息

本项目总共提供了四个数据文件：

- train.csv - 包含了营业额及顾客数的历史数据
- test.csv - 不包含营业额及顾客数的历史数据
- sample_submission.csv - 提交文件的正确模板文件
- store.csv - 每家店的补充信息

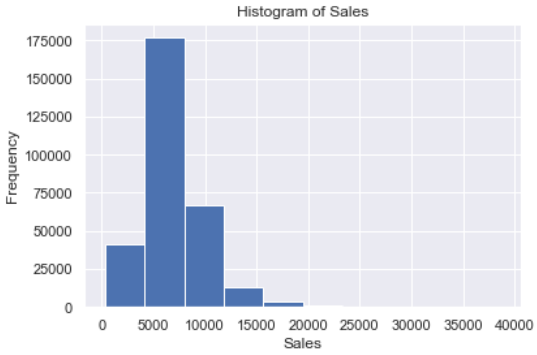
数据集中相关列的说明：

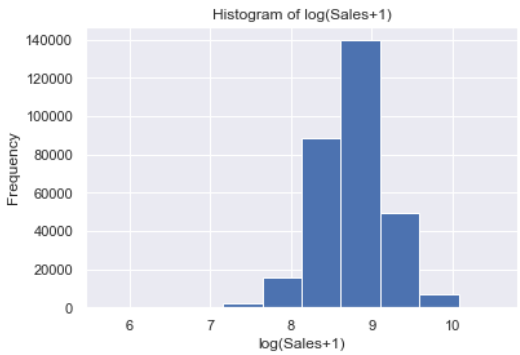
- Id – 只在 test set 中有的，对于每家店和营业日期的组合 ID。整数型变量

- Store – 每家店独有的 ID，train set 和 test set 中都有。整数型变量
- Sales – 单店营业额，也是我们的预测对象。整数型变量
- Customers – 单日顾客数。整数型变量
- Open – 是否营业，0 = 关店, 1 = 营业，整数型变量
- StateHoliday – 是否为国家公共假期。通常在公共假期，基本上所有店都会关门，除了极个别的情况。同时，学校会在所有公共假期和周末关闭。a = 公共假期, b = 复活节, c = 圣诞节, 0 = 非假期日。字符型+整数型变量
- SchoolHoliday – 学校假期，0 = 非假期, 1 = 放假。分类整数型变量
- StoreType – 有四种店的类型: a, b, c, d。分类字符型变量。
- Assortment – 进一步对店进行分类: a = 基础店, b = 额外店, c = 延展店。分类字符型变量
- CompetitionDistance – 离最近的竞争者的距离，米。小数型变量
- CompetitionOpenSince[Month/Year] – 最近的竞争者的开店时间（月和年）。小数型变量
- Promo – 单店当日是否在进行促销活动，只在 train set 中。分类整数型变量。
- Promo2 – 某些店进行的一些持续打折活动: 0 = 该店不参与促销, 1 = 该店参与促销。只在 test set 中。分类整数型变量。
- Promo2Since[Year/Week] – 单店开始参与 Promo2 促销的开始年份和周数。小数型变量
- PromoInterval – 单店开始 Promo2 促销的月份间隔，数值型。比如 "Feb, May, Aug, Nov" 表示该店每逢 2 月，5 月，8 月和 11 月进行促销。字符型变量

## 2. 新的特征变量的创建及变换

- Year – 从 Date 中提取的年份。整数型变量
- Month -从 Date 中提取的月份。整数型变量
- Day – 从 Date 中提取的天。整数型变量
- WeekOfYear – 一年中的周数。整数型变量
- DayOfYear – 一年中的天数。整数型变量
- SalesPerCustomer – 单位顾客的零售额。小数型变量
- log(Sales+1) – 对零售额的对数变换，小数型变量（以下两张 Sales 的直方图，可以看出 sales 的分布严重右偏。进行了对数变换的分布比较接近正态分布，同时+1 是为了避免 Sales=0 的情况）
- CompetitionOpen – 竞争者开店持续的时间（以月计）。小数型变量
- PromoOpen - Promo 持续的时间 (以月计）。小数型变量




综上，最终用于模型训练的所有特征变量为如下列表（共 22 个）：

1) Store ,

2) DayOfWeek ,
3) Date ,
4) Open ,
5) Promo ,
6) StateHoliday ,
7) SchoolHoliday ,
8) Year ,
9) Month ,
10) Day ,
11) WeekOfYear ,
12) DayOfYear ,
13) StoreType ,
14) Assortment ,
15) CompetitionDistance ,
16) CompetitionOpenSinceMonth ,
17) CompetitionOpenSinceYear ,
18) Promo2 ,
19) Promo2SinceWeek ,
20) Promo2SinceYear ,
21) CompetitionOpen ,
22) PromoOpen

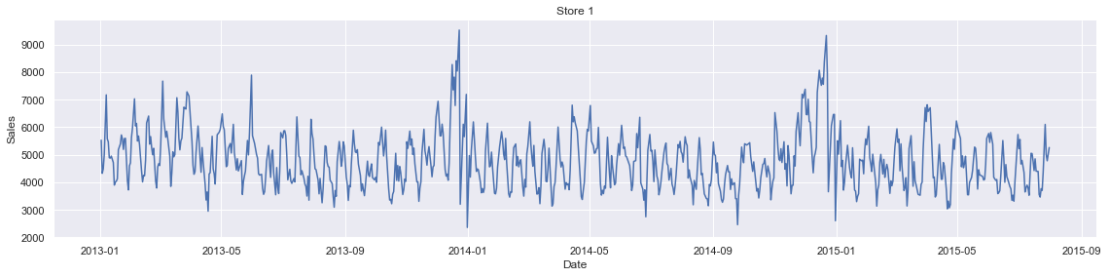
### 3. 数据异常值，缺失值的处理

数据集中存在的缺失值和异常值，都已经在数据预处理和训练模型的部分被检测和处理了，请见附加的 Rossmann_sales.ipynb 文件。

## 探索性可视化

由于 EDA 可视化的图片过多，探索性可视化的图片和讨论的完整版都在 Rossmann_sales.ipynb 文件中。在这份报告中，只选取部分重点图片和相关的讨论。

### 1. 查看单店随时间序列的趋势





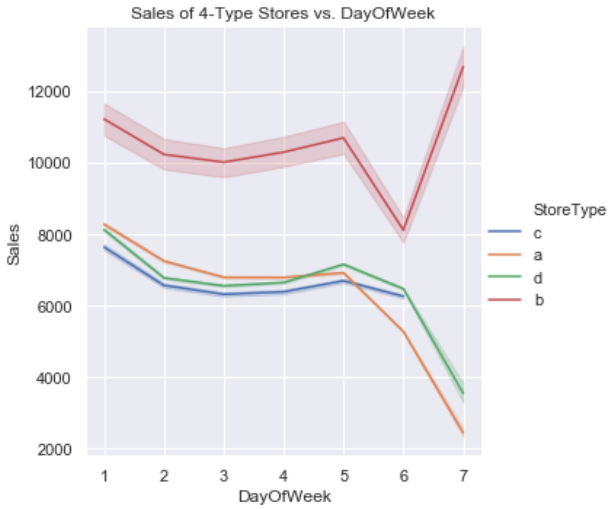
以上几家店这 31 个月中的日营业额没有明显的增加或减少的趋势。季节性的趋势也不明显，除了每年 12 月底的圣诞季有明显的营业额增加的情况。Store100 中有明显的两次关店装修的情况，以及一个零售额的异常值。

## 2. 查看四种店的类型对营业额的影响


以上为四种类型的店的营业额，从 1-12 月的变化趋势。四种类型的店的变化趋势类似，都是从 1 月到 5 月稳步增长，然后在 5-8 月区间，略有下降。从 8 月-12 月，四种店的营业总额呈现非常明显的增长趋势。在圣诞节的假期季，总的销售额达到全年最高峰。

Type b 的店的总营业额远远大于另外三种的店（Type a,c,d）的营业额。其它三种类型的店的总营业额很接近。Type b 的总营业额远远大于另外三种店，这是因为这种类型的店的数量较多。以上四种类型的商店的营业额平均值比较接近，从而证实了这个猜测。


每年 1-12 月，四种类型的店中，单位客人的零售额呈现非常明显的分化趋势：Type d 基本是 Type b 的两倍，而 Type a 和 Type c 基本持平。从竞争者的距离来看，Type d 的平均距离远大于 Type b，Type a 和 Type c 的平均距离比较接近。远离了竞争者，这可能是能促进销售的很重要的一点。这点在下面的 paired correlation matrix 也得到了验证。



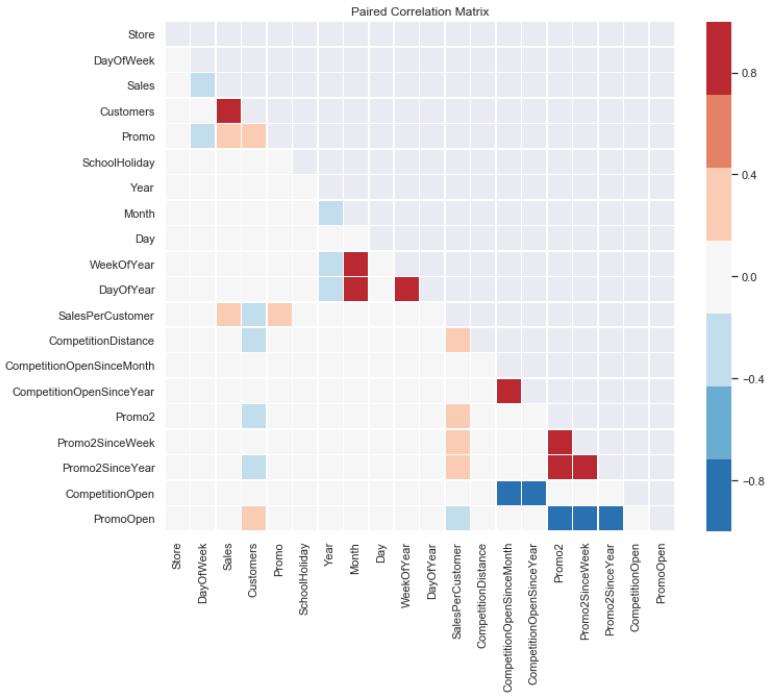

Type c 店周日是不营业的。a 和 d 在周末的销售额急剧下降，而 b 在周六零售额下降，却在周日急剧上升。同时，大众会倾向于在每周的前五天内购物，尤其是周一的零售额的增加非常明显。


四种店在月内，呈现相同的趋势。大家在月初和月末的购买的量比较大。而且很明显的，在每个月 12 日和 24 日左右，有两个零售额的低点。月内的这种低点和月初月末的高点，应该和德国两周发一次工资的周期是相关联的。

## 3.查看 Promo 和 Promo2 对零售额的影响


可以看出，Promo 的促销明显能提升销售额。但是 Promo2 的持续性促销，则并没有明显提升零售额的效果。

## 4. Correlation Matrix




由以上配对的 correlation matrix 可以看出，

有正相关性的配对特征有(相关系数在 0 - 0.4 之间，比较弱)：

- Promo - Sales： 促销能促进销售额
- Promo - Customers： 促销能增加顾客
- Promo2 - SalesPerCustomer： 持续的季节性的促销，能促进单位顾客的购买量
- PromoOpen - Customers： 促销的时间越长，能带来的顾客越多
- CompetitionDistance - SalesPerCustomer： 竞争对手的距离越远，单位顾客的购买量越多

有负相关性的配对特征有(相关系数在- 0.4 - 0 之间，比较弱)：

- CompetitionDistance - Customers： 竞争对手的距离越远，带来顾客数越少（这点比较意外）
- Sales - DayofWeek： 大家倾向于在每周前几日购物，而不是周末（这个也许跟周末关门或促销有关系）
- Promo - DayofWeek： 促销通常在每周的前几日进行
- Promo2 - Customers： 持续的季节性促销，反正不能显著带来顾客

# 算法和技术

**算法选择**：排名靠前的参赛者大部分都用监督学习中的集成模型 XGB(Extreme Gradient Boosting) + 决策树的方法，即可得到令人满意的预测结果。XGB 的原理和算法见[6]。当然也有参赛者用了 CNN 的方法进行模拟。MLP 层数有 1 层，也有三层的。但是最终打分都不如 XGB 算法[5]。因此，在本项目中，也采用了 XGB+决策树的算法。

**XGboost 原理介绍**：XGboost 全称为 e**X**treme **G**radient **B**oosting，它属于 Gradient Boosting 的总框架中。但不同于常规的梯度提升树 Gradient Tree Boosting（GBDT 或 GBM），它提供了一种多线程平行树提升的算法，这样能更快更准确[13]。

- **GBDT 的基本原理：**

Gradient Tree Boosting 集成法是对一些基本算法或者弱训练者（这里初始默认的是一层的决策树算法）来进行多轮重复的训练，训练的目的是为了提高分类的准确度，或者说减小分类的错误。

最初，为了让训练的数据集中的每个点维持一定的分布，我们都给予每个点相同的重要性。在训练每一轮之后，针对每片叶子都有分数，这轮中那些分类错误的点的权重就会被增加，这样能够确保在下一轮训练中，弱训练者能够重点关注和训练分类错误的点，从而使上一轮的错误分类的误差最小化。下一轮训练之后，又会有新的分类错误的点出现。同样的，这些错误分类点的权重也会被增强。在权重的增强上，为了让正确分类的点的权重等于错误分类的点的权重，同时又对权重取了对数，这样就保证了权重在-1 到 1 之间区间之中。

训练完之后，我们合并所有训练结果时，将每轮训练区间的值乘以权重值进行相加，得到最终的训练后的区间。所有相加后为正的区间，和所有相加后为负的区间，他们的边缘连线，自然分割开预测为正和预测为负的区间 [14]。

- **XGBoost 的原理和优化：**

XGBoost 原作者是这样解释 xgboost 和 gbm 的区别的：都是基于 gradient boosting 的原理。只是在建模细节上不一样。 xgboost 用的是更正则化的模型，这样可以控制过拟合和提升表现[15]。更详细的数学推导请见这里[6, 16]。

概括来说，就是最小化起始目标函数：

$$\text{obj} = \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t)}) + \sum_{i=1}^{t} \Omega(f_i)$$

其中，$\sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t)})$ 为训练损失函数，$\sum_{i=1}^{t} \Omega(f_i)$ 为正则化项。每训练一次，都会增加一颗新的树，正则化项为每个步长 i 树的函数的加和。

经过定义损失函数为 MSE 进行带入推导，目标函数则变为如下：

$$\sum_{i=1}^{n} [g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)] + \Omega(f_t)$$

通过正则化的数学推导，目标函数如下：

$$\begin{aligned} \text{obj}^{(t)} &\approx \sum_{i=1}^{n} [g_i w_{q(x_i)} + \frac{1}{2} h_i w_{q(x_i)}^2] + \gamma T + \frac{1}{2} \lambda \sum_{j=1}^{T} w_j^2 \\ &= \sum_{j=1}^{T} [(\sum_{i \in I_j} g_i) w_j + \frac{1}{2} (\sum_{i \in I_j} h_i + \lambda) w_j^2] + \gamma T \end{aligned}$$

其中 w 为叶子的得分向量，T 为叶子的数量。

最终目标函数如下，可以用来量化整个树的结构的得分：

$$\text{obj}^* = -\frac{1}{2} \sum_{j=1}^{T} \frac{G_j^2}{H_j + \lambda} + \gamma T$$

XGBoost 并不会穷尽枚举所有树的排列可能性，这样的组合是无限的。它聪明的做法是，每一次只变化并优化树的某一级。每一次的变化之后，树的增加的得分可以通过以下来量化（即把原来的最小化目标函数的问题，转化成了最大化树形变化而带来的得分增加上）：

$$Gain = \frac{1}{2}\left[\frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{(G_L+G_R)^2}{H_L+H_R+\lambda}\right] - \gamma$$

第一项为左边的新叶子的得分

第二项为右边的新叶子的得分

第三项为旧叶子的得分

第四项为新叶子的正则化项。

如果前三项小于第四项，则不增加树的分支。

通过每一次在树的某一级的变化的组合的得分计算，很容易从中找到在该级上的最优变化。依次进行分级优化。

- **XGboost 有如下优势：**

对数据集的规模要求很灵活，可大可小，而准确性很高。基本上数据挖掘比赛项目中，过半的优胜者都使用的 xgboost。

**与 GBDT 的最大区别在于**，对目标函数中的正则化项的处理上。大部分的算法都着眼于损失函数，而忽略了正则化项。而 XGboost 量化了正则化项，把它放入到目标函数中并进行计算，由此对每轮树形变化的得分和相应的树形优化，会跟常规 GBDT 不同，并优于传统 GBDT。

**算法包的选择：**关于 python 自带的 XGBoost (Extreme Gradient Boosting) 和 sklearn 的 GradientBoostRegressor 的比较，可以看出：XGB 包比 sklearn 在对决策树的处理上优于 sklearn 包，所以 XGB 的回归表现上也优于 sklearn，同时速度快，省内存[7]。同时，也有参赛者对 sklearn 包给出的算法结果不太满意[5]。因此在本项目的算法回归上，我选用了 python 的 xgboost 包。在对数据列进行数据预处理时，用了 sklearn 中的 LabelEncoder 非数值型的列进行编码预处理。

**算法的参数选择和优化：**初始参数，我用了优胜参赛者提供的优化参数[11]，但通过回归，发现过拟合的情况比较严重，最后通过调整参数来减轻过拟合的情况。考虑到是否收敛，是否过拟合，rmspe_7 是否接近目标数值 0.11773。

# 基准模型

在 Kaggle 上的此次竞赛，最终吸引了 3303 名竞赛者提交了结果。其中第 330 名（10% percentile）的得分为 0.11773（即 RMSPE）。我会尽量在我的小数据集上跑出接近这样的分数。此为我设定的目标基准。

---

Root Mean Square Percentage Error (RMSPE)的计算公式如下[3]：

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

其中，$y_i$ 代表了单店的单日营业额，$\hat{y}_i$ 则代表了相应的预测营业额。当 $y_i$ =0，即单店单日营业额为零。在计算 RMSPE 的过程中，这种数据会被自然忽略。

选取 RMSPE 作为评估度量的好处在于：它是完全独立于数据集的大小的，这样可以拿来比较评估不同规模的数据集。即使我采用原数据集的一部分，也可以和参赛者的 RMSPE 一同进行比较。

# III. 方法

## 数据预处理

因为空间有限，所有必要的数据预处理步骤，以及异常值的处理，请参见 Rossmann_sales.ipynb

这次改变了对离散变量的预处理方法。上次我使用的是 sklearn 中的 LabelEncoder 对离散变量 StoreType 和 Assortment 进行了编码，但是这种编码更适用于有顺序的离散变量（ordinal categoricals ）。因此我选用 pandas 的 get_dummies 的方法，跟 sklearn 的 OneHotEncoder 的功能一样。但在 pandas 中使用 get_dummies 更方便。

特征选取和变化，请见此报告 II 数据的探索中的第 2 部分以及 Rossmann_sales.ipynb。

## 执行过程

**参数选择**：初始训练模型时，选用了优胜参赛者的提供的参数（详情见 Rossmann_sales.ipynb）。该参数能够在回归模型较快的回归，但是过拟合的情况明显。由于该参数是参赛者针对全数据集采用的，并不一定适用于我的小数据集。因此为了减轻过拟合的情况，以及尽量让 rmspe 靠近 0.11773 的目标值，我对 xgb 的参数进行了微调。

这次在如下参数中做了多种组合的尝试，当然也包括了比赛优胜者和上个版本的参数，rmspe 的分数在 0.130-0.132 的范围内。大部分组合都给了 0.130 的得分，略低于上个版本的 0.132 的得分。

最终选择如下的参数：

```
params = { 'objective': "reg:linear",
           'booster': "gbtree",
           'eta' : 0.01,       # learing rate，从 0.02 降到 0.01，可以减少过拟合，但需同步增加 num_round
           'max_depth': 10,    #  树的深度从 12 降到 10，可以有效减少过拟合，default = 6
           'subsample': 0.8,    #  从 0.9 降到 0.8，20%的几率随机生成树，这样可以避免过拟合,default =1
           'colsample_bytree':0.7, #也可以降低过拟合的情况，default = 1
           'seed': 0}
```

这个模型分数为 0.129，被认为是多种组合中的最优选择。

**异常值的处理**：最开始的回归模型，我是没有处理异常值的，结果用以上的参数跑出的 rmspe_7 为 0.165。因为 rmspe 对异常值比较敏感，因此我尝试将异常值删除。当营业额的标准偏差 std 三倍于营业额的平均值时（此处的阈值 3 为默认值），则该营业额被认为是异常值。根据这个标准，训练集中的 1.6%和验证集中的 1.5%的数值被当作异常值删除了。在删除了异常值的训练集中再次训练模型，获得的 rmspe_7 为 0.132，大幅度降低。可见异常值对误差的影响。

## 完善

xgb 模型的初始参数如下（基于优胜者提供的参数[11]）：

```
params = { 'objective': "reg:linear",
           'booster': "gbtree",
           'eta' : 0.02,
           'max_depth': 12,
           'subsample': 0.9,
           'colsample_bytree':0.7,
           'seed': 0}
 num_round = 5000
xgb_1 = xgb.train(params, dtrain, num_round, evallist, early_stopping_rounds = 10, feval = rmspe_xg, verbose_eval = 50)
```

在之后改变了离散变量的编码方法之后，通过多次微调参数，rmspe 得分基本在 0.129 - 0.132 范围内。

然后又回到训练集，降低了异常值阈值，将删除的异常值从之前的训练集的 1.6%降到 4.0%，再在之前 0.129 的模型上跑出了 0.124 的 rmspe 得分。

最终参数如下：

```
params = { 'objective': "reg:linear",
          'booster': "gbtree",
           'eta' : 0.01,       # learing rate，从 0.02 降到 0.01，可以减少过拟合，但需同步增加 num_round
           'max_depth': 11,   #  树的深度从 12 降到 11，可以有效减少过拟合，default = 6
           'subsample': 0.9,   # 10%的几率随机生成树，这样可以避免过拟合,default =1
           'colsample_bytree':0.7,   #也可以降低过拟合的情况，default = 1
           'seed': 0}
num_round = 10000
xgb_2 = xgb.train(params, dtrain, num_round, evallist, early_stopping_rounds = 10, feval = rmspe_xg, verbose_eval = 200)
```

# IV. 结果

# 模型的评价与验证

这个模型的建立是通过如下的流程来完成的：

1. 数据集预处理&特征构建
2. 可视化&探索性数据处理 EDA&特征选择
3. 训练模型
4. 对训练模型的可视化及优化
5. 对模型的讨论

在建立模型时，我花了大量时间阅读参赛者的经验讨论和优胜者的方法。在构建特征，算法选择和参数选择上，都采用了参赛者的一些方法和建议。

比如在特征的构建上，优胜者提到的一些证明了无用的特征（比如天气因素等）[12]，这些我在构建时就直接避开了。不过优胜者并没有很明确的说明，他到底用了哪些特征，哪些特征有效。因此我也结合了其他排名靠前的竞赛者的意见，并通过我自己的 EDA 发现，从而创建了以上 22 种特征，作为模型的输入。

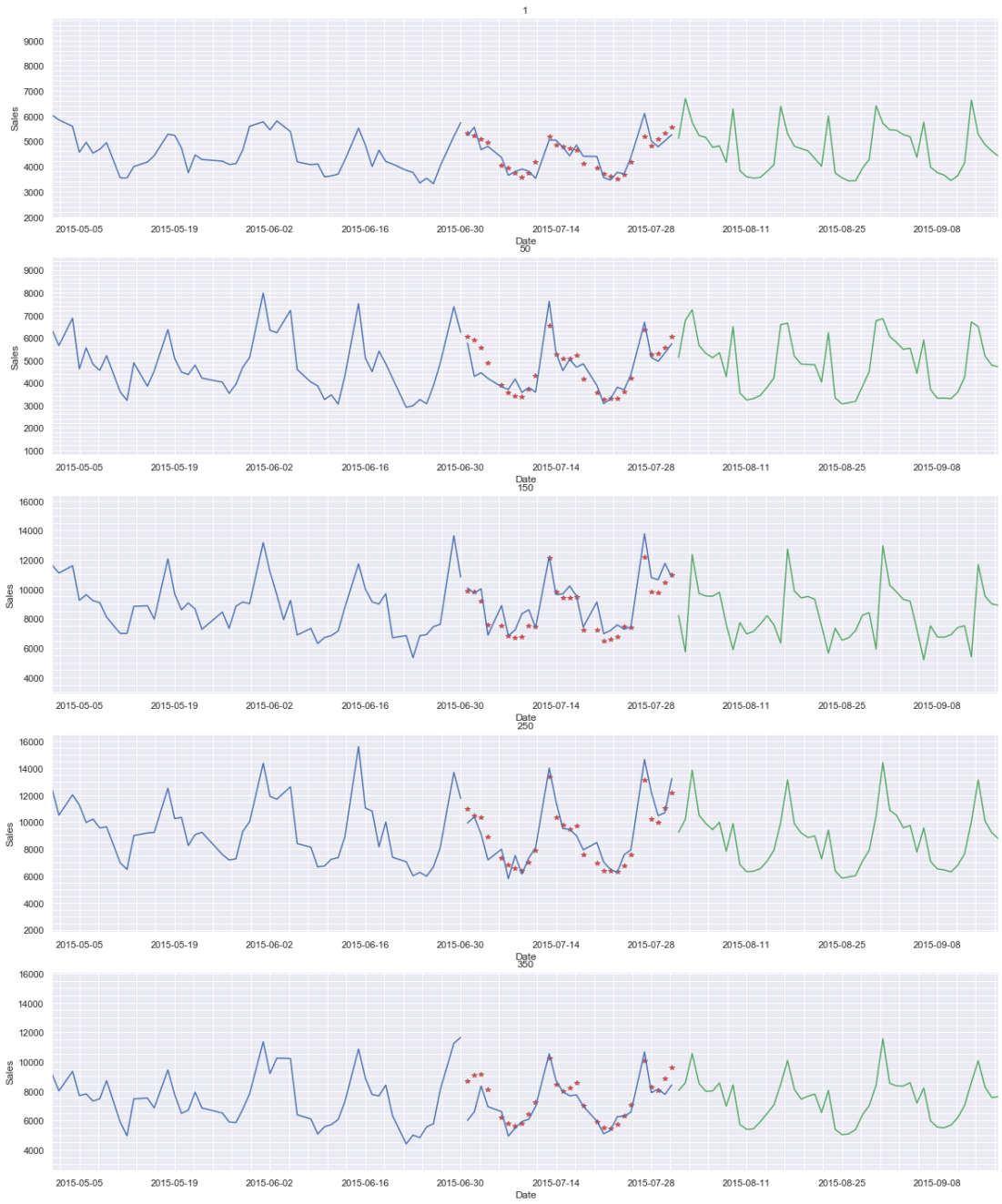
在算法的选择上，也是直接采用了大部分的参赛者所使用的 xgb + 决策树的集成回归方法。因为他们都证明了这个方法的有效性和优势。模型初始参数的选择也是借鉴了优胜参赛者的参数。只是在后期针对我自己的模型进行了参数的调整。调整了参数之后，模型的过拟合情况得到有效的减轻，同时也给出了不错的预测（见下图的预测可视化）。

在删除了异常值之后，我的回归模型的 rmspe 从 0.165 降到了 0.132。 然后在重新用 get_dummies 处理了离散变量之后，再次参数微调得到了最低的 0.129。为了进一步降低 rmspe 分数，我又重新设定了异常值的阈值，从之前的 3 降到 2，把异常值占比训练集的比重从之前的 1.6%，提高到了 4.0%。然后再用之前的参数最优模型再次进行建模，从而得到了 0.124 的分数，在 kaggle private leaderboard 上排名 1260 位。

虽然这个跟我的 0.11773 的目标还有差距，但是在花费了近 100 个小时后，我对这个模型结果还是相当满意的。当然要让 rmspe 进一步降到接近 0.11773，这个已经无法通过微调回归参数来解决了，而是需要通过重新回到特征工程，重新发掘和构建新的更有效的特征，来构建新的模型。

综上，从预测的可视化和 rmspe = 0.124 来看，这个模型还是具有一定的合理性和可信度。

从鲁棒性的角度来说，rmspe 确实对异常值比较敏感，从删除训练集中 1.6%异常值的情况来看，rmpse 下降了 20%。继续删除训练集中 4%异常值的情况，rmpse 又下降了 6%。而且所有参赛者也不知道 test 数据集中是否存在异常值。当然既然大家的提交结果都是通过 rmspe 来计算得出的，那么对大家来说都是公平的。为了优化模型的鲁棒性，后续可以进行对数据的异常值的处理上多做研究，比如通过 EDA，或者减小定义异常值的阈值，从而可以考虑去掉更多的异常值，也许能够进一步减少误差，增强鲁棒性。



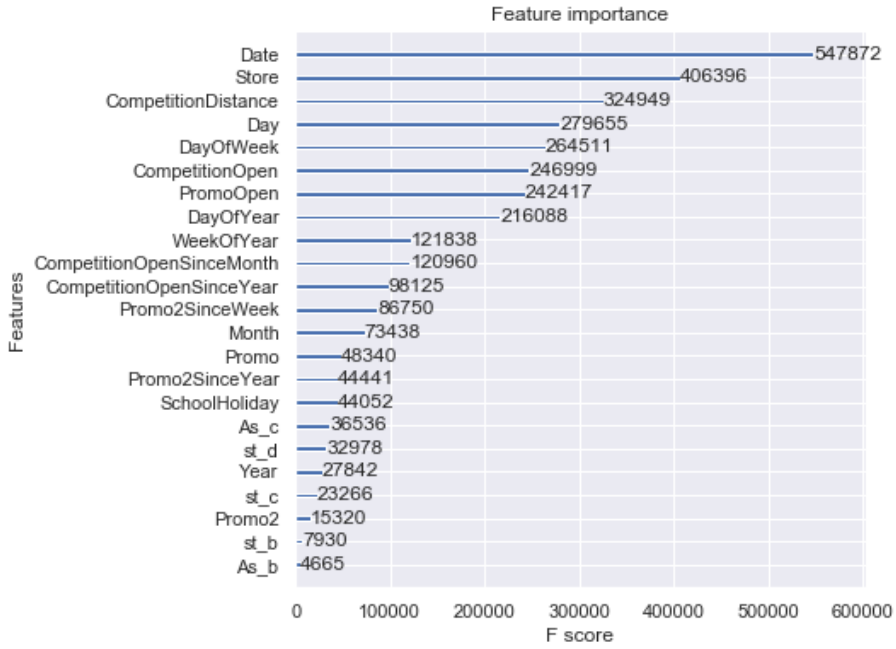

## 合理性分析

我的最终模型能够对 test 数据集中做出预测，同时对验证集给出的预测，以上图中红色点的预测值，呈现出了正确下降和上升的趋势，而且在大部分的单日营业额的预测上，还是有一定的准确度。评价指标 rmspe = 0.13204，虽然我之前设定的 0.11773 的目标还是有差距，但是 0.1237 的得分能够在项目总排名中排到 1260 名/3303 名（Top 38.1%）。这个预测误差也能够接受，因此存在一定的可信度。

# V. 项目结论

# 结果可视化

**1.模型的特征相关性排名**




从以上最新的特征排名来看，特征的重要性排名依次为：Date, Store, CompetitionDistance, DayOfWeek,Day，等等。可见日期和店的差异性还是具有非常差异化的重要性。除去时间和店的差异化的情况，对营业额起到重要作用的特征有：CompetitionDistance,PromoOpen,CompetitionOpen,Promo2SinceWeek。可见，竞争者的距离和开业情况，以及两种促销都会影响到营业额的变化。而商店的 StoreType 以及 Assortment 来看，重要性垫底。这也就解释了为什么针对这两个变量改变了编码方法之后，对模型的分数提升作用不太大。

同时需注意到，排名第一名和第二名分别是 Date 和 Store，说明时间序列和店之间存在很大的差异，而特征工程并没有有效地捕捉到有规律的时间特征和店的特征。

2.对验证数据集的预测可视化

从上部分的预测可视化图中（IV.结果-模型的评价与验证中），可以看到，红色点的预测值，能够正确地预测出下降和上升的趋势，而且在大部分的单日营业额的预测上，还是有一定的准确度。当然在更细化的预测上，有些点的预测距离实际值还是有偏差。这个需要通过进一步优化模型来改进。

# 对项目的思考

本项目的详细流程如下：

- 首先是大量地阅读参赛者的讨论和经验总结，这个给我在特征工程，算法和参数的选择，算法包选择上，都给与了很重要的参考，同时也避免了重复一些无用的工作（比如那些外部的资料如德国的天气资料，google trend 是否有效等），节省了很多时间。之后才开始了如下的探索。
- 导入数据集

- 了解数据集，可视化，异常值和缺省值的检测
- 进一步可视化&探索性数据分析 EDA，通过这一步发现有关联的特征
- 结合参赛者的对有效特征的经验，同时通过 EDA 的对相关特征的自行探索，进行了特征创建和特征选择。
- 对训练模型进行预处理，根据初始参数来训练模型
- 根据初始运行结果，调整模型参数，处理异常值，再次训练模型。根据运行结果再次优化参数和训练。
- 对训练模型的可视化，查看是否能有效的预测零售额，并比较最终模型的 rmspe 值是否达到目标
- 对模型的讨论和思考进一步优化模型的方向

比较有困难的地方，在于前期的特征工程。参赛优胜者提到他花了大量的时间进行特征工程，创建了上百种特征，并测试了上 500 种模型，才优选出现在的 20 来种特征。最终他也没有明确指明他优选的特征有哪些。而我对于优选特征欠缺有效的方法。

比较有意思的地方，对建模来预测实际营业额，而且是这么大的数据集，这本身就是一个很有意思的事情。

我对最终的模型还是比较满意。因为模型的可视化中，能看出其预测的有效性，同时 rmspe = 0.124 这个误差在合理范围内。

在通用的情景下，比如相同的数据挖掘预测的项目中，这个项目流程和探究方法基本是一样的。只是在细化的特征工程中，算法选择和参数选择上，是无法通用的。每一个项目的回归模型都不会相同。

# 需要作出的改进

1. 更多的特征工程和选择：

    a) 此次竞赛第一名反复强调了，他花了 80%左右的时间在进行特征工程，选了上百种特征，跑了大约 500 个随机模型，最终在相同的模型上跑了十几对不同的特征，才拿到这样的分数。遗憾的是，他并没有明确说最后选定了哪十几个特征值。其他优胜者也基本做了大量的特征选择工作，最终选定了 10-20 种特征值。

    b) 下一步我还需要花更多的时间来进行特征的创建和选取。以上 rmspe_7 = 0.1237 的得分，很难再通过对 xgb 的参数进行微调来大幅度降低。必须通过更多的特征探索来优化模型，尤其是在时间序列的探索上和降低各店差异化的特征探索上。从回归的 xgb_2 的重要性排名上，排名第一名和第二名分别是 Date 和 Store，说明时间序列和店之间存在很大的差异，而特征工程并没有有效地捕捉到有规律的时间特征和店的特征。这些都需要在下一步的特征工程中深入探究。

    c) 尝试非监督学习中的 PCA 和聚类等来对不同的店进行特征归类。

    d) 特征工程方案，大致分为五大类：时间特征，假期特征，促销特征，竞争对手和商店本身的特征。可以考虑从这几个角度进行特征构建，比如：

        - 利用特征 CompetitionOpenSinceYear，CompetitionOpenSinceMonth 以及新特征 Year，Month 构建新的特征 CompetitionOpen，表示最近竞争对手开业有多少个月的时长。
        - 利用特征 Promo2SinceYear，Promo2SinceWeek 以及新特征 Year，WeekOfYear 构造新特征 PromoOpen，表示持续促销开始有多少个月的时长。

2. 时间序列的分析（TSA）和 arima 模型探索。正如在开题报告中所说，有多位参赛者提到了 TSA 和 arima 模型，甚至有参赛者进行了 tsa 模型和 xgboost 的模型的叠加，得分跟 xgboost 接近。由于 TSA-arima 是常用的预测带

有季节性时间周期的有效线性回归模型，下一步可以考虑加入 arima 模型来捕捉到更多的时间周期和季节性的变化，优化时间序列相关的特征选择。

3. 删除更多的异常值。本次模型中，我把阈值设定为默认的 3，从而分别从训练集和验证集中删除掉仅仅 1.6%，1.5%左右的异常值。由于 rmspe 对异常值比较敏感，我没删除异常值时，跑出的 rmspe_7 的数值为 0.165 左右，而在删除掉 1.6%左右的异常值之后，rmspe 的得分从 1.65 有效的降低到 1.32。但其实全数据集中 1.6%的异常值，不是太多。继续删除训练集中 4%异常值的情况，rmpse 又下降了 6%。可以考虑降低阈值，从而删除更多的异常值，有望进一步降低 rmspe 得分值。

4. 算法选择上，也可以尝试 lightGBM。lightGBM 也是基于 GBDT 框架的算法。不同于传统的 GBDT 建模过程中在树的深度的每一级上进行生长，而 lightGBM 是直接在树叶上进行生长和拟合，这样‘贪婪’的算法，极大的减少了损失函数，提高了准确率，同时极大减少训练的时间[17]。因此，lightGBM 能够用来处理大的数据集和大量的特征。它的预测准确率基本跟 xgboost 媲美或略高[17, 18]，但是训练时长却只是 xgboost 的百分之几[18]，这对于本项目的大数据集的训练是大有裨益的。

5. 算法上，还可以尝试比赛第三名使用的带有 entity embeddings 的神经网络算法[19，20]。Entity embeddings 可以用来有效地处理分类变量，进行数据聚类分析，降低特征维度。在本项目中，entity embeddings 可以用来进行特征选择和数据的预处理。

# 参考资料


[1] https://github.com/udacity/cn-machine-learning/tree/master/Rossmann_Store_Sales
[2] https://www.kaggle.com/c/rossmann-store-sales#description
[3] https://www.kaggle.com/c/rossmann-store-sales#evaluation
[4] https://cssd.ucr.edu/Papers/PDFs/MAPE-R%20EMPIRICAL%20V24%20Swanson%20Tayman%20Bryan.pdf
[5] https://www.kaggle.com/c/rossmann-store-sales/discussion/17896
[6] https://xgboost.readthedocs.io/en/latest/tutorials/model.html
[7] https://stats.stackexchange.com/questions/282459/xgboost-vs-python-sklearn-gradient-boosted-trees
[8] https://machinelearningmastery.com/arima-for-time-series-forecasting-with-python
[9] https://www.kaggle.com/c/rossmann-store-sales/discussion/16930#97601
[10] https://www.kaggle.com/c/rossmann-store-sales/leaderboard
[11] https://www.kaggle.com/abhilashawasthi/xgb-rossmann
[12] https://www.kaggle.com/c/rossmann-store-sales/discussion/18024
[13] https://github.com/dmlc/xgboost
[14] https://www.datacamp.com/community/tutorials/xgboost-in-python
[15]https://www.quora.com/What-is-the-difference-between-the-R-gbm-gradient-boosting-machine-and-xgboost-extreme-gradient-boosting
[16] https://homes.cs.washington.edu/~tqchen/pdf/BoostedTree.pdf
[17] https://www.analyticsvidhya.com/blog/2017/06/which-algorithm-takes-the-crown-light-gbm-vs-xgboost/
[18] https://papers.nips.cc/paper/6907-lightgbm-a-highly-efficient-gradient-boosting-decision-tree.pdf
[19] https://arxiv.org/pdf/1604.06737.pdf
[20] https://github.com/entron/entity-embedding-rossmann