基于 XGBoost 的 Rossmann 商店销量预测分析

朱林

2018/4/5

一、问题定义

1.1、项目概述

在商业数据分析领域，对于分析的准确性要求越来越高，之前的主观推测方法，由于掺杂了很多人为因素，容易导致误判，所以很难满足要求。如今随着数据量增多、计算机算力提升迅速这两个因素的影响下，机器学习的方法有了天然的土壤，于是有很多机器学习的方法得以运用起来。本研究是基于欧洲一家连锁药店-Rossmann 药妆商店的数据进行的分析，数据包括了：1115 家商店的相关信息、一百万多条关于上述商店的日销量等信息。分析目的是，试图通过给定的销售量等数据，分析得出在接下来的一段时间内，上述商店的销售情况。

首先要对德国的商业环境有所了解，德国的法律规定，只有在少数周日商店才可以开门。[1]可想而知，周日的销量可能不会和 weekday 的销量有太大差别。但是由于周日不开门，因此周六商店会开门很长时间，所以周六的销量可能会大大高于平日。基于上述信息，首先对给出的数据有有一定的了解。kaggle 给出的数据主要包括三类：store-商店相关信息；train-商店的日销量等信息；test-测试数据。[2]有哪些因素会影响一家商店的销量呢？日常逛街的时候，主要关注的信息有：星期几、商店是否有折扣，是否是假期、商店类型、到店人均消费。另外，还会关注周边的信息，比如商店和其他类似商店的远近比较。这些信息大多数都在给定的数据中可以找得到，另外一些需要经过数据处理才能得到。当然，还有一些周边信息，没有在数据中体现，比如当时的天气状况、气温等信息，这些信息也会影响人们去商店的意愿。

上述提到的机器学习方法中，有监督学习和非监督学习、半监督学习。另外现在图像识别问题中最常用的应该是深度学习方法。随着 Alpha Go 的巨大成功，宣告人类在围棋这个极其复杂的棋种，已经不可能超越机器了，机器也不再像以前一样需要人工输入大量棋局和指导，因为他们能够自己进行学习，强化学习让大家眼前一亮。该问题中，由于我们是基于给定的数据和标签结果，通过计算得出我们的判断，因此我们需要使用的是监督学习模型。

---

[1] 张慰. 德国周日与节日法律保护之评述. 德国研究. 2017

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

1.2、问题陈述

由于需要预估的销售量，是一个连续的数据，因此该问题属于一个回归问题，而且是有监督的回归问题。有监督学习的模型中，我选择 XGBoost 作为训练结果，因为他作为 boost 类别的方法，能够集合多个弱分类，综合判断得到结果，往往比单个模型得到的结果要好。

该问题的数据分为商店数据和训练数据，所以首先需要进行合并操作。其次，需要对数据进行一些预处理，去掉一些误差点等信息。最后再进行 XGBoost 模型的训练和调参。

由于该问题是基于 kaggle 的项目，因此项目最终结果需提交 kaggle 得到最终的结果。总共有 3303 支队伍参赛，评价标准是，起码需要达到总排行榜的前 15%，力争进入前 10%。由于有两个排行榜，Private 榜 和 Public 榜，分别表示随机取所有数据的 61% 和 39%。因此我选取更小随机性的 Private 榜作为参照，进入前 15% 和 10% 的分数分别为：**0.11959** 和 **0.11773** 。

1.3、评价指标

最终需要输出一个 csv 文件，包含所有 test 数据的预测销量结果，将其提交到 kaggle 进行评价。该项目 kaggle 基于的评价指标 RMSPE，表示均方根误差的百分比值，能够将误差放到同一个量纲上进行比较。它的计算公式如下：

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

y_i ： 销量误差的实际值
yhat_i ： 销量误差的预测值
n ： 测试集的数目
当然，该判断中，排除了销量为 0 的数据不计算在内。

二、分析

2.1、数据的探索

数据集包括三个表格：

store.csv-商店相关数据：

"Store"：int，商店编号，是商店唯一编码
"StoreType"：object，商店类型，a,b,c,d
"Assortment"：object，产品级别，a=basic,b=extra,c=extended
"CompetitionDistance"：float，竞争对手距离，可以看成是，李竞争对手越近，顾客被分流的可能性就越大
"CompetitionOpenSinceMonth"：int，竞争对开门月份，和竞争对手开门年份合起来组成竞争对手开门时间，表示可能受到竞争对手影响的时间长度。
"CompetitionOpenSinceYear"：int，竞争对开门年份，同上。
"Promo2"：int，是否开展促销 2。该促销是一个长周期的促销活动。0 代表商店未参与该活动，1 代表商店参与该活动。
"Promo2SinceWeek"：int，开展促销 2 的周。和促销 2 年份字段一起，表示促销 2 开展持续了多久。
"Promo2SinceYear"：int，开展促销 2 的年份，同上。
"PromoInterval"：object，促销间隔，表示促销 2 在一年的哪些月份开展。刚开始进行促销的时候，往往对于用户的吸引力最大。所以开始开展促销 2 活动的月份字段也是很重要的。

train.csv-训练数据：

"Store"：int，商店编号，是商店唯一编码
"DayOfWeek"：int，周几
"Date"：date，这条数据的日期
"Sales"：float，销量，该字段是我们需要预测的字段
"Customers"：int，客户数量，该字段在测试数据中不存在，所以不能依照该字段进行预测，但是为了充分榨取所有字段的剩余价值，将该字段与其他字段进行组合，构成 store 的特性，例如：客单价、每日客户数量等字段。
"Open"：int，该商店在当天是否开门，训练数据中有几条记录是没有开门与否的信息的，一般情况下是将其丢弃，但是我认为数据都有其存在的价值，这里不妨将其设置成 open=1，即当天该商店是开门的状态。
"Promo"：int，是否进行促销活动，明显，由于是否进行促销对于当天的销量影响不小，因此这个字段也颇为重要。
"StateHoliday"：object，当天是否是国家法定假日，如果是假日，对于销量也会有很大影响，不同类型的国家假日对于销量的影响情况也不一样。a = public holiday, b = Easter holiday, c = Christmas, 0 = None。
"SchoolHoliday"：int，学校假日，表明当天该商店是否受到学校房价的影响，比如学校放寒假、暑假。依据经验，寒暑假的时候，小孩在家，大人应该会有更多时间陪小孩，所以逛街的可能性也会增加。

test.csv-测试数据

"Id","Store","DayOfWeek","Date","Open","Promo","StateHoliday","SchoolHoliday"
字段信息如上述描述所示。另外需要注意的是，最终进行测试的数据不会只有这几个字

段，还应该包括之前提到的客单价等派生信息。

2.2、探索可视化

2.1.1、销量分布情况：

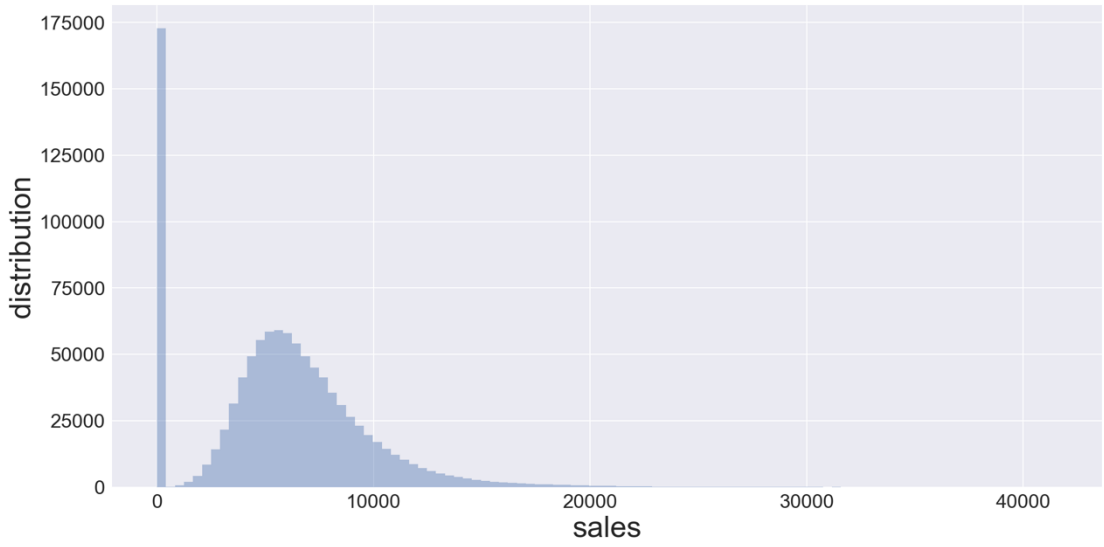

图 1

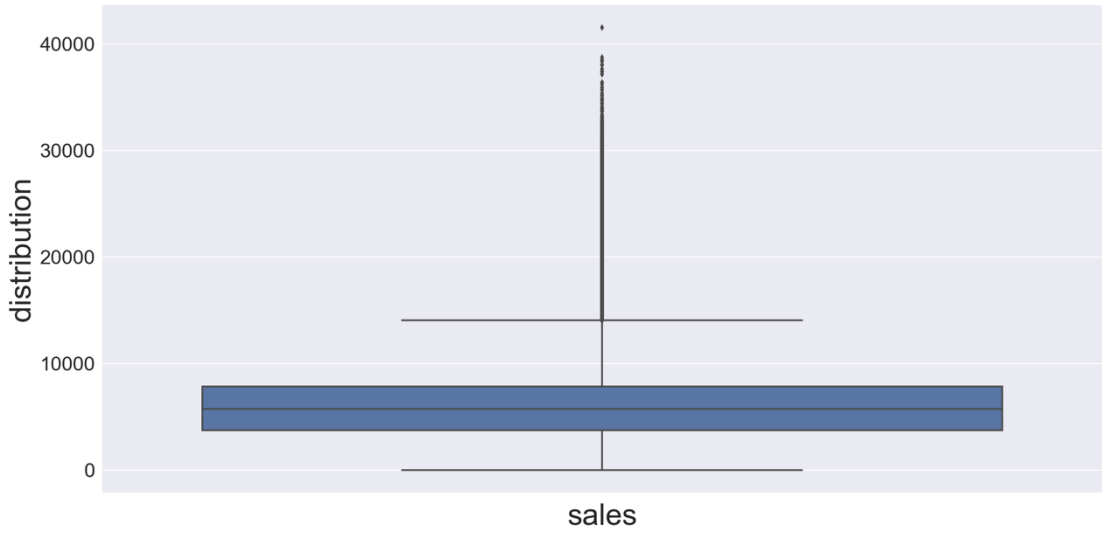

图 2

结论：

图 1，可以看出，除开高达 150000+条销量为 0 的记录外，其余数据还是比较服从期望销量为 5000 左右的近似正态分布。

图 2、可以看出，Sales 的分布状况是一个长尾分布，超过 40000 之后的分布就很少了，可能存在离群值需要处理。

2.1.2、DayOfWeek 分布状况：

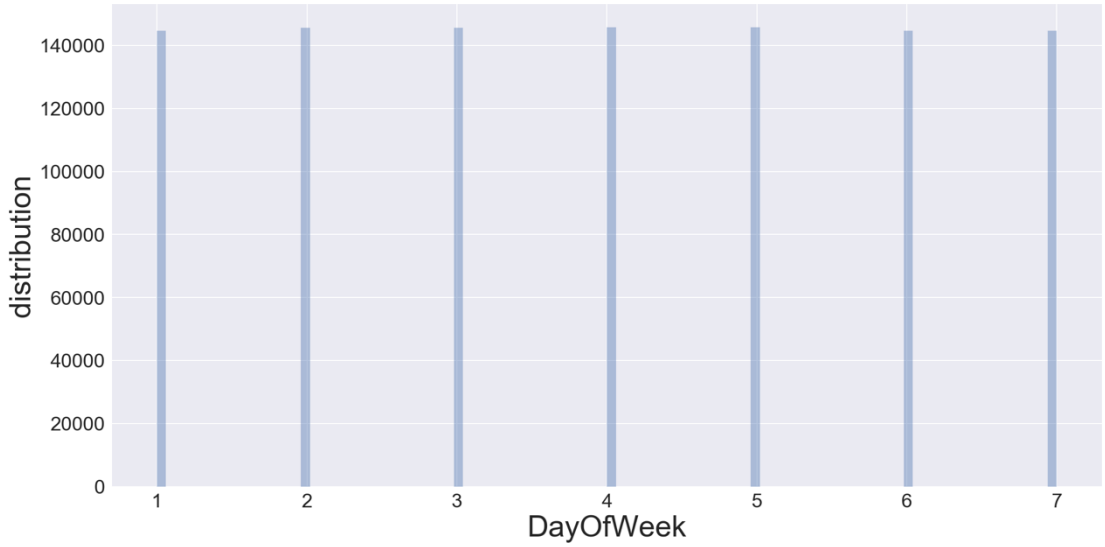

图 3

注意，1-7 分别代表的是周日开始到周六结束的分布情况。

结论：

1、一周内每天的数据都比较接近，不大可能出现因为采样的偏差导致最终对于某些天预测不准确的情况。

2、数据量都超过了 140000 万条，足够我们开展训练了。

2.1.3、Store 分布状况：

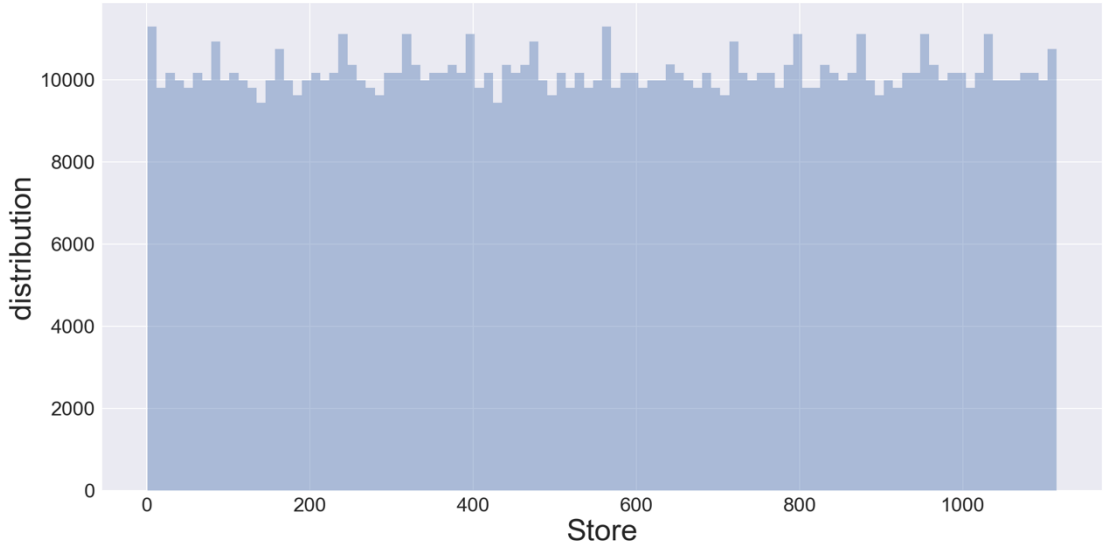

图 4

结论：

对于 1115 家商店来说，采样分布均匀，同理也不会出现某些商店训练数据过少导致的不均匀的情况。

2.1.4、holiday 对于销量的影响

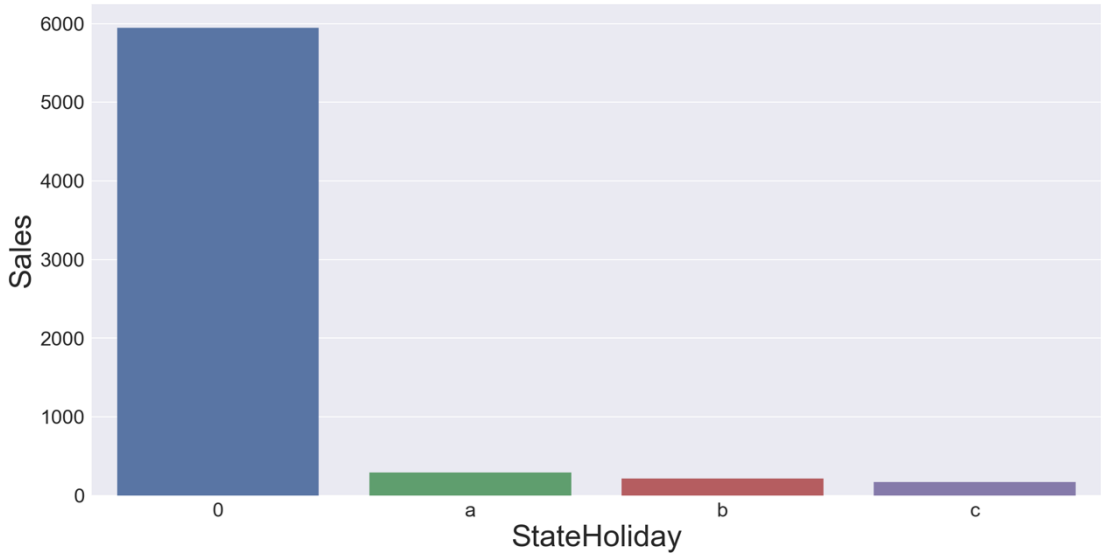

图 5

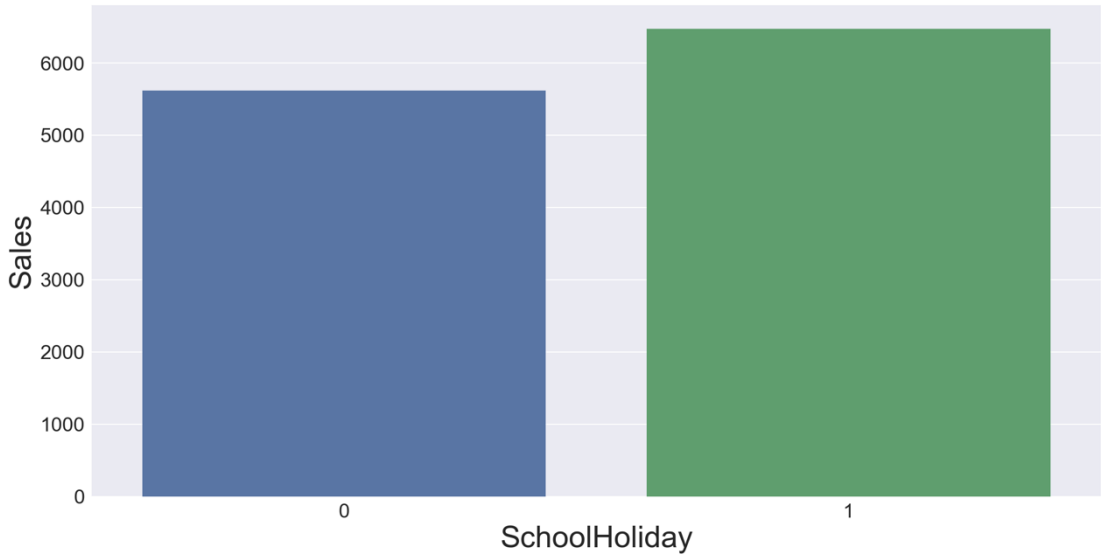

图 6

结论

1、对于法定假日，无论是什么类型的假日，销量都会明显低于非节假日。

2、对于学校假日和非学校假日，前者的销量要比后者高很多，这也印证了之前的判断，即学校放假=》学生需要人陪=》家长带去逛街=》销量增加。

2.1.5、Customer 与 Sales 之间的关系：

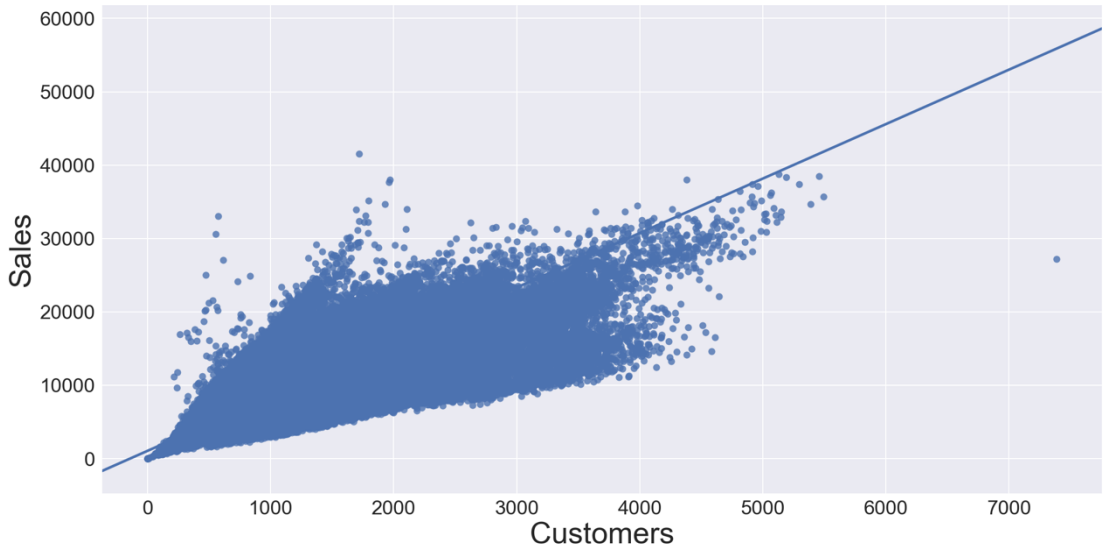

图 7

结论

客户量与销量之间呈现一定的线性相关性，即客户量越大，销量也越大。说明，虽然客户量这个参数在测试数据中不存在（因为你无法预测未来的客户量，否则这就相当于利用了未来函数了），我们还是要重视这个参数，可以将其和其他参数综合之后，作为 store 的参数进行处理。

2.1.6、商店类型与产品类型

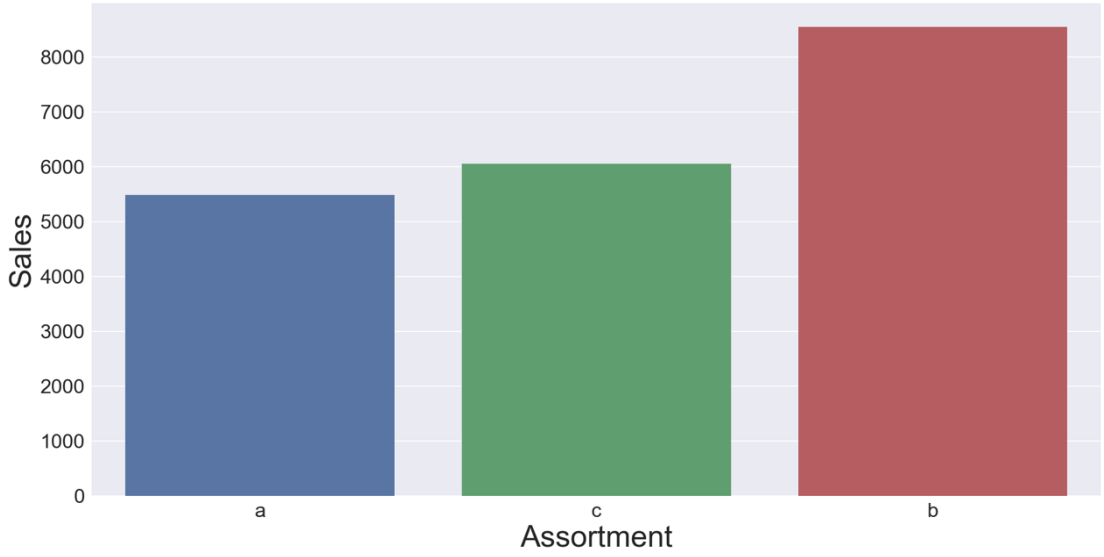

图 8

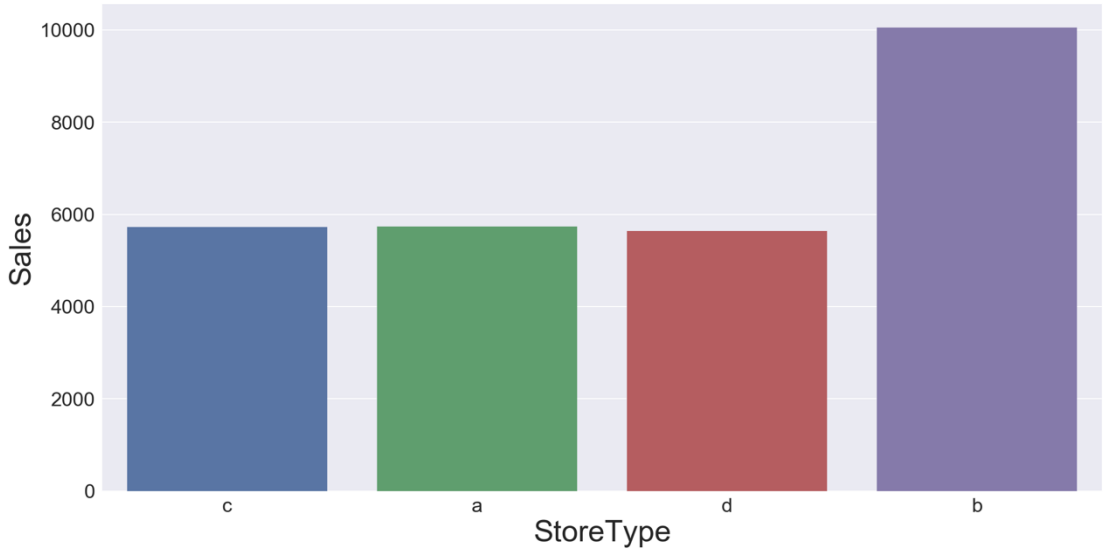

图 9

结论

Assortment：a 类型和 c 类型产品，销售量明显低于 b 类型的产品。

StoreType：a 类型、c 类型和 d 类型的商店，其销售数量明显低于 b 类型的商店。

说明以上两个参数，对于预测结果帮助是很大的。

2.1.7、promo 促销与 promo2 促销对于销量的最终影响

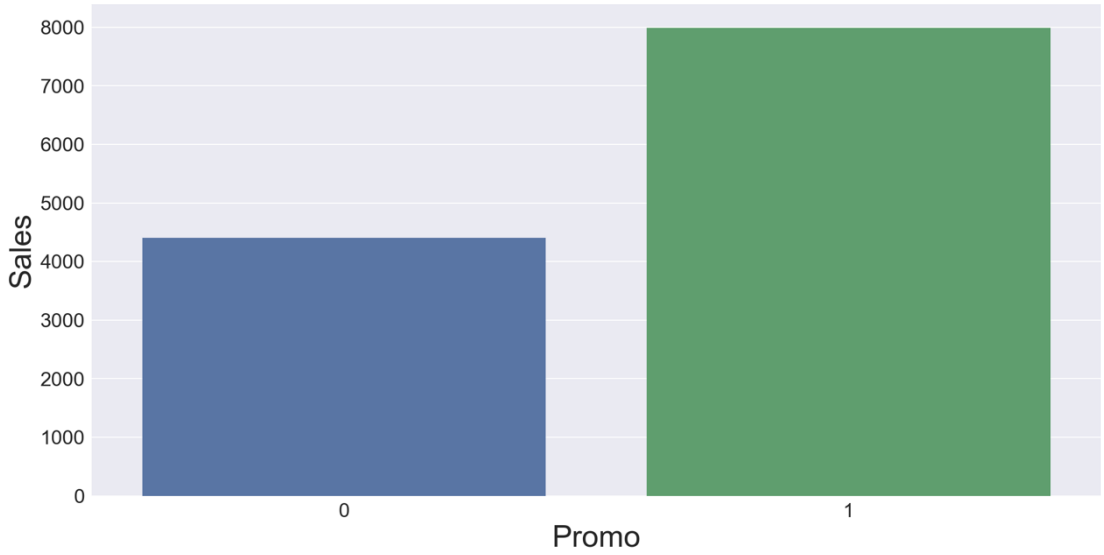

图 10

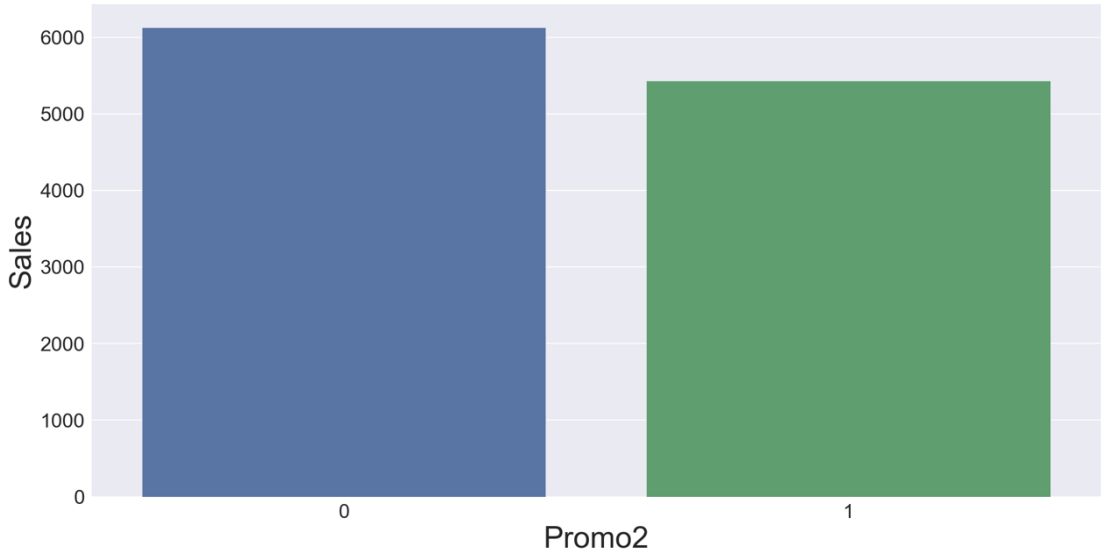

图 11

结论

promo 与 promo2 这两个参数，对于销售量 sales，呈现两种截然相反的关系：

进行 promo 促销的商店当天销量要比没有进行促销的销量多了近一倍；但是进行 promo2 促销的商店，当天销量反而略低于没有进行促销的情况下的销量。

我觉得可能的原因是，进行短期促销，有利于用户从残存的记忆中快速对比出来，究竟价格和平时差了多少；但是长期促销，可能会让用户觉得这是一种很难销售出去的滞销品，进而不再进行购买。

2.3、算法和技术

此次分析需要用到的数据集由 store、train 和 test 三个部分组成，如果想要将数据划分好，首先需要做的就是如何将 store 很好的与 train、test 俩个数据集融合；另外，数据集合中 “customer” 属性在测试集中并不存在，所以需要额外的加工处理才行；数据存在一些缺失的情况，需要考虑到的对缺失数据的处理过程；数据集在训练集上也要切分为测试集与验证集，才能很好的确认训练过程的有效性。

2.3.1、预处理

经过上面的分析可以得出，数据都是来自于 kaggle 的项目。对于 customer 属性，如果丢弃该属性，这个属性的相关信息就不能很好的利用了，如果要将其利用起来，需要将其作为商店的属性进行处理。每家商店由于所处的地理位置的不同、商店面积的大小，药品的种类不同，都会带来人流量、客单价的不同。因此，将 customer 作为商店属性的一部分，能够充分利用属性中包含的信息。
另外，“Date” 属性是一个 object 类型，但其实质上是一个 “YYYY-MM-DD” 的日期类型，所以需要将其处理，拆分成 “year” 、“month” 、“day” 三个字段，进行处理。

2.3.2、数据集拆分

测试集已经被 kaggle 单独区分出来，现在需要处理的就是训练集。由于我们需要判别训练过程的好坏，需要将 train 数据集合再进行切分，需要将其分为训练集和验证集两种类型，为了让训练过程对训练集合的利用最大化，采取了 1%的数据集作为验证集、99%的数据集作为训练集进行处理。这样能够最大化的利用训练数据，同时也能兼顾到对训练过程的监控。

2.3.3、模型算法

由于 kaggle 赛事结束，对第一名的采访资料[3]和他对自己训练参数的总结[4]，我决定采用 xgboost 算法进行训练。xgboost 全称是 “extreme gradient boosting” ，他是基于 GBDT 的加强版本，我认为选择该算法的理由有：

1、xgboost 在代价函里面加入了正则化项，正则项里包含了叶子节点的数目、每个叶子节点输出的 score 的 L2 平方和。能够有效控制模型复杂度，减少过拟合的发生。

2、xgboost 对于缺失值，首先会不考虑缺失数值，将其分别分到左子树和右子树，分别计算损失，然后再选取最优的那一个，如果训练时候没有缺失，预测时候出现了缺失，则默认分到右子树。[5]

3、相比于 GBDT 以 CART 作为基分类器，xgboost 还支持线性分类器，相当于利用了带 L1 和 L2 正则化项的 logistic 回归或线性回归。

4、xgboost 首先会对特征进行排序并保存下来，然后再进行节点分裂时，计算节点特征增益就可以并行进行，节省时间。由于该数据集训练条数达到了 100 万，对于如此大数据量的训练，xgboost 模型能够快速训练得出结果，节省时间。

基于以上讨论，数据经过预处理、数据集拆分，然后再利用 xgboost 算法进行训练，得到最优结果的可能性会大大增加。

2.4、基准模型

由于该分析是基于 kaggle 的项目进行的，对于结果的判断也需要上传到 kaggle 上才能得到最终得分[6]。总共参赛队伍有 3303 支，我给自己定的目标是：

优-前 10%；

良-前 15%。

由于包含两个榜单-Priavte(基于 61%的测试数据)和 Public(基于 31%的测试数据)，因此我选取覆盖度更高的 Private 榜单作为模型的衡量标准。榜单进入前 15% 和 10% 的分数分别为：0.11959 和 0.11773 。这个结果是有参考意义的，因此我决定采用上述的基准来评价模型表现。

---

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

[4] https://www.kaggle.com/c/rossmann-store-sales/discussion/18024

[5] https://blog.csdn.net/jasonzhangoo/article/details/73061060

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

三、方法

3.1、数据预处理

3.1.1、异常值处理

首先我们需要对数据异常值进行处理，在预处理章节和可视化章节看到，数据集合存在着一些异常值，其可能的来源主要是机器原因或者人为原因导致的，它们对于预测过程带来很大影响，数据源头被污染，接下来无论怎么进行调参都无法达到满意的结果。

我们这里采用箱型图判别的方法进行处理，原理如图所示：

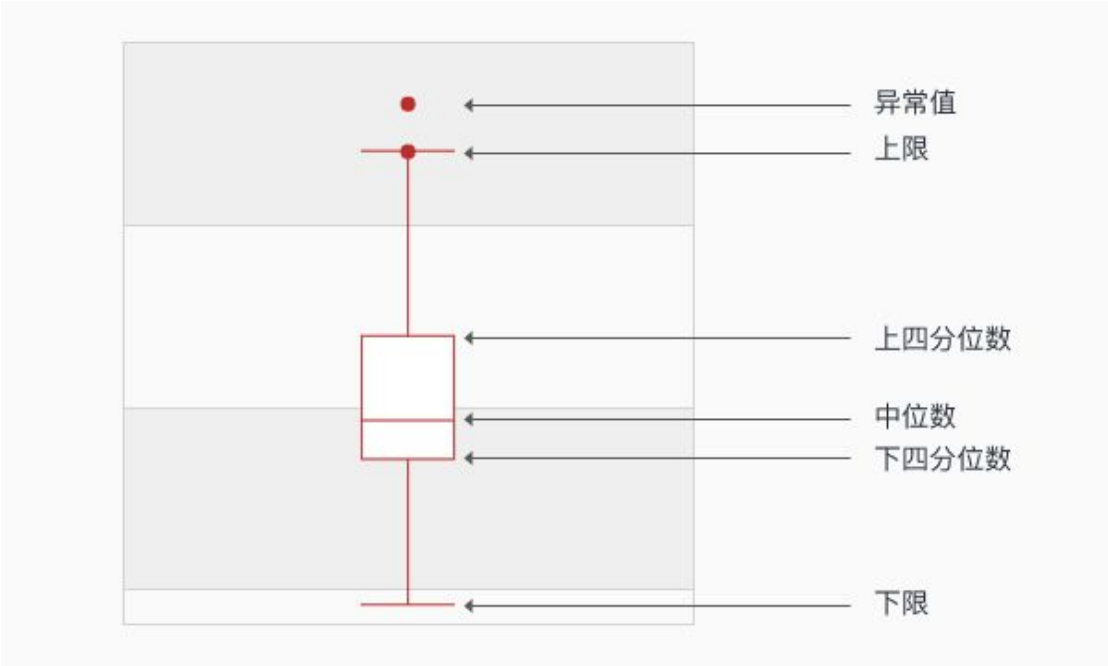

图 12

上图可以表示数据的上限、下限、中位数，上四分位表示中位数和最大值之间的的中位数，下四分位表示中位数和最小值之间的中位数。一般用“Q1”代表上四分位，用“Q3”代表下四分位。用 IQR 表示 Q1 与 Q3 的差值，异常值被定义为大于（Q1+1.5*IQR）和小于（Q3-1.5*IQR）。

通过下图可以看到，在我对异常值进行去除之后，并没有影响到整体的分布，并且去除之后，长尾的那部分数据被去掉，整个数据看起来更加平滑，也更加贴近于正态分布。

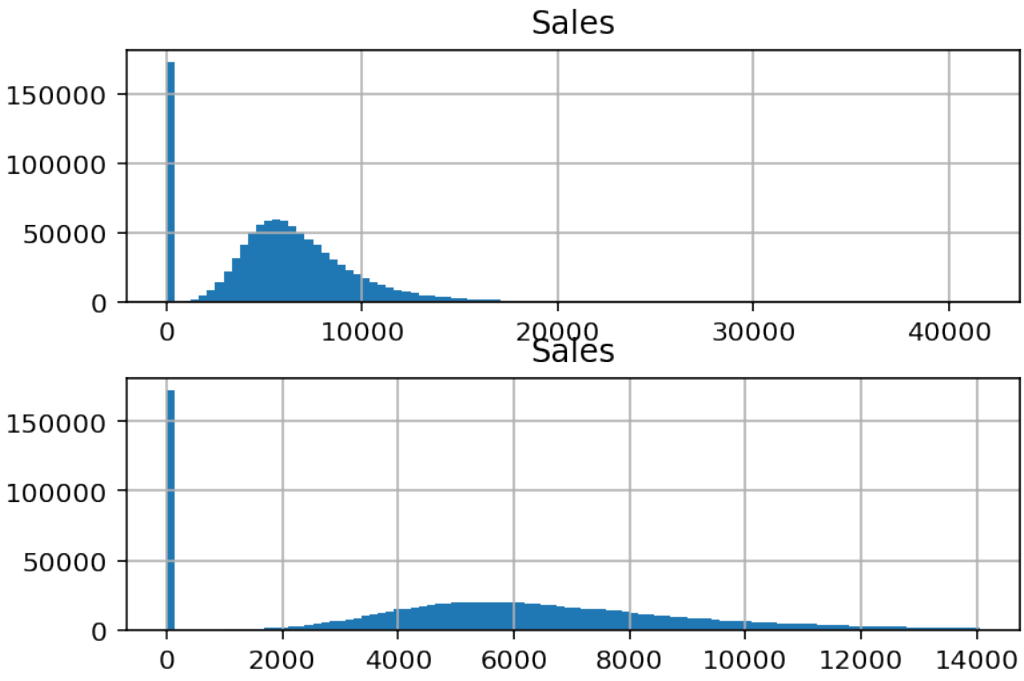

图 13

3.1.2、OneHot 编码

由于机器学习方法，对于数据类型的限制，需要将类比的数据进行 Onehot 编码才能输入到模型中去进行计算。需要将 “StoreType” 、 “Assortment” 、 “Stateholiday” 其中的 “abcd” 对应到多个字段中去。对于 “ProniInterval” 字段，需要将 12 个月分别对应到 12 个字段中去，得出算法可以处理的数据类型。

3.1.3、时间处理

由于销售数据是带时间序列的，其 “Date” 字段代表当天的时间，由于 “Date” 字段实际为 “YYYY-MM-DD” 格式，需要将其转化为 “Date_year” 、 “Date_month” 、 “Date_day” 。观察 Store 为 1 的商店其走势情况可以看出，时间周期对其影响还是很大的，年末销量会有大曾，季度与季度之间的销量也有很大区别，因此需增加字段来表示当天在一年中处于什么阶段，于是新增两个字段 “DateOfYear” 、 “WeekOfYear” 。

3.1.4、特征组合

考虑到当前属性虽然能够进行预测，但是和排行榜的衡量标准来比较，发现差距还是很大，因此需要新增数据。

首先，是将 Customer 字段与 Sales 字段等相结合，处理成 “客单价” 、 “客流量”

等商店属性。

其次，根据对排行榜第一名的 Gret 的采访，他与属性的处理很有借鉴意义，于是将数据分为"有/无促销情况下的销量"、“2013/2014/2015 不同年份的销量平均值”、“第一到四季度销量的平均值”等特征属性。

第三，由于对竞争对手开业时间分布在“CompetitionOpenSince[Year/Month]”两个字段，如果不进行处理就很难分析，因此将其处理成，截止这条记录发生时，竞争对手开业的总月数就可以方便分析了。同理也可以处理“Promo2Since[Year/Month]”字段。

第四、节假日前后，会进行一些打折活动，因此对于商家的销量有一定的影响。由于之前的分析看出，学校假对于销量影响并不是很大，而且学校假期普遍较长，因此这里只对“a 类”法定假日进行分析，得出距离各个“a 类”法定假日的距离。

第五、最近一周的 Holiday 数量也会对销售量有一定影响，因此算出最近一周时间内，“学校假”和“法定假”各自的数量。

3.1.5、缺失值处理

对于缺失值，处理方式有很多，直接删除是一种方法，均值插入也是一种方式。这里我的处理方式是：不作处理。因为 XGBoost 有一套对于缺失值的处理方式：
训练集中的缺失值处理方法：分别分配到左右子树，选取损失函数最优的解决方案；测试集中的缺失值处理方法：默认分配到右子树。因此，我们可以放 XGBoost 去处理缺失的数据。

3.2、执行过程

选取的 XGBoost 的 Python 版本作为算法库，本章节会基于代码，详细描述整个分析过程，包括数据获取，预处理，特征工程，训练，预测等过程。

3.2.1、首先是获取数据：

```
1.  # 1、获取数据
2.  data_store = pd.read_csv('../data/store.csv',low_memory=False)
3.  data_test = pd.read_csv('../data/test.csv',low_memory=False)
4.  data_train = pd.read_csv('../data/train.csv',low_memory=False)
```

3.2.2、处理 Customer 属性

```
1.  # 增加客单价、Store 分类、人流量 等商店参数
```

```
data_train_count = data_train
data_train_count['count'] = 1
data_train_count = data_train_count.groupby("Store").sum()
data_train_count["salesPerCustomer"] = data_train_count["Sales"]/data_train_count["Customers"]
data_train_count["salesPerCount"] = data_train_count["Sales"]/data_train_count["count"]
data_train_count["customerPerCount"] = data_train_count["Customers"]/data_train_count["count"]
data_train_count = data_train_count.reset_index()
```

对于 customer 字段的处理之前有提到过，将其处理为商店的属性。

3.2.3、合并表格

```
# 将两个表合为一个
data_feature_all = data_test.append(data_train)
data_test["from"]=1
data_train["from"]=0
data_customer_entire=data_train.append(data_test)
# 将商店数据 merge 进去
data_train_count_real = data_train_count.loc[:,["Store","salesPerCustomer","salesPerCount","customerPerCount"]]
data_store_all = pd.merge(data_store,data_train_count_real,on="Store")
data_feature_all = pd.merge(data_customer_entire,data_store_all,on="Store")
```

合并表格是为了方便进行统一的属性变换等操作。

3.2.4、观察缺失值

```
# 缺失值：
print("字段描述：")
print(data_feature_all.describe())
print("\r\n 值为 nan 的比例：")
print(data_feature_all.isnull().sum()/len(data_feature_all))
```

“Promo2SinceWeek”、“Promo2SinceYear”、”PromoInterval“、“CompetitionOpenSinceMonth”、“CompetitionOpenSinceYear”这五个字段为空的比例比较大。对于某些模型例如：knn、svm 来说，缺失值影像比较大；但是对于我将要使用的 xgboost 来说，它在遇到有缺失值的属性时，会把缺失值分别放到左叶子节点和右叶子节点，然后再计算增益，哪个增益大就放到哪个叶子节点上。所以暂时不处理

他们。

测试数据中，没有“customer”字段，所以进行训练的时候，不能使用该字段作为训练集属性，但是为了能够充分利用所有字段，我预计会将其进行加工处理，比如将其处理成商店的属性，如：“客单价”、“日客流量”等属性。

3.2.5、其他字段处理

```
def insertOneHot(x,strCol,arrValues):
    for val in arrValues:
        newCol = strCol+"_"+val
        x[newCol]= 0
        x.loc[x[strCol]==val,newCol] = 1
    return x

# 处理 【Date】 字段
data_feature_all["Date"] = pd.to_datetime(data_feature_all["Date"], format="%Y-%m-%d", errors='ignore')
data_feature_all["date_year"] = data_feature_all["Date"].dt.year
data_feature_all["date_month"] = data_feature_all["Date"].dt.month
data_feature_all["date_day"] = data_feature_all["Date"].dt.day
data_feature_all['DayOfYear'] = data_feature_all["Date"].dt.dayofyear
data_feature_all['WeekOfYear'] = data_feature_all["Date"].dt.weekofyear

# 处理 【StoreType】 字段
StoreTypeKeys = ["a","b","c","d"]
insertOneHot(data_feature_all,"StoreType",StoreTypeKeys)

# 处理 【Assortment】 字段
AssortmentKeys = ["a","b","c"]
insertOneHot(data_feature_all,"Assortment",AssortmentKeys)

# 处理 【Promotnterval】 字段
PromoIntervalKeys = ["Jan","Feb","Mar","Apr","May","Jun","Jul","Aug","Sept","Oct","Nov","Dec",]
for keyVal in PromoIntervalKeys:
    keyValEntire = "PromoInterval"+"_"+keyVal
    data_feature_all[keyValEntire] = data_feature_all["PromoInterval"].apply(lambda x: 1 if str(x).find(keyVal)>=0 else 0)

# 处理 【Sales】 字段
fig1, axes = plt.subplots(nrows=2, ncols=1)
from_sales = data_feature_all[data_feature_all["from"]==0]["Sales"]
pd.DataFrame(from_sales).hist(bins=100,ax=axes[0])
```

```
sales_Q1 = np.percentile(from_sales,25)
sales_Q3 = np.percentile(from_sales,75)
sales_QR = sales_Q3 - sales_Q1
data_feature_all.drop(data_feature_all[data_feature_all["from"]==0][data_feature_all["Sales"]>(sales_Q3 + 1.5*sales_QR)].index,axis=0,inplace=True)
data_feature_all.drop(data_feature_all[data_feature_all["from"]==0][data_feature_all["Sales"]<(sales_Q1 - 1.5*sales_QR)].index,axis=0,inplace=True)
pd.DataFrame(data_feature_all["Sales"]).hist(bins=100,ax=axes[1])

# 处理 【StateHoliday】 字段
StateHolidayKeys = ["a","b","c","0"]
insertOneHot(data_feature_all,"StateHoliday",StateHolidayKeys)

# 处理 【Open】 字段
data_feature_all = data_feature_all.fillna({"Open":1})
```

3.2.6、新增属性

```
#新增平均值数据
data_feature_all_train = data_feature_all[data_feature_all["from"]==0]
data_feature_all["store_sales_mean_promo"] = 0
data_feature_all["store_sales_mean_promo_no"] = 0

data_feature_all["store_sales_mean_2013"] = 0
data_feature_all["store_sales_mean_2014"] = 0
data_feature_all["store_sales_mean_2015"] = 0

data_feature_all["store_sales_mean_qur1"] = 0
data_feature_all["store_sales_mean_qur2"] = 0
data_feature_all["store_sales_mean_qur3"] = 0
data_feature_all["store_sales_mean_qur4"] = 0

for storeItem in data_train_count["Store"]:
    data_feature_all_train_store = data_feature_all_train.loc[data_feature_all_train["Store"] == storeItem,:]
    data_feature_all.loc[data_feature_all["Store"] == storeItem,'store_sales_mean_promo'] = \
        data_feature_all_train_store[data_feature_all_train_store["Promo"]==1]["Sales"].mean()
    data_feature_all.loc[data_feature_all["Store"] == storeItem,'store_sales_mean_promo_no'] = \
```

```
        data_feature_all_train_store[data_feature_all_train_store["Promo"]==0]["Sales"].mean()

    data_feature_all.loc[data_feature_all["Store"] == storeItem,'store_sales_mean_2013'] = \
        data_feature_all_train_store[data_feature_all_train_store["date_year"]==2013]["Sales"].mean()
    data_feature_all.loc[data_feature_all["Store"] == storeItem,'store_sales_mean_2014'] = \
        data_feature_all_train_store[data_feature_all_train_store["date_year"]==2014]["Sales"].mean()
    data_feature_all.loc[data_feature_all["Store"] == storeItem,'store_sales_mean_2015'] = \
        data_feature_all_train_store[data_feature_all_train_store["date_year"]==2015]["Sales"].mean()

    data_feature_all.loc[data_feature_all["Store"] == storeItem,'store_sales_mean_qur1'] = \
        data_feature_all_train_store[(data_feature_all_train_store["date_month"]>=0) & (data_feature_all_train_store["date_month"]<=2)]["Sales"].mean()
    data_feature_all.loc[data_feature_all["Store"] == storeItem,'store_sales_mean_qur2'] = \
        data_feature_all_train_store[(data_feature_all_train_store["date_month"]>=3) & (data_feature_all_train_store["date_month"]<=5)]["Sales"].mean()
    data_feature_all.loc[data_feature_all["Store"] == storeItem,'store_sales_mean_qur3'] = \
        data_feature_all_train_store[(data_feature_all_train_store["date_month"]>=6) & (data_feature_all_train_store["date_month"]<=8)]["Sales"].mean()
    data_feature_all.loc[data_feature_all["Store"] == storeItem,'store_sales_mean_qur4'] = \
        data_feature_all_train_store[(data_feature_all_train_store["date_month"]>=9) & (data_feature_all_train_store["date_month"]<=11)]["Sales"].mean()

# 竞争对手开业月数
data_feature_all["CompetitionForMonths"] = (data_feature_all["date_year"] - data_feature_all["CompetitionOpenSinceYear"])*12 \
    +(data_feature_all["date_month"]-data_feature_all["CompetitionOpenSinceMonth"])

# promo 持续天数
data_feature_all["Promo2LastDays"] = (data_feature_all["date_year"]-
data_feature_all["Promo2SinceYear"])*365 \
    +(data_feature_all["DayOfYear"]-7*(data_feature_all["Promo2SinceWeek"]))

#距离 holiday 日期
```

```
holidayofyear = data_feature_all[data_feature_all["StateHoliday"]=="a"]
sortedholidays = sorted(holidayofyear["DayOfYear"].reset_index(name="DayOfHoliday")["DayOfHoliday"].unique())
for holiday in sortedholidays:
    data_feature_all["DayOfHoliday_"+str(holiday)] = holiday - data_feature_all["DayOfYear"]

#新增最近 7 天 holiday 数量
for storeItem in data_train_count["Store"]:
    data_feature_store = data_feature_all[data_feature_all["Store"]==storeItem]

    data_feature_store["index"] = data_feature_store.index
    data_feature_store.set_index("Date", inplace=True)
    data_feature_store.sort_index(inplace=True)

    #最近 7 天假日数
    data_feature_store["StateHoliday"] = data_feature_store["StateHoliday"].apply(lambda x: 0 if x=="0" else 1)
    data_feature_store["StateHolidayLast"] = data_feature_store["StateHoliday"].rolling(7).sum()
    data_feature_store["SchoolHolidayLast"] = data_feature_store["SchoolHoliday"].rolling(7).sum()
    data_feature_store.set_index("index", inplace=True)

    data_feature_all.loc[data_feature_all["Store"]==storeItem,'StateHolidayLast'] = \
        data_feature_store["StateHolidayLast"]
    data_feature_all.loc[data_feature_all["Store"]==storeItem,'SchoolHolidayLast'] = \
        data_feature_store["SchoolHolidayLast"]
```

从逻辑推理上看，新增字段对于预测结果应该是有帮助的。并且这也从最后的模型分析中可以看出其重要性。

3.2.7、删除多余属性

```
# 去掉 customer、Store、count、Date、StoreType、Assortment、PromoIntervalPromoInterval 列
data_feature_all.drop("Customers",axis=1,inplace=True)
data_feature_all.drop("count",axis=1,inplace=True)
data_feature_all.drop("StoreType",axis=1,inplace=True)
data_feature_all.drop("Assortment",axis=1,inplace=True)
```

```
data_feature_all.drop("PromoInterval",axis=1,inplace=True)
data_feature_all.drop("StateHoliday",axis=1,inplace=True)
data_feature_all.drop("Date",axis=1,inplace=True)
```

删除的数据主要是为了去掉之前已经处理过的字段，这些字段的信息已经被提取到其他字段中去了，比如 Customers 字段、Date 字段等。

3.2.8、训练

```
# 切分【测试数据】与【训练数据】
data_x_test = data_feature_all[data_feature_all["from"]==1]
data_x_test = data_x_test.sort_values(by=['Id'],axis=0,ascending=True)
data_x_test = data_x_test.reset_index()
data_x_test.drop("Id",axis=1,inplace=True)
data_x_test.drop("index",axis=1,inplace=True)
data_x_test.drop("Sales",axis=1,inplace=True)
# # 处理空 Open
# data_x_test.loc[data_x_test["Open"].isnull(),"Open"] = 1

# 训练数据
data_all_Train = data_feature_all[data_feature_all["from"]==0]
data_all_Train.drop("Id",axis=1,inplace=True)
data_x_Train = data_all_Train.drop("Sales",axis=1)

# 利用 xgboost 模型训练数据
import xgboost as xgb
import operator
from sklearn import cross_validation

def create_feature_map(features):
    outfile = open('xgb.fmap', 'w')
    for i, feat in enumerate(features):
        outfile.write('{0}\t{1}\tq\n'.format(i, feat))
    outfile.close()

def ToWeight(y):
    w = np.zeros(y.shape, dtype=float)
    ind = y != 0
    w[ind] = 1./(y[ind]**2)
    return w

```

```
def rmspe(yhat, y):
    y = y.get_label()
    w = ToWeight(y)
    rmspe = np.sqrt(np.mean( w * (y - yhat)**2 ))
    return "rmspe",rmspe

params = {"objective": "reg:linear",
          "booster" : "gbtree",
          "eta": 0.01,
          "max_bin":500,
          "max_depth": 9,
          "subsample": 0.9,
          "colsample_bytree": 0.7,
          "silent": 1,
          "seed":1200,
          "min_child_weight" : 52 ,
          "alpha":2,
          "gamma":2
          }
num_trees = 1200

data_all_Train, data_all_Valid = cross_validation.train_test_split(data_all_Train, test_size=0.01,random_state=40)
data_y_Train = data_all_Train["Sales"]
data_y_Valid = data_all_Valid["Sales"]
data_all_Train.drop("Sales",axis=1,inplace=True)
data_all_Valid.drop("Sales",axis=1,inplace=True)
data_x_Valid = data_all_Valid
data_x_Train = data_all_Train
dtrain = xgb.DMatrix(data_x_Train, data_y_Train)
dvalid = xgb.DMatrix(data_x_Valid, data_y_Valid)
dtest = xgb.DMatrix(data_x_test)
watchlist = [(dvalid, 'eval'), (dtrain, 'train')]
gbm = xgb.train(params, dtrain, num_trees,evals=watchlist, early_stopping_rounds=50,feval=rmspe, verbose_eval=True)
data_y_test = gbm.predict(dtest)
```

对于 XGBoost 的参数[7]，主要有如下几种：

1、General Parameters 控制总体功能的参数

booster [default=gbtree]:

gbtree：树模型

gblinear：线性模型

[7] https://zhuanlan.zhihu.com/p/28672955

silent [default=0]
是否打印信息，1-表示打印详细信息；0-表示不打印信息。

nthread [default to maximum number of threads available if not set]
希望在机器多少个核上并发处理，默认是使用最大数量的核进行处理。

2、Booster Parameters 控制单个学习器的属性
eta [default=0.3]
代表算法的学习率
通过在每一步中缩小权重来让模型更加鲁棒
一般数值使用：0.01-0.2

min_child_weight [default=1]
表示叶子节点中，如果一个叶子节点的样本权重和小于该值，则拆分过程结束。调大这个参数可以控制过拟合。

max_depth [default=6]
是用来设置树的最大深度，深度过大会导致过拟合，一般使用 3-10

subsample [default=1]
算法需要原始数据进行随机采样来构建单个树，这个参数表示随机采样的百分比。
较小的值可能导致欠拟合，较大的值会导致过拟合的发生。
一般取值 0.5-1

colsample_bytree [default=1]
表示创建树的时候，从所有列中选取的比例。

alpha [default=0]
L1 正则惩罚系数，当数据位维度极高的时候可以使用，是的算法运行速度快。

gamma [default=0]
该参数控制节点的划分的最小损失函数。一般是划分到该节点的损失函数为 0 的时候才停止，该参数定义了最低的损失函数值，低于该值即不进行划分。

seed [default=0]
为了能够产生可重现结果，需要设置该值。

3、Learning Task Parameters 控制调优的步骤
objective [default=reg:linear]
定义需要被最小化的损失函数，常用的值有：
“reg:linear” –线性回归。
“reg:logistic” –逻辑回归。
“binary:logistic” –二分类的逻辑回归问题，输出为概率。
“multi:softmax” –让 XGBoost 采用 softmax 目标函数处理多分类问题，同时需要设置

参数 num_class（类别个数）

3.2.9、结果输出

```
# 输出
f = open("submission_xgb_nodayin_9.csv", "w")
print("\"Id\",\"Sales\"", file = f)
for i in range(len(data_y_test)):
    print("%d,%0.2f"%(i+1, data_y_test[i]), file = f)
f.close()
```

3.2.10、重要性分析

```
# 绘制重要性曲线
create_feature_map(data_x_Valid.columns)
importance = gbm.get_fscore(fmap='xgb.fmap')
importance = sorted(importance.items(), key=operator.itemgetter(1))
df = pd.DataFrame(importance, columns=['feature', 'fscore'])
df['fscore'] = df['fscore'] / df['fscore'].sum()
featp = df.plot(kind='barh', x='feature', y='fscore', legend=False, figsize=(6, 10))
plt.title('XGBoost Feature Importance')
plt.xlabel('relative importance')
fig_featp = featp.get_figure()
fig_featp.savefig('feature_importance_xgb_9.png', bbox_inches='tight', pad_inches=1)
```

为了分析之前的判断是否正确，找出究竟哪些属性对于预测结果有重要意义，借鉴了 cast42 的方法[8]，打印出重要性如图所示：

[8] https://www.kaggle.com/cast42/xgboost-in-python-with-rmspe-v2

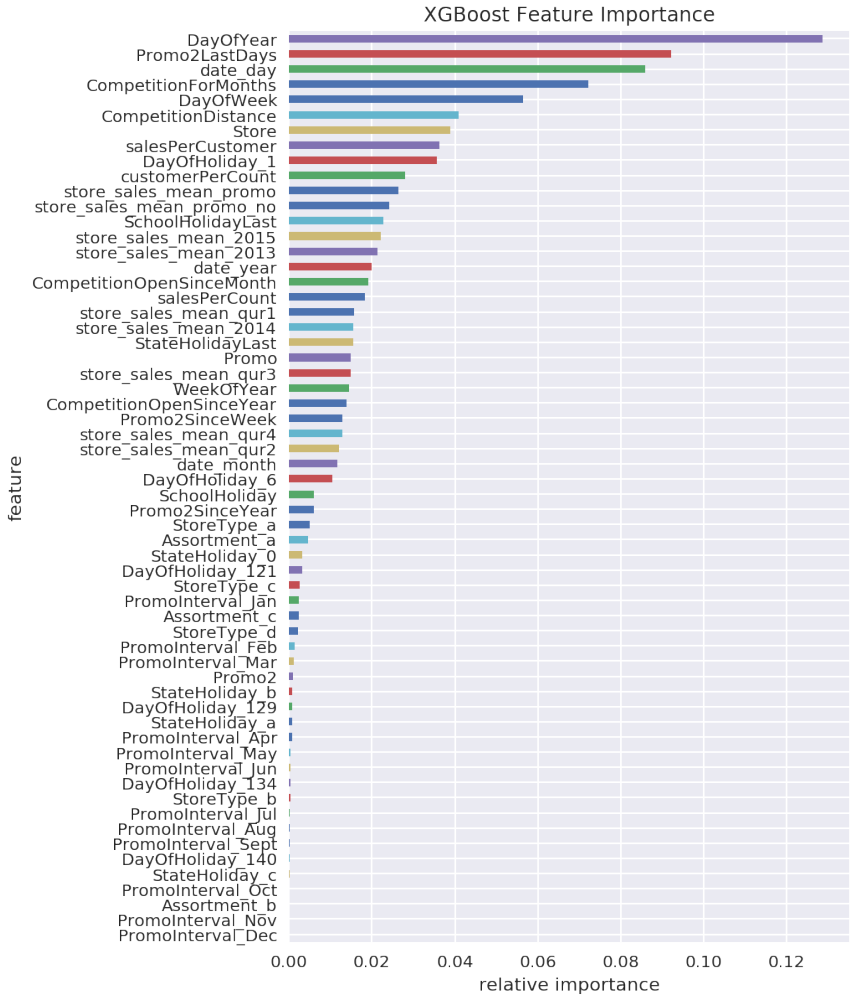

图 14

可以看出：

1、重要性排名前 5 的字段有：

“DayOfYear”、“Promo2LastDays”、“date_day”、“CompetitionForMonths”、“DayOfWeek”

前四个字段是都经特征工程处理过的，可以看出，特征工程对于预测结果有很大影响的。

2、之前判断，销量有很大的周期性，从图上可以看出，“DayOfYear”、“DayOfWeek”、“date_day”等表示时间的字段占据排行榜的前几名，也是验证了之前的结论。

3、“DayOfHoliday_1”、“SchoolHolidayLast”、“StateHolidayLast”等字段，排名比较靠前，说明假期前后，对于销量也是有一定影响的，验证了之前的结论。

4、从“store_sales_mean_[2013\2014\2015]”字段排名来看，各个年份的销量平均值对于销量也是有一定影响的。

3.3、完善

对于真实的训练数据和测试数据，存在很多缺失值和异常值的情况。如果不进行特征工程的处理过程，肯定很难达到预期要求的分数。在进行了异常值去除、特征 One-Hot 编码之后，还需要组装一些自定义的属性，才能让分数得到有效提升。算法方面，

先是利用默认的参数进行训练，结果不是很理想，后来深入了解了 XGBoost 算法之后，研读了相关资料[9]，对于参数都有了大致的了解，通过参数的调整，使得分数有效提升。

特征工程

特征工程对于整个机器学习过程是重中之重，因为数据决定了整个分析的天花板，算法只是去逼近这个天花板。一开始使用默认的属性，发现属性存在很多的缺陷，首先就是离群值，对于预测结果影响很大，将其去掉之后，训练效果就有了提升。由于我选取的是 XGBoost 算法，对于缺失值有自己的处理方式，所以决定不对其进行处理。但是训练还是很难达到要求，于是我就加上了一些派生的属性，比如最近 7 天内的 Holiday 数量、每年销售量的平均值等属性，使得算法有所提升。

算法

一开始我尝试着使用线性回归的方式进行预测，发现效果并不是很好，很难去拟合到理想的得分。在查看了对 Gert 的采访之后，考虑到特征属性的数量、训练记录的数量等因素，决定采用的 XGBoost 算法[10]。一开始使用算法的默认参数进行训练，结果并不理想。于是想到去了解每个参数的实际意义，在此基础上对参数进行了一些调整，例如树的深度字段，分别尝试了 “max_depth=3\6\9\12” 这四种深度，得出的结果可以看出，当深度选择为 “max_depth=12” 作为训练参数。经过对类似于上述参数的调整过程之后，结合对不同参数对训练结果的影响，选出了较优的一组进行训练，并用得出的模型对测试数据进行预测。

四、结果

4.1、模型的评价与验证

模型验证的过程需要在 kaggle 上进行，首先要对 test 数据集进行预测，把输出的文件上传到 kaggle 上，由系统来打分。在 kaggle 上，有两种排行榜 :Private 和 Public，分别计算了 31% 和 61% 的数据集，因此我选择计算更多数据的 Private 榜单来作为对预测结果的度量标准。

整个分析过程中，我提交上传了大概 63 次之多，均方根误差百分比由一开始的 0.32005 下降到最后的 0.11756。

在调参的整个过程中，举例来说，比如对于 max_depth 的调节过程。固定了其他参数，以判断 max_depth 的不同取值对于运行结果的影响，我分别取了 max_depth 为 [3\6\9\12]来进行训练，取得了如图所示的结论：

[9] https://blog.csdn.net/han_xiaoyang/article/details/52665396

[10] https://www.cnblogs.com/mfryf/p/6293814.html

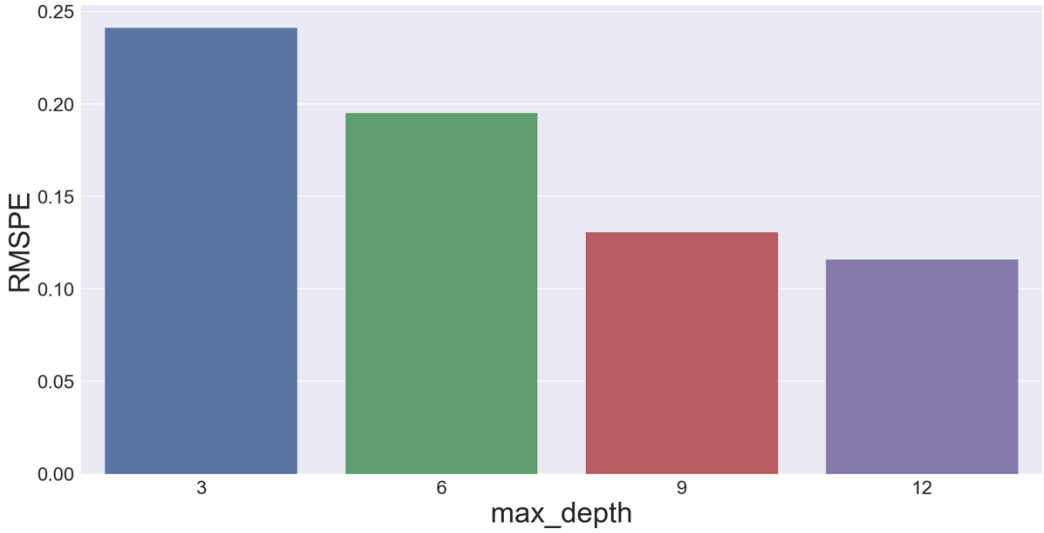

图 15

从图 15 上可以看出，随着 max_depth 的提高，训练模型对于测试数据的预测 RMSPE 值也在不断降低，当 max_depth 为 12 时达到最佳值。

4.2、合理性分析

通过了对参数的调整，最终得到了一组比较满意的参数，其计算结果的 Private Score 值为 **0.11756** ，Private 榜单的排名为 **315**，符合 **10%**的要求。

经过对第一名 Gert 的采访可以发现，不仅仅是使用了出题者给出的数据，还利用了一些额外的数据，例如天气的影响。天气对于人们是否出门购物，也有一定的影响，因此模型还可以增加一些额外的参数来增加预测的准确度。

五、项目结论

5.1、结果可视化

由于预测参数较多，为了了解各个参数对于预测结果的影响，绘制了各属性对于结果的重要性排序图：

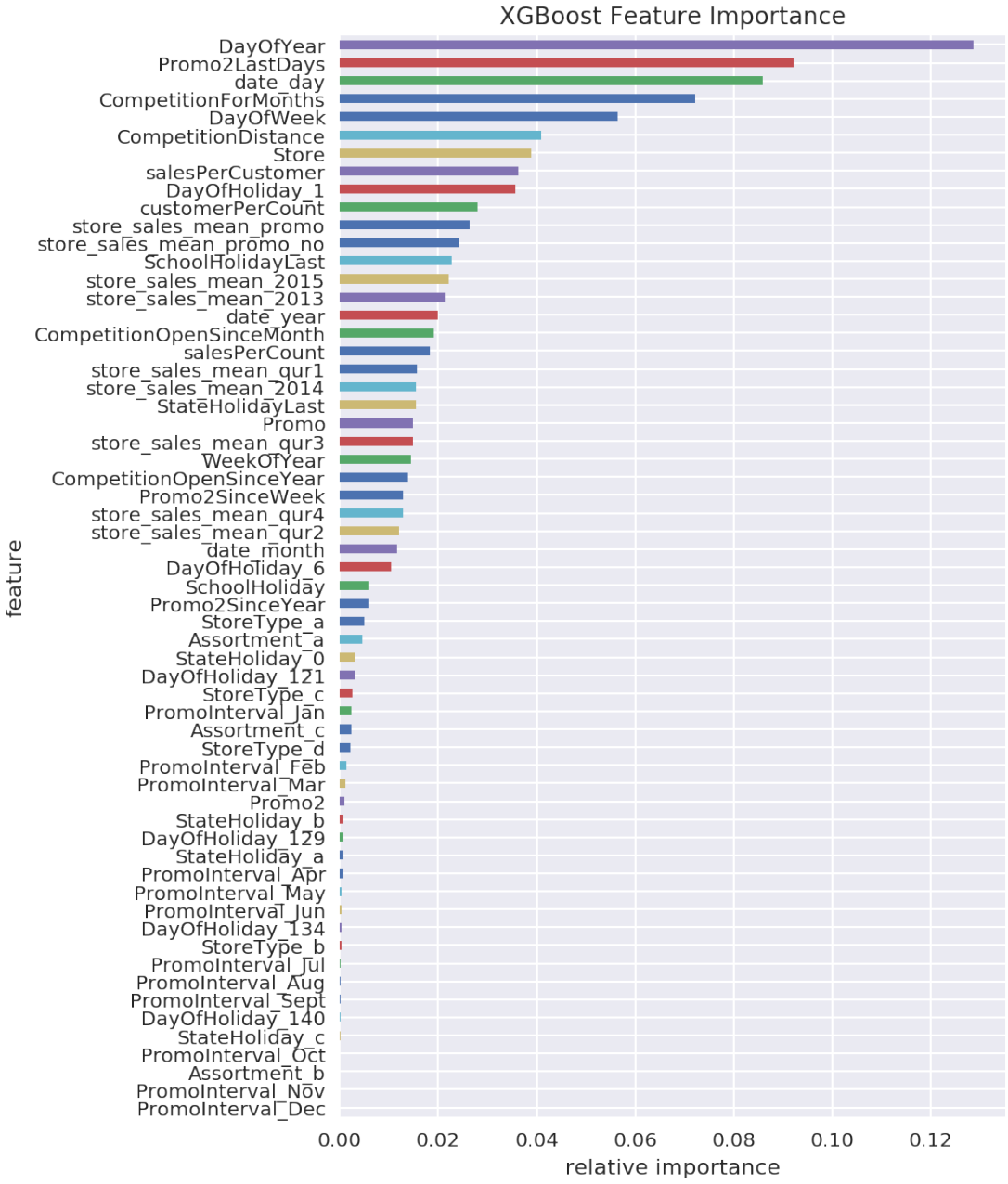

图 16

该重要性排序图是在 max_depth=12 的情况下绘制出来的。有几个出乎意料之外的地方：首先是 “Promo2LastDays” 属性排名如此之高，被排在了第二名的名词，但是其原始属性 Promo2 的重要性排名却很低，几乎可以忽略不计了，说明当天是否开展长期促销对于销售量并没有很大帮助，反而是开展长期促销的持续天数对于销量有一定影响；从 date_day 和 DayOfMonth 排名如此之高，也可以看出周期性的影响对于销售数据来说有很大的作用。

5.2、对项目的思考

由于对商业数据的处理十分感兴趣，并且该项目的数据来源更贴近于真实的商业数据，因此选择了此项目。经过本次对于 Rossmann 数据的处理过程可以看出，数据预处

理环节对于起到了决定性的作用，不仅仅需要对原始数据进行处理，而且仅仅使用原始数据是远远不够的，因此还需要对数据进行一些加工，才能有可能在接下来的训练过程的出比较满意的结果。

在模型选择与调参的过程中，不能仅仅依赖与模型的默认属性，还需要在对于各个参数有充分了解的基础上，知道参数之间的关系，并且需要保证每次调参过程都只能够修改一个参数，这样才能有足够的参照，对于预测结果才能有很好的评估。

5.3、需要做出的改进

很明显，达到基本要求还是远远不够的，还有如下几个方面有待优化：

1、数据处理：

数据处理方面，对于给定的数据还是有更多可以挖掘的地方，比如可以加入更多时间序列的属性，来帮助拟合销售量的波动。另外，不仅仅停留在处理给出的数据上，也可以从外部获取一些数据，比如天气数据等，通过加入更多的信息来源，才能在预测过程中提升数据带来的天花板。

2、模型选择：

可以增加一些模型来帮助预测结果。例如通过增加 ARMA 时间序列模型，即自回归移动平均混合模型，该模型对于周期性波动的数据有较好处理能力。

3、参数选择：

对于参数的调整，由于 XGBoost 的参数众多，有大量的排列组合情况。因此我认为使用 Grid Search 的方式寻找最优组合，会带来巨大的运算，是很费时间的，还是要首先基于对于各个参数内涵的了解，最好是能够深入到 XGBoost 的代码实现原因中去，才能跟快的找出满意的参数组合。

参考文献：

[1] 钱晓星. 新田公司摩托车销售预测. 中南大学. 2002.

[2] 何鹏. 销售预测模型在世纪达公司的应用研究. 中南大学. 2006.

[3] 叶倩怡. 基于 Xgboost 的商业销售预测. 南昌大学. 2017.

[4] 唐新春. 基于机器学习方法对销售预测的研究. InfoQ. 2017.