Rossmann 门店销售预测项目报告

I.问题的定义

项目概述

Rossmann 是一家药妆连锁店，在欧洲的 7 个国家拥有超过 3000 家的门店。近期，Rossmann 的经理们拿到了一项任务，要求他们把日常销售的预测提前到六个星期。药妆店的销售额受到很多因素的影响，包括促销，竞争，学校和国家公共假日，季节性变化还有位置。数千的私人经理基于他们所在的特殊的环境做了销售额的预测，这些预测结果的准确性也大不相同。可靠的预测可以帮助门店经理们安排有效的员工日程来提高整个门店的生产力和动力。Rossmann 希望有一个稳健的模型来预测位于德国的 1115 家门店未来六周的销售额。

这个项目涉及了计算机和商业两个领域领域，要求我们利用计算机科学中的机器学习方法，来解决商业上的一个问题。同时也让我们看到了机器学习助力商业发展的可能性。

问题陈述

预测 Rossmann 未来的销售额在本质上属于机器学习中的**监督学习**，更具体一点，是有监督的序列预测任务。我们的目标是根据已有的历史数据，提取合适的特征构造有效的模型，根据新的数据预测出目标值，在这里就是指成功预测出未来的销售额。

首先我会在训练模型前对数据集进行探索，试图发现特征和目标值的关系。随后会用机器学习中的 xgboost 建模学习，根据不同参数建立不同模型，对不同模型的效率和性能进行相互比较，直到选择出表现最好的一个模型。最后，我会用**测试集**来进行数据验证。

评价指标

由于这个项目源自 kaggle（一个机器学习竞赛网站），最好的评价方式是把预测的结果提交到该平台，查看自己所得的分数，以及在所有提交分数中的排名。对于这个项目，kaggle 的评价指标是 RMSPE。

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

y_i 表示一家门店一天的实际销售量，$\hat{y}_i$ 表示一家门店一天的预测销售量

II. 分析

数据的探索

该项目包含三部分数据，train.csv 主要是门店的历史销售数据，store.csv 是门店的基本信息，test.csv 数据内容与 train.csv，只是时间是未来时间而不是历史，没有销售数据需要我们预测出结果，提交到 kaggle 进行评判。

在 train.csv 中，一共有 1017209 条样本，**没有缺省值**。共有 9 列，每列的含义如下：

Store	每个商店有一个独特的 id 值
DayOfWeek	一周中的第几天
Date	日期
Sales	销售量
Customers	某一天的客户数量
Open	商店是否开业：0 =关闭，1 =营业中
Promo	商店是否在当天有促销
StateHoliday	国家假期，a =公众假期，b =复活节假期，c =圣诞节，0 =无
SchoolHoliday	表示该商店或者日期是否受到公立学校关闭的影响

以其中一条样本例：

```
Store                1
DayOfWeek            5
Date        2015-07-31
Sales             5263
Customers          555
Open                 1
Promo                1
StateHoliday         0
SchoolHoliday        1
```

在该样本中，这家商店的 id 是 1，它在 2015 年 7 月 31 日周五当天开业，销售量是 5263，当天有 555 个客户。商店当天有促销活动，而且受公立学校关闭影响，但不是国家假期。其中 sales 就是我们后续建模中的目标值。

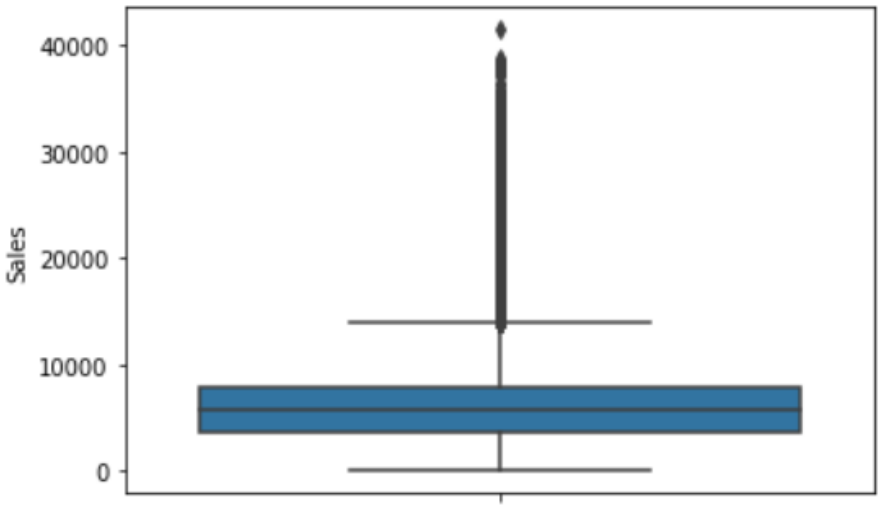

图 1-销售额箱图

由图 1 可得，我们的目标值即销售额还是存在着很多异常值。假如按照上述箱图的标准，即定义为大于 QU+1.5IQR 的即为异常值（QU 是上四分位，IQR 是四分位间距），那么 1017209 个样本中，有 26694 条样本的销售额为异常值，占总量的 2.62%，如上限由 QU+1.5IQR 为 QU+2.5IQR，则异常值减少到 7490 个，占总量的 0.74%，在后面会详细描述如何在训练模型前去除异常值。

在 store.csv 中，一共有 1115 条样本，共有 10 列，每列含义如下：

Store	每个商店有一个独特的 id 值
StoreType	区分 4 种不同的商店模式：a，b，c，d
Assortment	描述分类级别：a = 基础，b = 额外，c = 扩展
CompetitionDistance	距离最接近的竞争对手商店的距离
CompetitionOpenSinceMonth	最近的竞争对手大约的营业时间月份
CompetitionOpenSinceYear	最近的竞争对手大约的营业时间年份
Promo2	有些门店有一些持续的推广：0 =商店不参与，1 =商店正在参与
Promo2SinceWeek	描述商店开始参与持续推广的周
Promo2SinceYear	描述商店开始参与持续推广的年份
PromoInterval	描述了持续推广的开始间隔，而用月分命名新的促销活动。例如。 “二月，五月，八月，十一月”是指每一轮推广在该店的任何一年的二月，五月，八月，十一月份开始。

其中 CompetitionDistance，CompetitionOpenSinceMonth、CompetitionOpenSinceYear，Promo2SinceWeek，Promo2SinceYear 和 PromoInterval 列存在着缺失值，缺失数量和所占比例如下：

	缺失值量	占总样本比例
CompetitionDistance	3	0.27%
CompetitionOpenSinceMonth	354	31.75%
CompetitionOpenSinceYear	354	31.75%
Promo2SinceWeek	544	48.79%
Promo2SinceYear	544	48.79%
PromoInterval	544	48.79%

因为这些缺失值是由于商店信息不全或者没有所导致的，无法通过常用手段，比如取平均，取中位数等将缺失值填充。不过，在后面会应用的 xgboost 算法能够较好的识别和处理缺失值，在这里就不用对 store 数据集中的缺失值进行过多的关注和处理。

探索性可视化

1）门店的关闭状态

在商店销售数据集中，有些样本显示的开业状态，即 open 列，为关闭状态（0）。

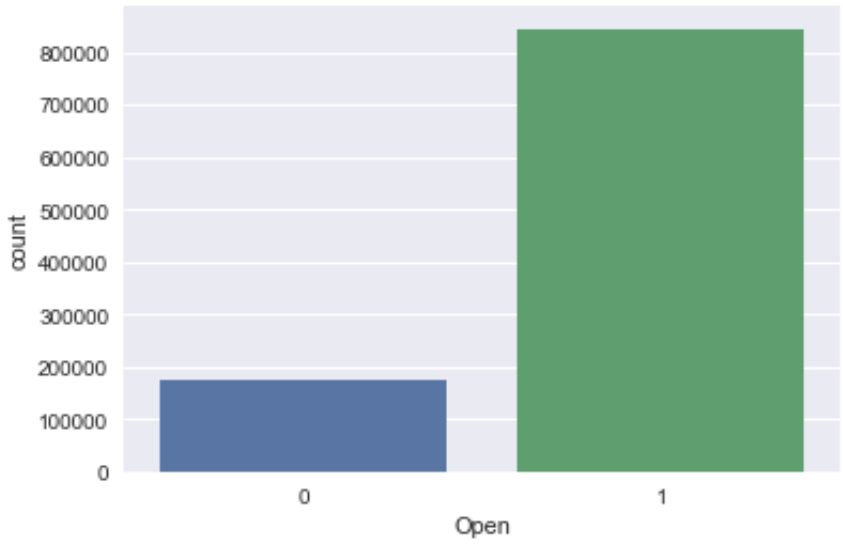

图 2-商店开业状态统计

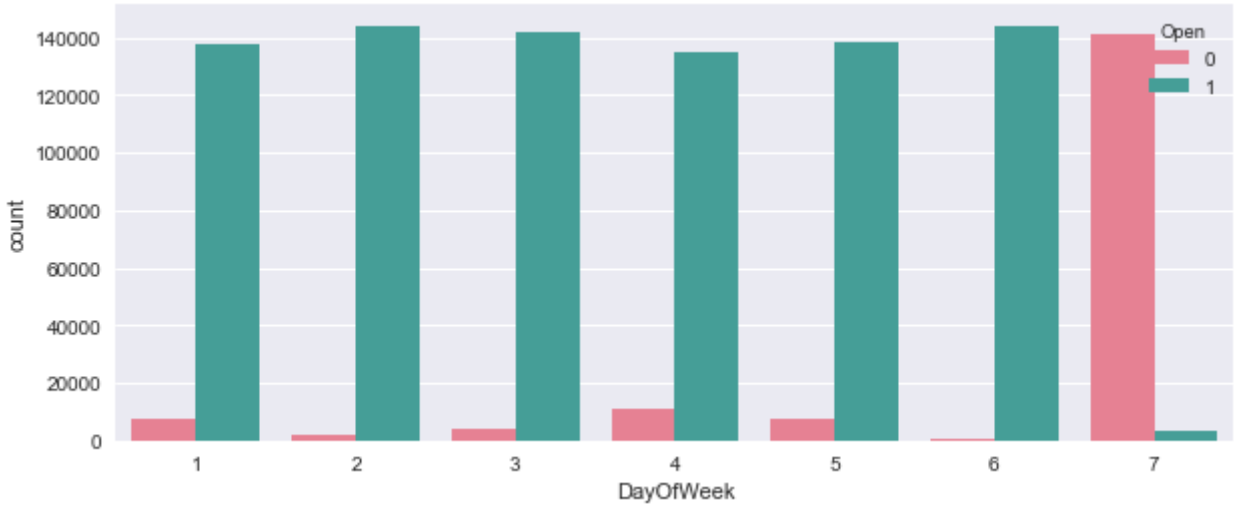

图 3-商店每周开业状态统计

由图 2 可得，在 1017209 条样本中，其中 172817 条样本显示为关闭状态，占了数据总和的约 17%，为了避免偏差，提高预测销售量的准确性，需要把这些样本从数据集中剔除。此外，根据图 3，发现大部分的商店在周日都处于关门状态，只有极少数的商店还在营业。

2）店铺种类

在剔除关闭状态的样本之后，首先对店铺类型做一个可视化。店铺类型，即 StoreType 一共有四种，a，b，c，d 四种店铺（为了后续建模方便重新编码为 1，2，3，4），对每种店铺从 1 月到 12 月的平均销售量做了一个可视化：

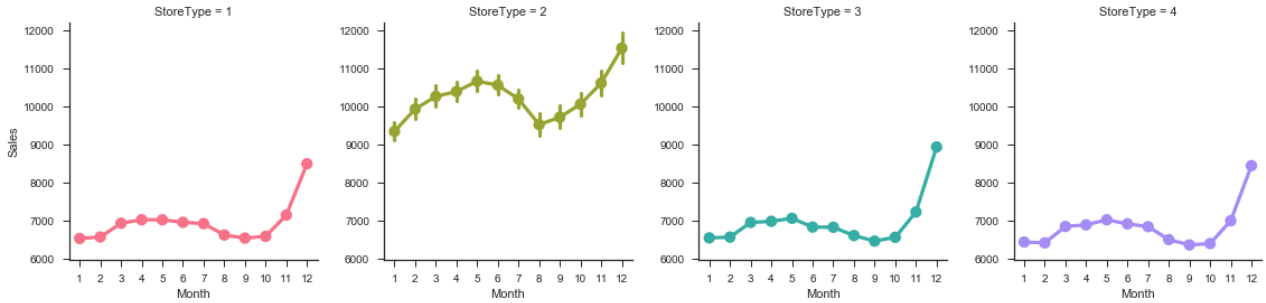

图 4-不同店铺种类的销售情况

店铺的类型对销售量的影响是存在的，尤其是 2 类（即 b 类）店铺，全年的平均销售量是高于其他类的店铺。

3）广告促销

在平常生活中，商家搞促销活动能一定程度上推动销售。由下图的可视化可得，在所有样本中，1 到 12 月的平均销售量，有促销的明显高于无促销的，由此可见，对 Rossmann 来说，有无促销也是影响当日销售量的一个重要因素，Rossmann 门店的广告促销的确起到了作用。

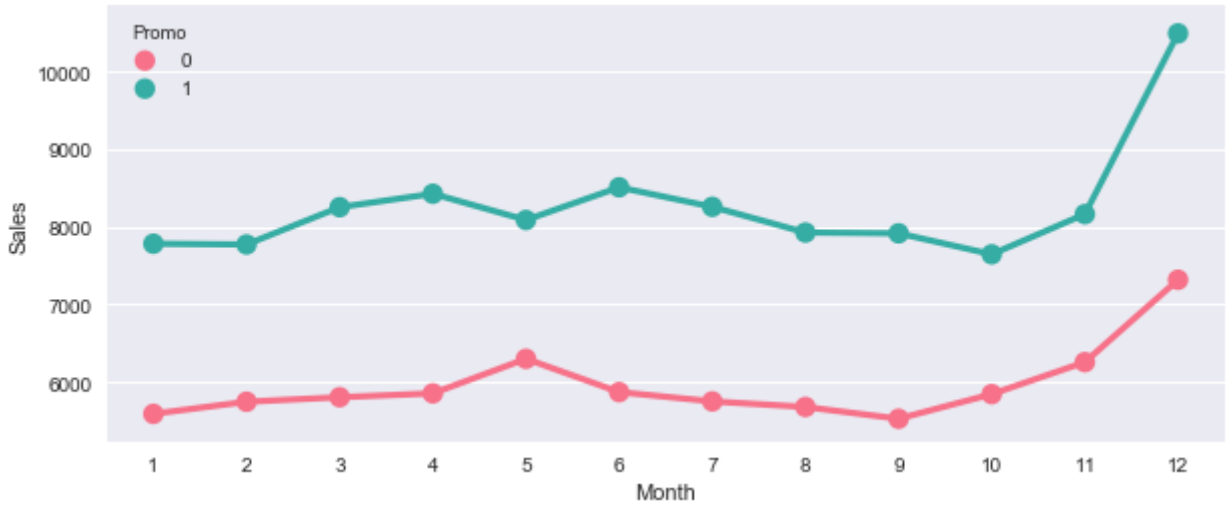

图 5-有无促销对销售量的影响

4）销售量与星期的关系

在剔除未开业的数据之后，对销售量和一周内不同日期的关系也进行了研究。发现周日的销售统计量小于其他日期，这是因为大多数的门店在周日都会选择关店的缘故，但是平均销售额却不低。

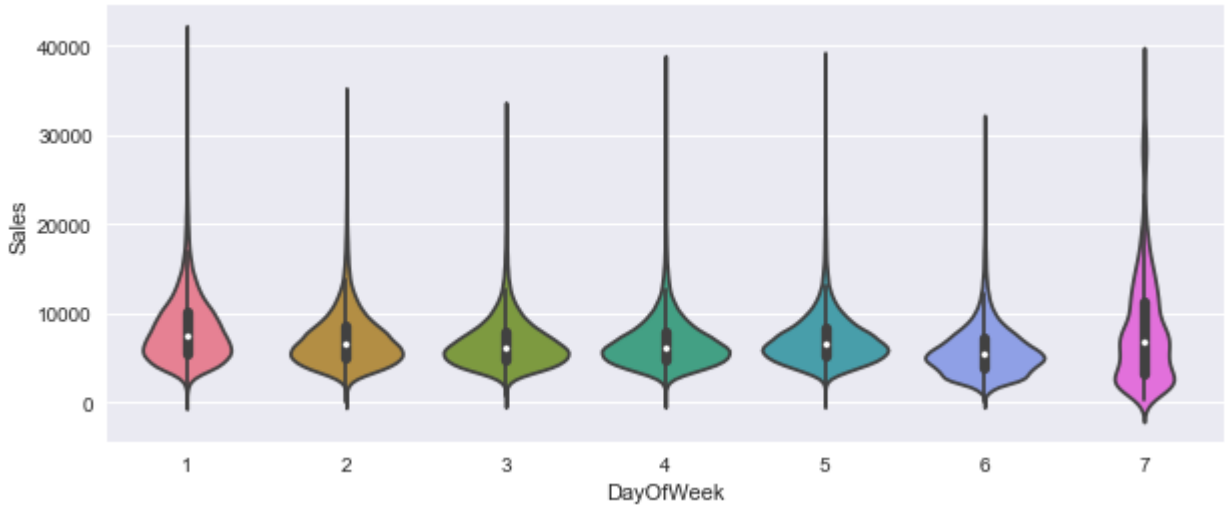

图 6-销售量和星期的关系

5）销售量与年月的关系

销售量和年月的关系如图 7 所示，2013 年和 2014 年有全年的数据，2015 年只有到 7 月的数据。不同年份的月均销售量差别较大，而且走势也并不有规律，尤其是 1 月份到 6 月份。不过下半年开始，2014 年和 2015 年月均销售量的走势逐渐相似，并且从 10 月开始大幅上升，到 12 月到达最高点。这可能和 12 月有圣诞节假期相关的。不过，还是能从每年的月均销售量中感受到，Rossmann 的销售整体情况是逐年提高的。

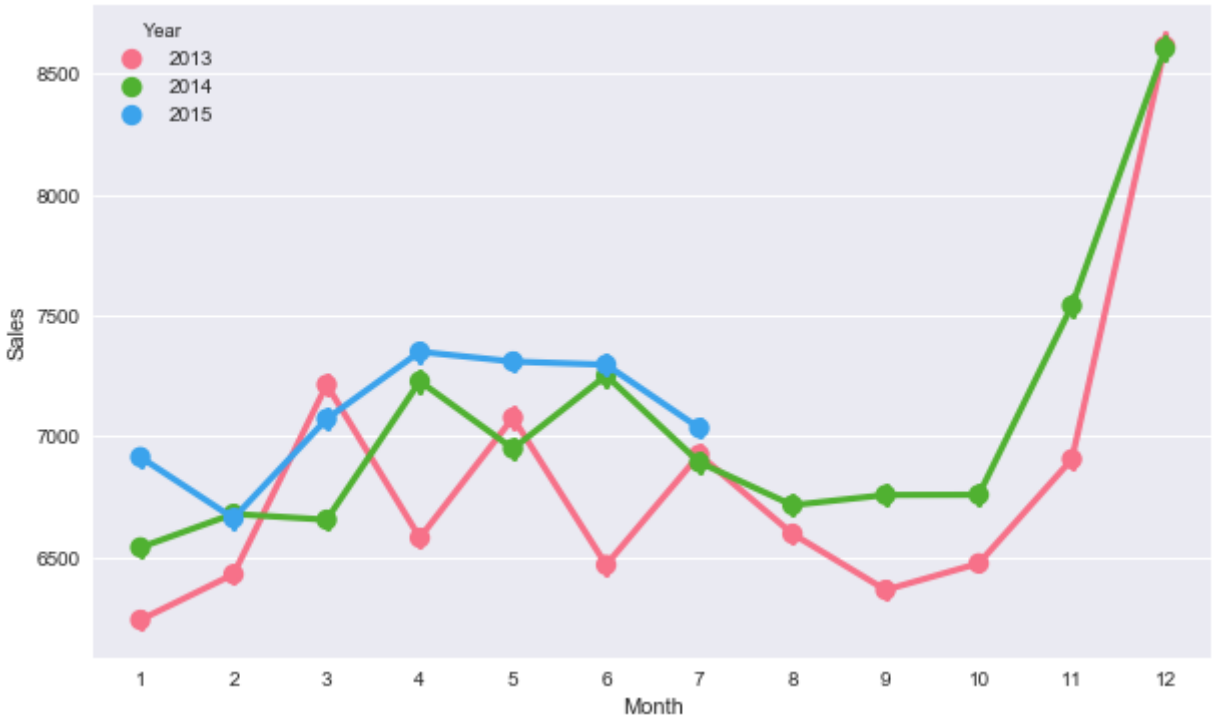

图 7-销售量和年月的关系

5）销售量与假期的关系

平均销售量额和假期的关系如下图所示（图 8）。是否假期对销售额的影响也比较明显。假期里部分

商店会选择关门，导致总体的销售量统计量低于非节假日。但是三种假期的平均销售额均高于非假期，这可能是因为在假期人们的购物意愿更大，愿意买更多或者更贵的东西作为节日储备或者节日礼物，特别是圣诞假期和复活节假期。

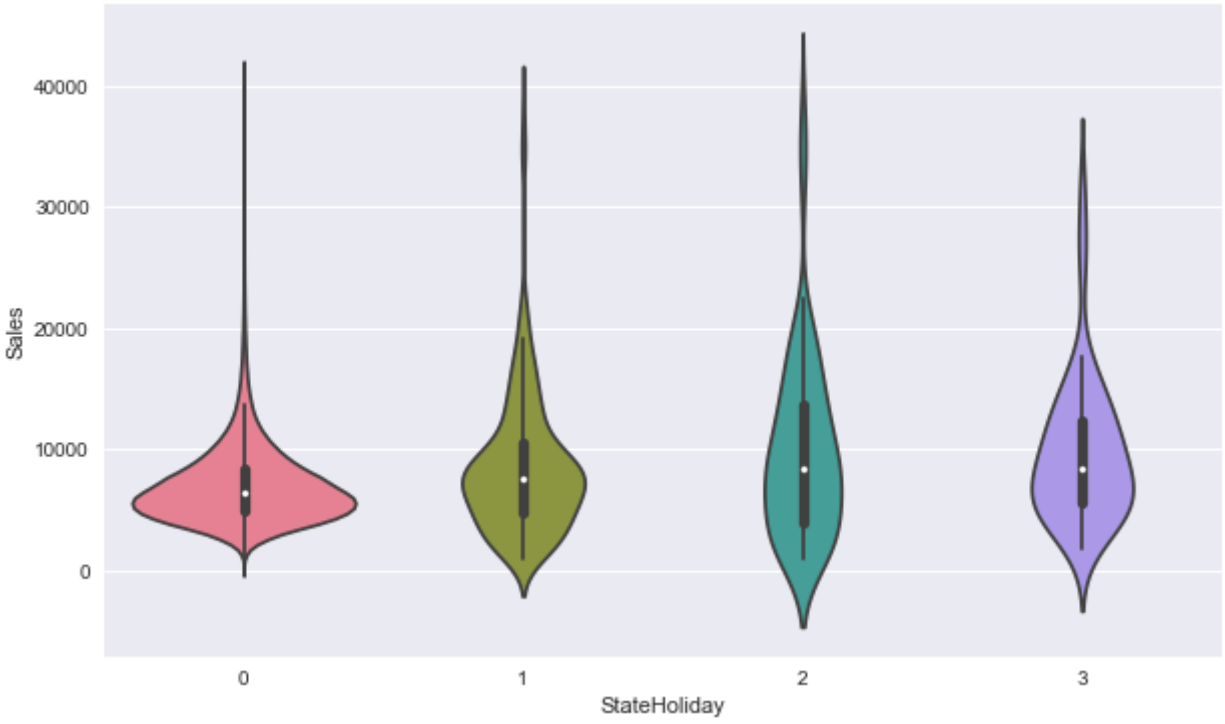

图 8-销售量与假期的关系

算法和技术

在本次项目中，主要应用了 xgboost 算法。在应用算法之前，对数据集进行了数据预处理，包括数据清洗、特征工程、数据分割等。

首先进行的是数据清洗，数据集中的目标值（即销售量）实际上存在着比较多的异常值，这些数据是需要清除的。主要用的方法有 Turkey's test 和标准差去除法，之后在具体实施的时候都会涉及。

清洗完数据之后便是要进行做特征工程，将数据属性转换为数据特征。这个项目主要涉及了几种特征工程：

1．时间戳的处理：将时间属性分离成多个维度，比如年月日小时星期
2．类别属性转换：将文本类别变量同一转换成数字类别变量
3．新的特征组合：根据已经有的特征，组合成新的特征，使算法学习的更有效率
4．特征选择（应用模型后）：根据模型的反馈，减少冗余特征，提高速度

然后在应用算法前还需要分割数据。因为本项目已经单独提供了测试集，所以输入模型的数据可以不用提取测试集，只要分割为训练集和验证集即可。训练集用于学习样本数据集，通过匹配一些参数来建立一个分类器，验证集的作用是当通过训练集训练出多个模型后，为了能找出效果最佳的模型，使用各个模型对验证集数据进行预测，并记录模型准确率。选出效果最佳的模型所对应的参数。

xgboost 算法

xgboost 是 2014 年 2 月诞生的专注于梯度提升算法的机器学习函数库，此函数库因其优良的学习效果以及高效的训练速度而获得广泛的关注。

机器学习中的学习算法的目标是为了优化或者说最小化 loss Function， Gradient boosting 的思想是迭代生多个（M 个）弱的模型，然后将每个弱模型的预测结果相加，后面的模型 Fm+1(x)基于前面学习模型的 Fm(x)的效果生成的。， Gradient boosting 算法中最典型的基学习器是决策树，尤其是 CART，GBDT 是 Gradient boosting 和 Decision Tree 的结合。要注意的是这里的决策树是回归树，GBDT 中的决策树是个弱模型，深度较小一般不会超过 5，叶子节点的数量也不会超过 10，对于生成的每棵决策树乘上比较小的缩减系数（学习率<0.1）。而 xgboost 扩展和改进了 GDBT，xgboost 中的基学习器除了可以是 CART（gbtree）也可以是线性分类器（gblinear）。

相比于传统梯度提升算法，xgboost 有很多优点，比如上述所提到的支持线性分类器，也就是可以认为它自带 L1 和 L2 正则化项的逻辑斯蒂回归（分类问题）或者线性回归（回归问题）。xgboost 还在代价函数里加入了正则项，降低了模型的方差，能够使学习出来的模型防止过拟合，更加简单。

$$Obj = \sum_{i=1}^{n} l(y_i, \hat{y}_i) + \sum_{k=1}^{K} \Omega(f_k)$$

Training loss　　Complexity of the Trees

图 9-xgboost 的目标函数

图 9 为 xgboost 的目标函数，后者为正则项，控制着模型的复杂度，包括了叶子节点数目 T 和 leaf score 的 L2 模的平方，比传统的 GBDT 更加细化。

$$\Omega(f_t) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2$$

Number of leaves　　L2 norm of leaf scores

图 10-正则项

xgboost 还支持列抽样，降低拟合，减少计算，速度上有显著的提升。除此之外，它还支持对缺失值的处理，对于特征值有缺失的样本，可以自动学习出它的分裂方向。在 Kaggle 的众多比赛中，很多排名前列的队伍都有用到过 xgboost 算法，因此这个项目我也准备尝试一下 xgboost 算法，虽然在之前从未实践应用过。

基准模型

Rossmann store 销售预测是 Kaggle 在 2015 年时候举办的比赛，当时共有 3303 个队伍（个人）参与，项目采用的评价指标是对比计算预测出来的销售额和真实的销售额之间的根均方误差百分比(RMSPE)，具体公式在第一章节的评价指标已经有说明。这个值越小，就说明预测和真实的销售额越接近，排名就会越高。所有队伍的分数都可以通过比赛页面的排行榜查询到[1]。我的目标是进入 Private

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

leaderboard 前 10%，即约前 330 名，这个名词对应的分数是 0.11773。这里需要注强调，分数指的 在 Private leaderboard 的得分而不是在 Public leaderboard 中的得分。在 Kaggle 比赛中，为了防止过拟合，比赛会设置不公开的 Private leaderboard，直到比赛结束后按照 Private leaderboard 上的分数决定最后的名次，所以 Private leaderboard 上的分数更能衡量模型的性能[2]。

总而言之，0.11773 将作为我的基准模型得分，如果最终我的分数能够超过这个分数，就意味着我的模型也能较好的预测销售额，并且和真实的销售额相差不大。

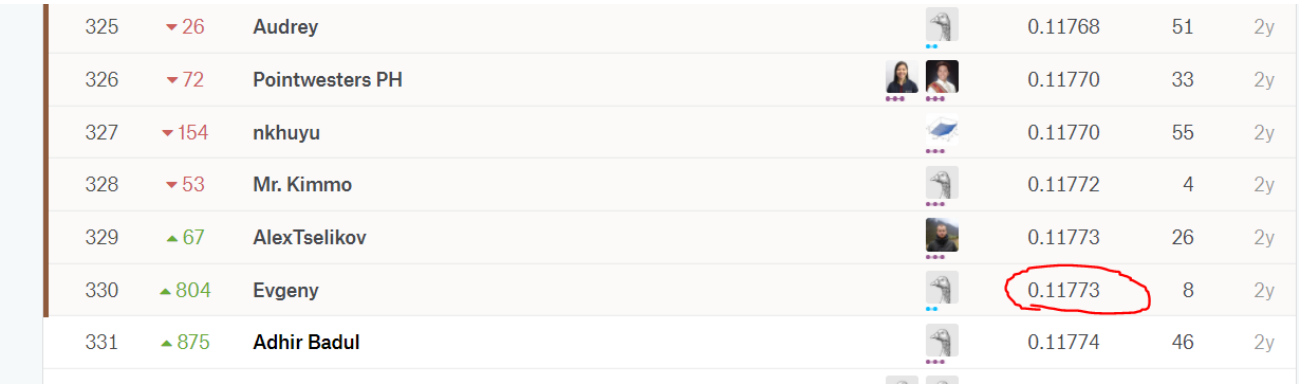

图 11-第 330 名的 RMSPE 分数

[2] https://www.kaggle.com/wiki/Leaderboard

III. 方法

数据预处理

去除异常值

第一步先是除去异常值，主要利用了 Turkey's test 和根据标准差去除异常值的方法。在后面的模型中，我将会应用两种不同的去除极值方法来得到两种模型，并选择最优的模型。[3]

```
def rm_outliers(df):
    q1 = np.percentile(df['Sales'], 25, axis=0)
    q3 = np.percentile(df['Sales'], 75, axis=0)
    k = 2.5
    iqr = q3 - q1
    df = df[df['Sales'] > q1 - k*iqr]
    df = df[df['Sales'] < q3 + k*iqr]
    return df

def rm_outliers_std(df):
    std = df['Sales'].std()
    mean = df['Sales'].mean()
    k = 3
    df = df[df['Sales'] >  mean - k*std]
    df = df[df['Sales'] < mean + k*std]
    return df
```

图 12-去除异常值的两种方法

这里值得一提的是，K 值的取值会影响去除异常值的数量，由于我在本地训练模型速度较慢，不可能尝试多个 K 值，所以结合了以往的经验和比赛论坛中其他人关于去除异常值的分享，我把两种去除异常值方法中的 K 值分别设为 2.5 和 3，目的是尽量保留了绝大部分的值。

特征工程之时间戳的处理

在原始数据中，时间是文本类型，并且以'年-月-日'的形式出现。这种原始时间数据是无法被机器学习的算法所学习的，所以要将时间戳拆分成几个小特征。

```
def data_processing(df):
    df['Date'] = pd.to_datetime(df['Date'])
    df['DayOfYear'] = df['Date'].apply(lambda x: x.dayofyear)
    df['WeekOfYear'] = df['Date'].apply(lambda x: x.week)
    df['Month'] = df['Date'].apply(lambda x: x.month)
    df['DayOfMonth'] = df['Date'].apply(lambda x: x.day)
    df['Year'] = df['Date'].apply(lambda x: x.year)
    return df
```

图 13-时间戳的拆分

在这个项目里我拆分出了 5 个主要特征，分别是年份(Year)，月份(Month)，该日所在的周是当年的第几个周(WeekofYear)，该日当年的第几个天(DayofYear)，以及该日是当月的第几天(DayofMonth)。注

[3] 由于用 Turkey's test 去除极值后的模型已经符合要求，所以为了节省运算时间，不再尝试用绝对值去除异常值法，两者实际上保留的有效样本数量比较接近。

意，该日是星期几的不用提取，因为原始数据已经为我们提供了。

特征工程之类别属性转换

在在原始数据集中，很多数据都是以字母的形式所表现的，不能直接被算法读取，需要转换成数字。比如将商店的类型由 a、b、c、d 转换成 1、2、3、4。

```
mappings = {'0': 0, 'a': 1, 'b': 2, 'c': 3, 'd': 4,'Jan,Apr,Jul,Oct':1,'Feb,May,Aug,Nov':2,'Mar,Jun,Sept,Dec':3}
data['StoreType'].replace(mappings, inplace=True)
data['Assortment'].replace(mappings, inplace=True)
data['StateHoliday'].replace(mappings, inplace=True)
data['PromoInterval'].replace(mappings, inplace=True)
```

图 14-分类变量的转换

特征工程之新特征（商店）

每家商店的日均销售额销量、预测目标日期前一个月（两个月/一个季度/半年）的平均销量，日均顾客书，预测目标日期前一个月（两个月/一个季度/半年）的平均顾客书，很大程度上也会对未来的预测起到参考意义，所以提取这部分数据是很有必要的。又由于是否促销对销售量影响很大，所以这些均值还要再引入是否有促销活动。

最后我一共提取了 16 个特征（均销售量，均顾客量各 8 个）。

mean_sales_promo, mean_sales_no_promo， mean_sales， mean_sales_2013， mean_sales_2014， mean_sales_2015， mean_store_sales_1month， mean_store_sales_2months， mean_store_sales_3months， mean_store_sales_6months

mean_customers_promo， mean_customers_no_promo， mean_customers， mean_customers_2013， mean_customers_2014， mean_customers_2015， mean_customers_1month， mean_customers_2months， mean_customers_3months， mean_customers_6months

合并销售数据和店铺数据

我们需要讲销售数据和店铺数据结合在一起，利用 pandas 提供的函数，以 store 为 key，很快就能完成合并。

```
#合并销售和商店
def merge_sale(sale_data, store_data):
    train = sale_data.join(store_data, on='Store', rsuffix='_')
    train = train.drop('Store_',axis=1)
    return train
```

图 15-合并销售和商店数据

特征工程之特征合并

在合并完的数据中，有两组数据引起了我的注意。分别是'CompetitionOpenSinceYear', 'CompetitionOpenSinceMonth'和'Promo2SinceYear'，'Promo2SinceWeek'。一个是竞争对手开业的年份和月份，一个是促销活动开始的年份和第几周。这四个特征分开来看好像无法对销售量产生直接的影响，但是如果两两合并就会得到竞争对手开始的时间和促销的时间，我们就可以计算出距离当日多久，显然会对销售量产生一定的影响，这两个组合特征比四个单独的特征更能反映竞争对手和促销活

动的属性。

```
data['CompetitionOpen'] = 12 * (data['Year'] - data['CompetitionOpenSinceYear']) + (data['Month'] - data['CompetitionOpenSinceMonth'])
data['PromoOpen'] = 12 * (data['Year'] - data['Promo2SinceYear'])+ (data['WeekOfYear'] - data['Promo2SinceWeek']) / 4.0
data['PromoOpen'] = data['PromoOpen'].apply(lambda x: x if x > 0 else 0)
```

图 16-特征组合

特征工程之临近节假日

我们在第二部分分析中的可视化中已经有所察觉，节假日对销售量的变化影响很大。从正常推断来看，距离节假日越近，人们为了准备度过假日越会发生购物行为，从而导致销售量提升，比如圣诞和复活节假期。因此，可以从节假日特征中创建一个新的特征，用于计算距离某节假日还有几天。这个值可以为正可以为负，正表示还有几天，负表示已经过去了几天。

```
holidayofyear = sorted(train[train['StateHoliday'].isin([1,2,3,4])]['DayOfYear'].unique())
def day2holiday(df,holidayofyear):
    for holiday in holidayofyear:
        df['DaysToHoliday' + str(holiday)] = holiday - df['DayOfYear']
    return df
```

图 17-计算距离某节假日还有几天

特征工程之每个日期之前的均销售量（额外特征）

这块的特征工程是我在阅读了该次比赛冠军的访谈[4]而借鉴创立的。总共新建了两个特征，*'past_quater_mean_sale'*和*'past_year_mean_sale'*。这个特征工程是计算了每家商店每一天之前半个季度和一年的均销售量。

需要注意的是，这个特征和之前提到的计算商店均值不同点在于，一个是静态，一个是动态。比如之前添加的一个特征'*mean_store_sales_1month*'，指的是这个商店在我们预测时间段前一个月的均销量（实际上就是这个商店在 2015 年 7 月的日均销售额），那么在训练集中，凡是这家商店的这个"*mean_store_sales_1month*"都是同一个值。这里就会有一个问题，某条样本，比如商店 a 在 2013 年 4 月的 1 日，它的"*mean_store_sales_1month*"同样也是 2015 年 7 月的日均销售额，其实该日的销售额预测没有什么实际意义。我们需要的是一个动态的值，我们需要参考的是 2013 年 3 月的日均销售量，所以要引入**每一天之前某段时间**的均销售量。这个值是会随着时间变化而变化，比如 5 月 1 日前一个月的销售均值是 4 月份的，6 月 1 日前一个月的销售均值是 5 月份。

在训练集中每个日期之前的均销售量比较好求得，但是在测试集中，由于无法用预测的销售量去求下一个日期的销售量，所以统一按照前 90 天和 183 天的销售量为准。

不过由于我统一按照前 90 天和 183 天的销售量，所以我担心可能会存在非但没有提高模型预测能力，反而降低了模型精准度的情况存在，所以在后续跑模型的时候，会按照带上这个特征和没有这个特征，分别建立模型，最终再进行评估。

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

```
#4、为每个日期添加过去一个季度，过去半年，过去一年，过去两年的这家店的平均日销售量
def store_sales_each_day(sale):

    def add_mean(store,sale,current_date,past_time):
        past_date = current_date - timedelta(days=past_time)
        mean_sale = sale[(sale['Date'] < current_date) & (sale['Date'] > past_date) & (sale['Store'] == store)]['Sales'].mean()
        return mean_sale

    sale['past_quater_mean_sale'] = sale.apply(lambda row: add_mean(row['Store'], sale, row['Date'], 90), axis=1)
    sale['past_year_mean_sale'] = sale.apply(lambda row: add_mean(row['Store'], sale, row['Date'], 183), axis=1)

    return sale

#测试集调整
def store_sales_each_day_for_test(sale,test):

    def add_mean(store,sale,current_date,past_time):
        past_date = current_date - timedelta(days=past_time)
        mean_sale = sale[(sale['Date'] < current_date) & (sale['Date'] > past_date) & (sale['Store'] == store)]['Sales'].mean()
        return mean_sale

    test['past_quater_mean_sale'] = test.apply(lambda row: add_mean(row['Store'], sale, row['Date'], 90), axis=1)
    test['past_year_mean_sale'] = test.apply(lambda row: add_mean(row['Store'], sale, row['Date'], 183), axis=1)

    return test
```

图 18-提取动态均销量特征

其他处理

此外，为了提高运算速度和减少运算量，将不开店的，销量小于等于 0 的，商店并没有出现在测试集中的样本全部删除。而且需要删除''Customers'，'Date'和四个之前已经被组合的特征。

执行过程

在数据处理完之后，就是输入到我们选定的算法中，让机器自己学习。

分割数据

首先我们要分割数据，一部分作为验证集，一部分为训练集。由于对未来的预测参照最近的会比较准确，所以把数据中的最后六个月的样本作为我们的验证集，其余作为训练集，而不是按照传统的比例进行随机分割。此外由于 12 月分并不在预测时间内，而且 12 月的平均销量比较高，不够稳定，也在训练集中予以删除。

```
X_valid = train[(train['Year']==2015) & (train['WeekOfYear'] <=31) & (train['WeekOfYear'] >=26)]

X_train = train[(train['Year']==2015) & (train['WeekOfYear'] <26)] + train[(train['Year']!=2015)]

X_train = train[train['Month']!=12]
```

图 19-分割训练集和验证集

随后，把目标值，即销量用 log 的方法进行归一化，为了提高模型的准确度。最后将验证集和测试集都转换为 xgboost 能够读入的格式。

```
y_train = np.log1p(X_train.Sales)
y_valid = np.log1p(X_valid.Sales)
dtrain = xgb.DMatrix(X_train[features], y_train)
dvalid = xgb.DMatrix(X_valid[features], y_valid)
```

图 20-log 目标值并转换为 xgboost 可读格式

参数选择

接下来就是选择 xgboost 的参数，在这个项目中，由于 xgboost 每跑一次模型时间较长，无法利用 grid search 办法去选取最优变量组合，因此模型参数的确定是基于 xgboost 的通用参数和前人的经验，只有一两个参数我将会根据模型得分进行微调。所涉及的参数如下：

'objective': 这个参数的默认值为 'reg: linear'，作用是把这个问题当作线性回归问题进行处理。

'max_depth': 树的最大深度，树的深度越大，则对数据的拟合程度越高（过拟合程度也越高）。该参数也是控制过拟合，在这个模型中我将尝试三个不同的 max_depth 值，8，9 和 11。

'booster': 有两种模型可以选择 gbtree 和 gblinear。gbtree 使用基于树的模型进行提升计算，gblinear 使用线性模型进行提升计算，需要选择 gbtree。

'min_child_weight': 孩子节点中最小的样本权重和。如果一个叶子节点的样本权重和小于 min_child_weight 则拆分过程结束。在现行回归模型中，这个参数是指建立每个模型所需要的最小样本数。调大这个参数能够控制过拟合，所以设为 50。

'eta': 0.05，为了防止过拟合，更新过程中用到的收缩步长。在每次提升计算之后，算法会直接获得新特征的权重。 eta 通过缩减特征的权重使提升计算过程更加保守。通常最后设置 eta 为 0.01~0.2。在这里取 0.05。

'alpha': L1 正则的惩罚系数，当数据维度极高时可以使用，使得算法运行更快。这里设置成 2。

'subsample': 用于训练模型的子样本占整个样本集合的比例。如果设置为 0.5 则意味着 XGBoost 将随机的从整个样本集合中抽取出 50%的子样本建立树模型，这能够防止过拟合。我设置成 0.9。

'colsample_bytree': 在建立树时对特征随机采样的比例。设置为 0.7。

'silent': 静默模式设置。设置为 1，开启静默模式，不会输出任何信息。这个参数不影响模型建立。

'seed': 随机数种子，通过设置此数值，可以复现随机数据的结果。随机取值就可以了，默认值是 0。这里设为 1301。

'gpu_id':设为 0，默认值。

'tree_method' : 为了加快模型的训练和性能，利用 gpu 版本的 xgboost 进行训练，所以这款 i 需要取值为'gpu_hist'。

训练模型

为了尽可能的提高精度，我将 num_boost_round 设为 5000。以下代码参考了 kaggle 上的一个 kernel[5]。

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

```
#rmspe计算以及赋值给xgboost
def rmspe(y, yhat):
    return np.sqrt(np.mean((yhat/y-1) ** 2))

def rmspe_xg(yhat, y):
    y = np.expm1(y.get_label())
    yhat = np.expm1(yhat)
    return "rmspe", rmspe(y,yhat)

watchlist = [(dtrain, 'train'), (dvalid, 'eval')]

gbm = xgb.train(params, dtrain, num_boost_round, evals=watchlist, \
  early_stopping_rounds=100,
  feval=rmspe_xg,
  verbose_eval=True)

print("Validating")
yhat = gbm.predict(xgb.DMatrix(X_valid[features]))
error = rmspe(X_valid.Sales.values, np.expm1(yhat))
print('RMSPE: {:.6f}'.format(error))
```

图 21 – 模型的训练

预测销售额并生成符合规范的 csv 文档

```
print("Make predictions on the test set")
dtest = xgb.DMatrix(test[features])
test_probs = gbm.predict(dtest)

# Make Submission
result = pd.DataFrame({"Id": test["Id"], 'Sales': np.expm1(test_probs)*0.98})
result.to_csv('n3000_k_25max_depth_11.csv', index=False)
```

图 22 – 预测销售额

保存模型

```
gbm.save_model('n5000_k_25max_depth_11.model')
```

图 23 – 保存模型到本地

完善

这个项目中所用到的技术和算法也是根据模型表现不断完善的。在特征工程方面，最早我主要是应用了时间戳的转换和类别特征转换，后来又陆续添加了一些组合特征，最后的结果还算不错。其中我对*'past_quater_mean_sale'*和*'past_year_mean_sale'这两个*特征进行了重点研究，分别在有这两个特征和没这两个特征的情况下跑了模型，并且根据结果最终保留了该特征，认为这两个特征可以提高模型的精度，具体的内容在模型评价中会详细描述。

至于算法方面，由于是我第一次应用 xgboost 算法，所以前期理解算法和各种参数所代表的含义花了

点时间。调参过程由于受本地计算机性能的限制，大部分的参数采用了基准的参数或者根据前人经验较为广泛使用的参数，只是重点关注了一下 max_depth 这个参数，根据不同的 max_depth 表现来选择最终的模型。

由于我的计算机比较落后，跑 5000 轮的 xgboost 模型大概需要五六个小时，所以为了提高速度，安装了 gpu 版本的 xgboost，但是效果不是特别理想，时间只是从五六个小时降低到了两个小时。所以无法尝试其他的参数，也无法利用 grid search 办法来为各个参数选出最优解。

最后，为了提高在 Kaggle 上的排名，我还采取了一个小技巧（其实很多人在讨论区都有提到）。根据 Correcting prediction for RMSPE[6] 这篇文章，当实际上的销售量比较小的时候，预测出来的和实际可能方差会稍大，所以可以进行适当补偿，从而获得更小的 RMSPE。我对最后的结果做了销售量乘以 0.98 的修正，在 Kaggle 上的分数也得到了明显的提高，说明修正还是非常有效的。

[6] https://www.kaggle.com/c/rossmann-store-sales/discussion/17601

IV. 结果

模型的评价与验证

因为这是 Kaggle 上的比赛，所以模型的评价和验证也比较简单，只要比较把每个模型预测的分数提交至比赛页面即可得到 Private leaderboard 的分数，并根据分数来找到表现最优的模型。我一共跑了六个模型，其中有无额外特征（*'past_quater_mean_sale* 和 *'past_year_mean_sale'*）和 max_depth 的选择是这些模型的主要区别。最终六个模型预测的销售额 RSMPE 最终得分[7]如下：

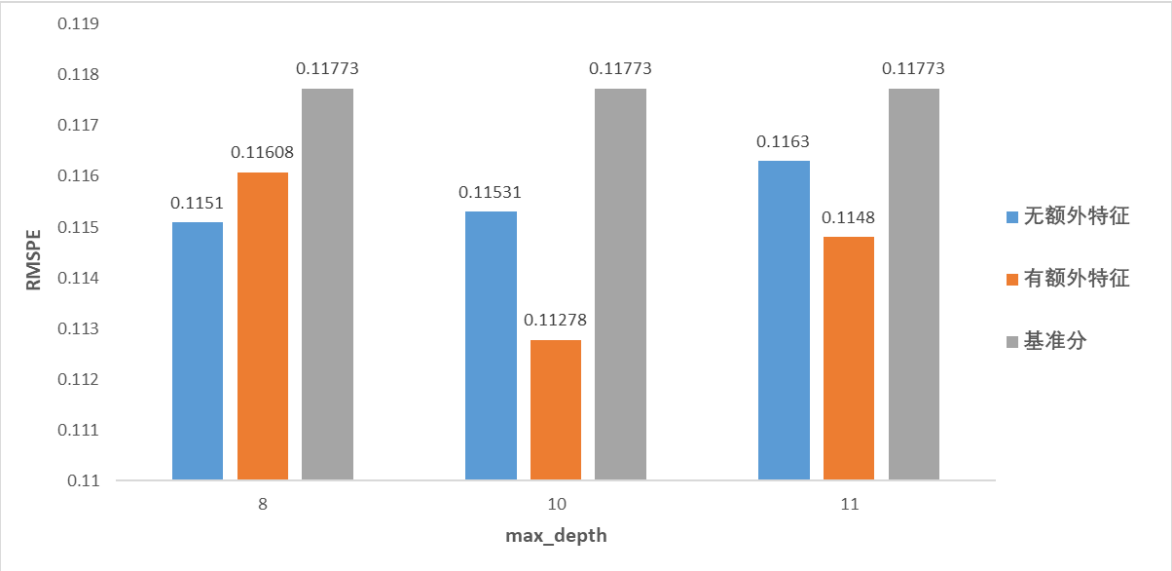

图 24 – 六个模型得分

由上图可得，当 max_depth = 10，并且数据包含额外特征（即 'past_quater_mean_sale' 和 'past_year_mean_sale' ）时，模型预测的销售额得分最高，RMSPE 为 0.11278，其次是 max_depth 为 11，训练数据包含额外特征的模型，RMSPE 为 0.1148。从目前得到的结果来看，'past_quater_mean_sale'和'past_year_mean_sale'两个特征，对准确预测销售量的确有很大的提升。除此之外，max_depth 对于模型的精度也有着明显的影响，不同的取值会导致最后的结果不同。

在实际比赛中，单一的模型可能会存在着一些不稳定性，融合多个模型可能能提升模型的精度。因此我也把六个模型预测的销售额进行了简单平均，合成了新的预测值。这个预测 RSMPE 果然比单一模型的得分更低，仅 0.11241。

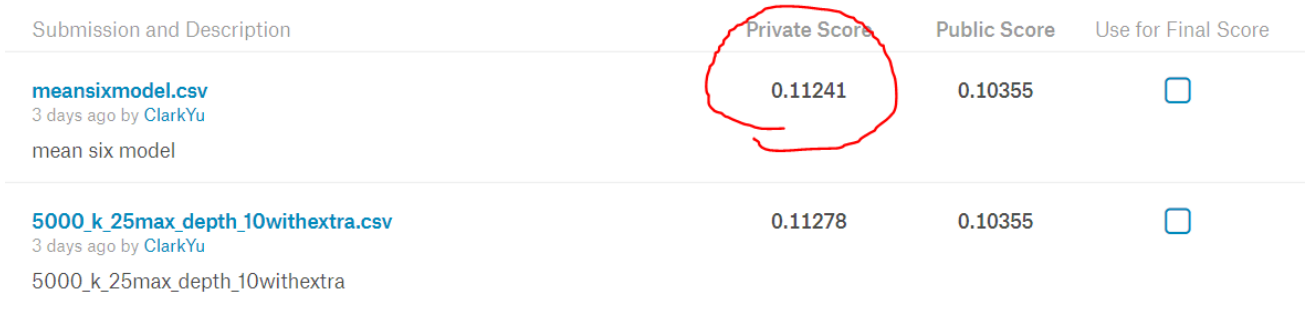

图 25 – 在 Kaggle 提交页面的分数

[7] 提交至 Kaggle 的预测 csv 文件均有存档，附在提交文件夹中，并对每个文件有详细解释。

合理性分析

可以从图 21 得出，学习得到的六个模型预测结果的 RMSPE 均小于在前文基准模型中定下的基准分数，即 0.1173，说明它们的精度都优于基准模型。根据 Kaggle 上的榜单可得，六个模型中单个得分最高的 0.11278 可以排在 Private Leaderboard 中的第 77 名，也就是**前 2.3%**，融合六个模型后的分数更低，只有 0.11243，可以排到第 60 名，也就是**前 1.81%**。这个分数对我来说已经是非常出乎意料了，毕竟一开始给自己定的目标是进入前 10%。此外，从商业角度来说，我的模型能够较好的预测 Rossmann 店铺未来的一个销售量情况，解决了一个商业问题，为店铺的商业活动提供了较为可靠的参考。

V. 项目结论

结果可视化

关于模型的最终得分已经在上一部分的模型评价与验证中有详细的解释。此外，xgboost 算法提供了特征评分。在训练的过程中给出各个特征的评分，从而表明每个特征对模型训练的重要性。我选取表现最优的单一模型，也就是得分 0.11278，当 max_depth = 10，并且数据包含额外特征（即'past_quater_mean_sale'和'past_year_mean_sale' ）的模型的特征评分，如下图所示：

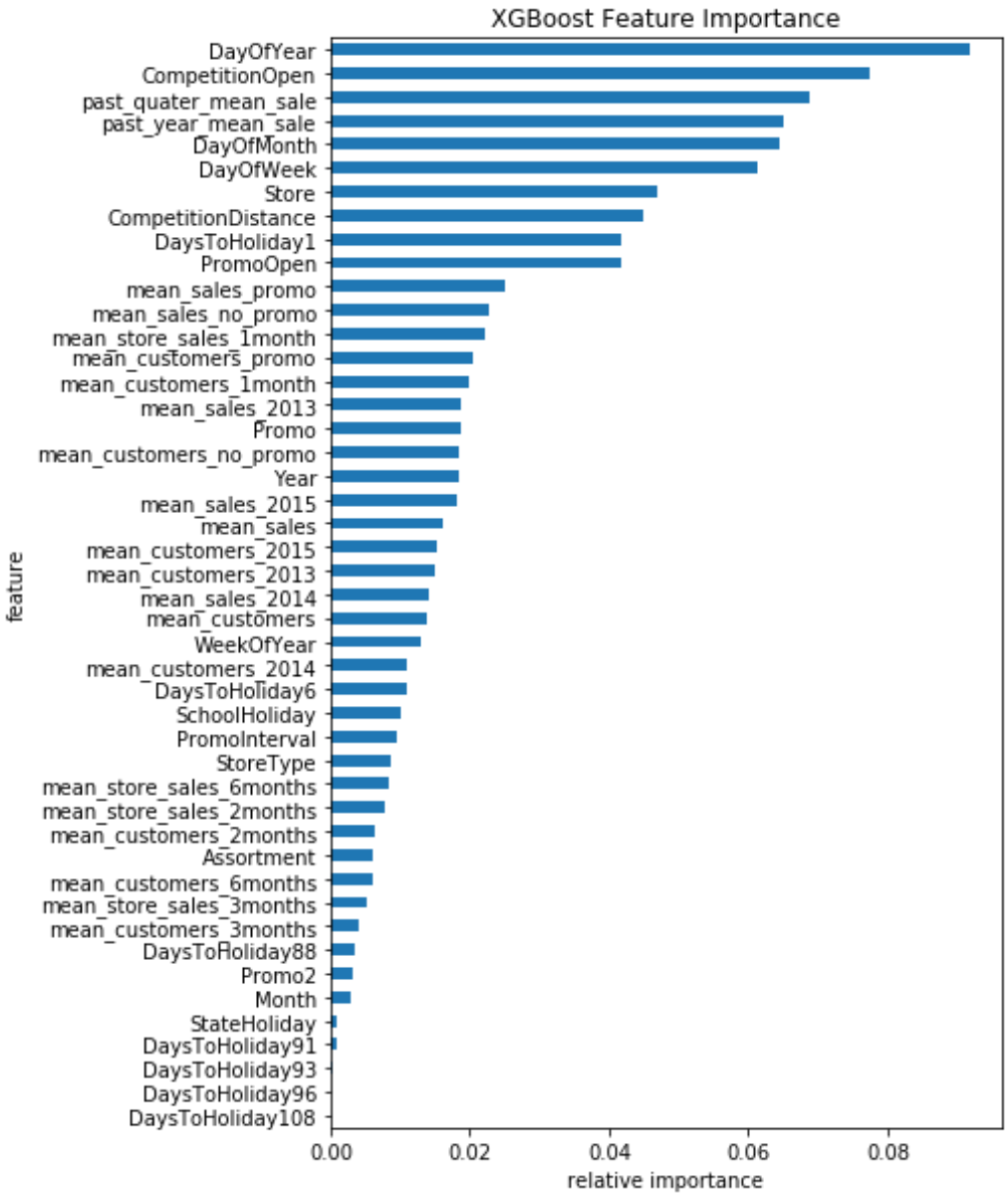

图 26 – 特征评分

我们可以看到，‘Dayofyear’,‘Competitionopen’，'past_quater_mean_sale'，'past_year_mean_sale'和‘DayofMonth’是前五比较重要特征。时间戳的转换和组合新的特征，的确对训练更高精度的模型起到了显著的效果。另外，还可以发现过去一个月的相关特征重要性往往比过去两个月，一季度，半年的

重要性要高，这在销售额和客户数的特征上都有体现。此外，还有一些特征重要性非常低，比如‘StateHoliday‘，‘DaystoHoliday91’等，如果后续想要提高模型训练速度，其实可以将这些重要性很低的特征删除。

对项目的思考

在这个项目中完整的经历了一次机器学习项目的流程。完整的经历了‘提出问题-获取数据集-清洗数据-特征工程-算法选择-训练模型-验证结果-合理解释’的这么一个过程，让我获益匪浅。

其中，比较有意思的有两点：一、时间戳的转换技巧，这是我第一次接触时间序列分析的问题，这个技巧让我觉得非常的有用；二、组合特征，组合特征能够提升模型的精度，而且组合特征也并不是盲目组合，是需要建立在现实情况的基础上，这个技巧就非常灵活，也需要我们对问题有较好的了解，还是非常有趣的。当然我也遇到了比较困难的地方。主要的困难来自于电脑性能的问题。一开始想更多的调整参数，跑更多的轮次来提升模型的精度，但是跑一次 5000 轮的需要 2 个小时左右，即使我使用了 GPU 加速，所以最后只跑出了六个模型，但所幸结果还是不错。这个项目也让我萌生了提升机器的想法。

当然，最终要的是这个项目能否解决实际的问题。这个项目让我了解到机器学习的确可以和商业紧密的结合，我的模型最终也较好的预测出了商店未来六周的销售额，商店可以根据预测出的销售额及时的调整运营策略。

需要作出的改进

虽然项目取得了我预期的目标，但是我觉得还是有很多地方可以做出改进。

第一，数据清洗的过程。在项目中我最后只是应用了 Turkey‘s Test 去除异常值的方法，而且 k 值取了 2.5，如果有时间可以再尝试不同的 K 值，以及其他的去除异常值的方法。

第二，特征的选择过程中，我只简单的提取了一些特征，比如每个商店均销售量，每日均客户数，其实还可以再进行深入的挖掘，比如考察销量和客户的比率，销量和学校假日的关系，还可以引入外部的数据，比如天气因素。

第三，在算法方面，只是应用了 xgboost 算法，其实还可以有更多的尝试，比如神经网络算法，lightgbm 算法等。此外，算法的调参也是可以做出改进的地方，我只是调整了 max_depth 这个参数，如果有时间，以及机器性能允许的话，其实可以对更多的参数进行调整，比如学习率等，最后找出最优参数组合。