机器学习纳米学位

毕业项目

王珏

优达学城

2019 年 10 月 24 日

I. 问题的定义

项目概述

Rossman 是欧洲的一家连锁药店，其在欧洲 7 个国家/地区拥有 3,000 多家店铺。因经营需要，Rossman 的经理需要提前 6 周来预期店铺的销售额。不同的店铺面临的经营情况大相径庭，他们的销售受到许多因素的影响，包括促销，竞争，学校和州假期，季节性和地区性。

该项目是一个典型的监督学习问题，关于监督学习，我已经学习了许多模型，包括感知机，决策树，随机森林，朴素贝叶斯等。由于我个人的工作行业关系，时常会遇到这类监督学习的问题，因此，希望利用这个机会来提高自己的分析能力，因此我选择了 Rossman 销售额预测。

问题陈述

监督学习可以分为回归问题和分类问题，我们需要根据 Rossman 商店过去的销售情况来预测未来的销售额，因此这是一个回归问题。一个店铺的销售额受到了诸多因素的影响，从参赛者经验上看，只使用其给出的特征难以得到一个较好的效果，创造衍生特征成为了必选项。同时，该数据集较大，其训练集包含了上百万条数据，因此在模型训练时，还需要兼顾运算速度。

评价指标

本次项目的评估指标由 Kaggle 给出，为均方根百分比误差(RMSPE)，其计算公式如下图:

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

RMSPE 与 均方误差 RMSE 相比，区别在于在计算误差时要除以真实值 yi，我认为这样的好处当 yi 的取值范围很大时，比如最大值与最小值相差几十万倍，那么当高值预测出现误差时，产生的误差就会非常大，从而影响到了对其他值预测误差的评估。 通过将误差除以 yi，相当于在归一化这些误差，这样能够平等对待较大值与较小值。

II. 分析

数据的探索

在 Rossman 项目中，数据一共被包含在 3 个表格中，分别为 Store，train 和 test。Store 表记录了店铺的相关信息，train 表则是某个店铺一天的营业情况，test 则是我们需要去预测的数据。所有表格中包含的特征可见下表:

特征释义

1. Id:测试集内(商店、日期)的组合。
2. Store:表示每个商店的唯一 Id。
3. Sales:任意一天的销售额，也是我们要预测的字段。
4. Open:是否开门，0=关门，1=开门。
5. StateHoliday:国家假日，一般假日国家假期都会关门，所有学校在公共假日都会关门，a=公共假日，b=东部假日，c=圣诞节，0=不是假日。
6. SchoolHoliday:校园假日，指当日学校是否关闭。
7. StoreType:商店类型，有四种，abcd。
8. Assortment:分类级别，a=基础，b=额外，c=扩展。
9. CompetitionDistance:竞争对手距离。
10. CompetitionOpenSince\[Month/Year\]:给出最近竞争对手的开张时间。
11. Promo:表示商店当天是否进行促销
12. Promo2:表示商店是否进行持续的促销活动，0=没有参数，1=参与。
13. Promo2Since\[Year/Week\]:商店开始持续促销的年/星期。
14. PromoInterval:持续促销活动开始的间隔，"Feb,May,Aug,Nov"表示给定商店某一年的 2589 月开始持续促销活动。

接下来需要查看数据的完整性。首先查看 Store 表。发现总店铺是 1115 个，但是有些数据是缺失的。

```
Data columns (total 10 columns):
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
dtypes: float64(5), int64(2), object(3)
memory usage: 87.2+ KB
```

首先查看 CompetitionDistance，发现 3 个缺失值不光距离数据缺失了，连后面的开业年月也没有了。推测应该是数据收集的时候就没有找到这些信息。

```
store_data[pd.isnull(store_data['CompetitionDistance'])]
```

	Store	StoreType	Assortment	CompetitionDistance	CompetitionOpenSinceMonth	CompetitionOpenSinceYear	Promo2	P
290	291	d	a	NaN	NaN	NaN	0	
621	622	a	c	NaN	NaN	NaN	0	
878	879	d	a	NaN	NaN	NaN	1	

在继续查看促销信息，虽然 3 个特征都是 571 个（却 544 个），但是还是需要确下是不是缺失在同一家店铺中。

```
In [8]: store_data[(pd.isnull(store_data['Promo2SinceWeek']))&(pd.isnull(store_data['Promo2SinceYear']))
                   &(pd.isnull(store_data['PromoInterval'])))]['Promo2'].value_counts()

Out[8]: 0    544
        Name: Promo2, dtype: int64
```

确认完毕，的确是缺失在相同的店铺了。那么促销信息和竞争者的信息我就用 0 值来补完。

接下来是 train 和 test 表的数据。从下图可以看到，train 表的数据完整，test 在 Open 特征有缺失。本来此处我对 Open 均按照 1 补全了，但是在建模的时候，认为 Open 是没用的特征，就给剔除了。

```
In [9]: train_data.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1017209 entries, 0 to 1017208
Data columns (total 9 columns):
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
dtypes: int64(7), object(2)
memory usage: 69.8+ MB
```

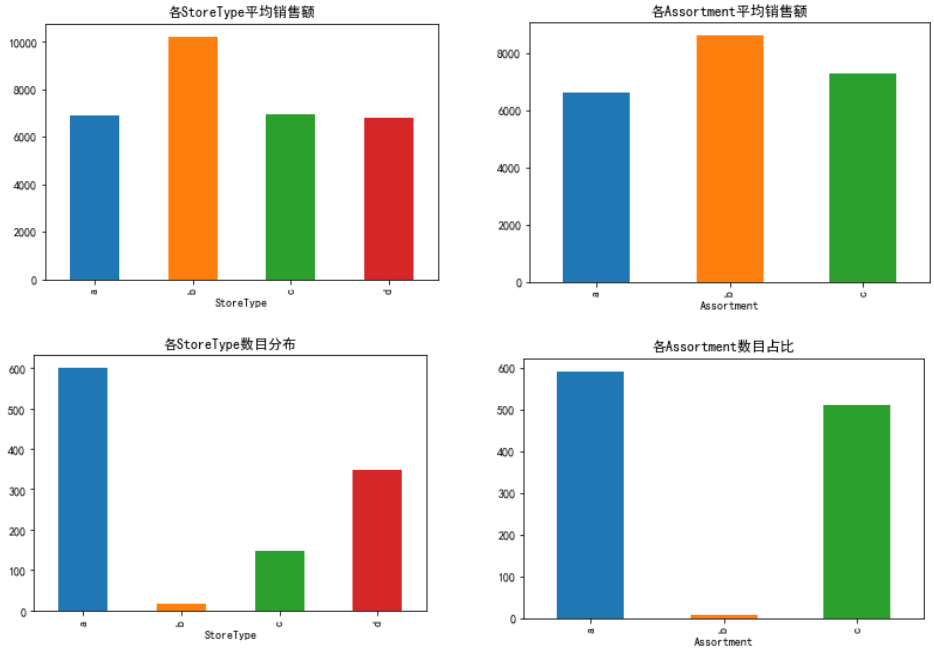
```
test_data.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 41088 entries, 0 to 41087
Data columns (total 8 columns):
Id               41088 non-null int64
Store            41088 non-null int64
DayOfWeek        41088 non-null int64
Date             41088 non-null object
Open             41077 non-null float64
Promo            41088 non-null int64
StateHoliday     41088 non-null object
SchoolHoliday    41088 non-null int64
dtypes: float64(1), int64(5), object(2)
memory usage: 2.5+ MB
```

探索性可视化

可视化是分析数据十分直观和方便的方式之一，为了更好的理解特征背后的含义，我们

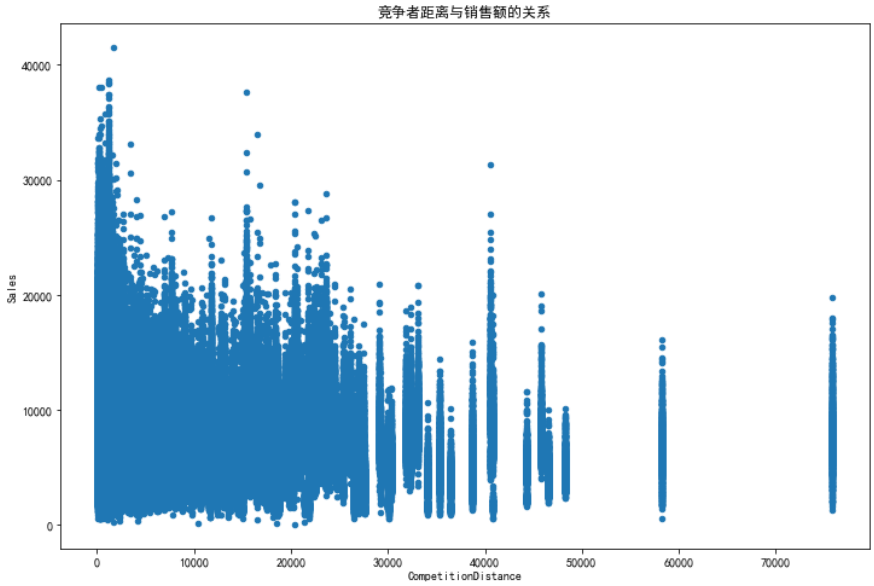
需要对数据进行一些分析。为了可视化分析，首先我们需要将 train 表和 Store 表进行合并，通过 Store 值进行关联，然后就可以进行分析了。

首先，查看一下 StoreType，Assortment 对于最终销售的影响，方法是查看每个类别的平均销量以及每个类别店铺的数目：



发现在这两个特征上，b 类型都意味着较高的平均收入，但是该类型都太少。可能这些店铺属于精品店或者旗舰店或者特殊位置的店铺，难以在其他地区推广。

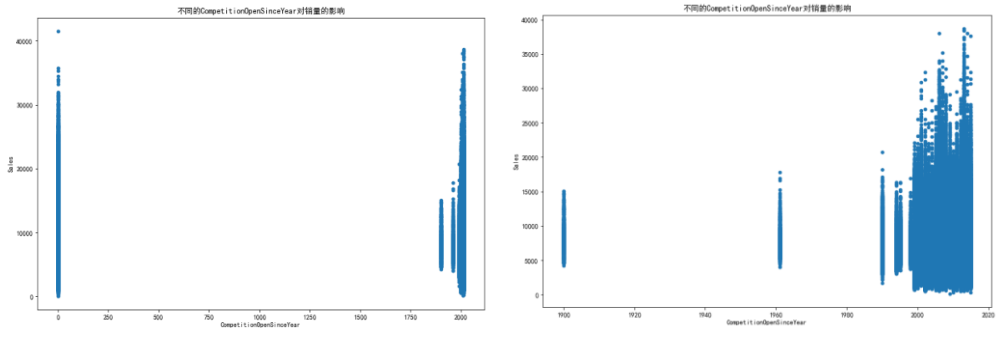
然后是竞争者的一些变量，首先是竞争者距离（CompetitionDistance），对于没有该特征的店铺，我用 0 代替，在排除 0 销售额的数据后，将不同距离的店铺的每日销售额做散点图，结果如下：



和我们常规的想法不同，在这个项目中，竞争者距离店铺近的情况更多，且两个店铺挨的比较近的话，反而会促进店铺的销售，这可能是因为店铺聚集在一起，形成了一定规模，促进了消费者进店消费。

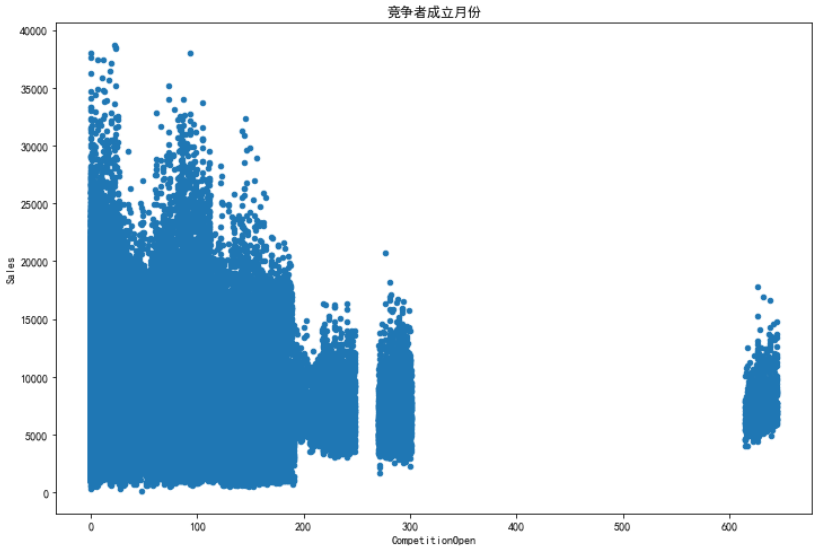
然后是 CompetitionSinceYear 这个值，由于有缺失值，为此将缺失值填为 0，得到如下

图：



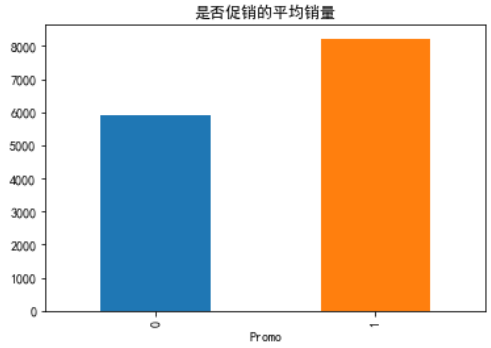
左图是将 CompetitionSinceYear=0 的店铺纳入的数据，右边是将 CompetitionSinceYear=0 的店铺排除的数据，可以看到即使排除了数据，该特征仍然很不直观。

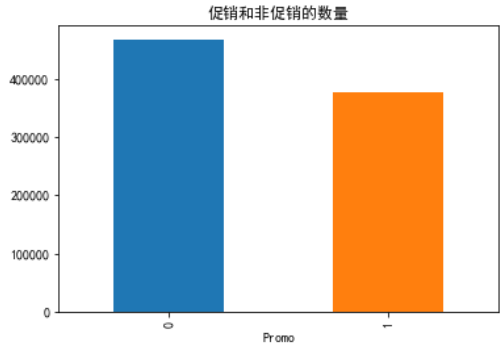
因而此处我构建了一个特征，即店铺成立多少个月，用这边特征和销量进行分析。得到下图：



可以看到，竞争者成立的时间越长，那么店铺的销售额的上限就越低，反之，则上限越高。这说明竞争者成立时间的长短能够影响到两个店铺的竞争优劣。

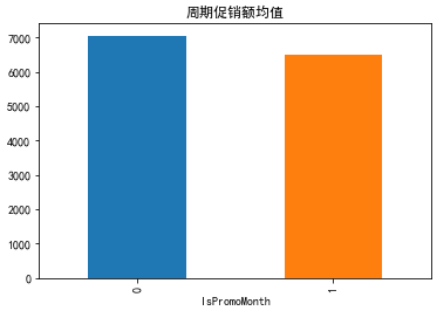
然后，分析一下促销活动对销量的影响。同样，需要排除销量为 0 的数据。先看 Promo 的情况，Promo 可以我理解就是临时促销，它不是周期性的，

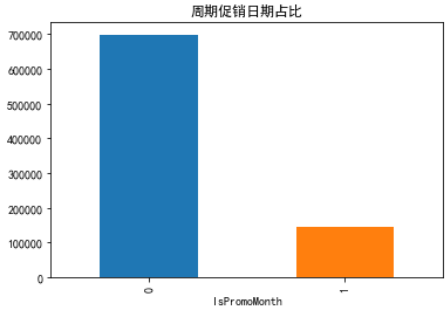




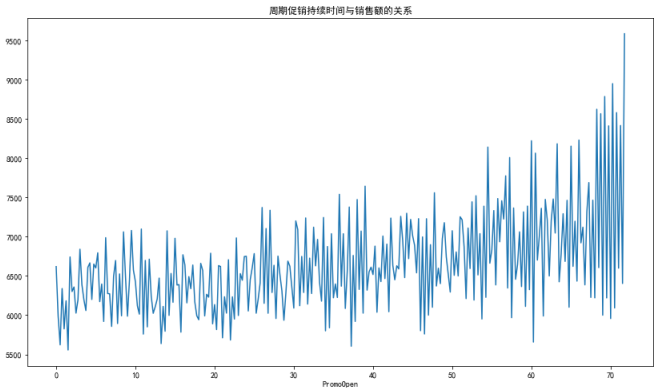
0 代表了没有促销，1 代表了促销，可以看到，促销对销售额的影响是非常显著的，平均销量高出 2000 左右。而且，促销的天数非常的多，大概 40%左右。

然后看 Promo2，这个是周期性的，他没有直接告诉你哪天后面有关于促销开始年分和月份，那么需要对其进行处理，获得其某一天是否进入 Promo2，同时我还增加了一个特征，即这个促销活动持续了多久。



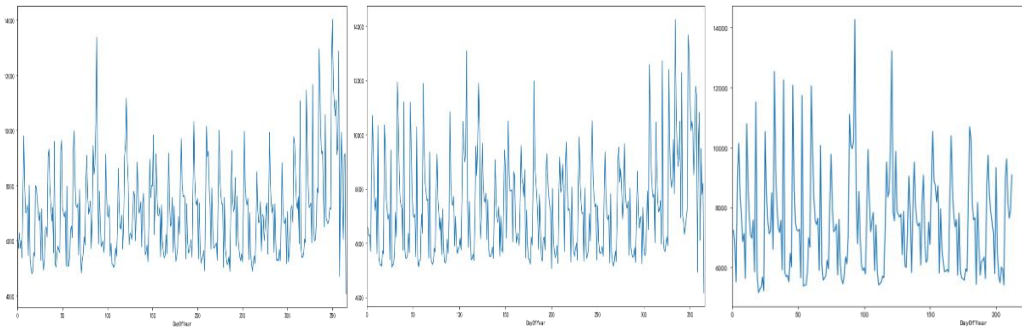


从促销日期来看，当日是否在周期促销中对销售额的影响很小，而且周期促销在所有日期中占比数量有限。

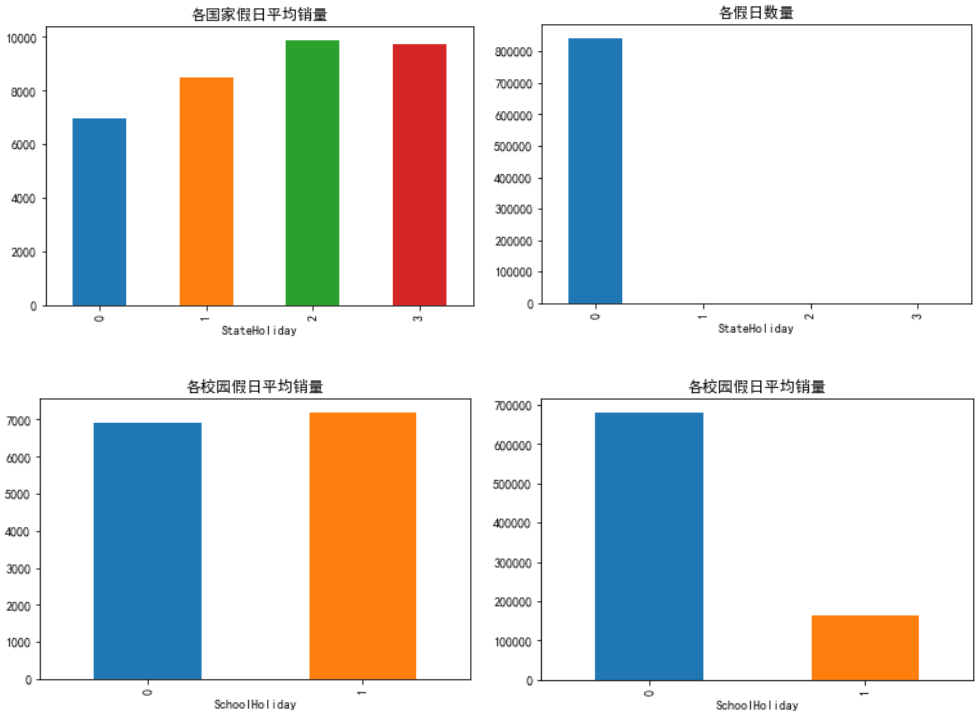


但是促销持续时间变量与销量相关性较高，即周期促销持续的越久，那么销量的均值约高。

然后查看日期与销售额的关系，首先需要将日期转化为数字，代表当年的第几天，然后观察其与销售额的关系。此图的纵轴为平均销售额，横轴为当年的某一天。3 幅图分别代表了 3 个年份。



如图可知，销售额是一个与日期强相关的数据，Rossman 店铺的销售有很强的相关性。



最后是 StateHoliday 和 SchoolHoliday2 个特征，结论是 StateHoliday 对销量的影响很大，但是国家假日数量太小了。SchoolHoliday 对销量的影响不大，但是数量会相对多一点。

算法和技术

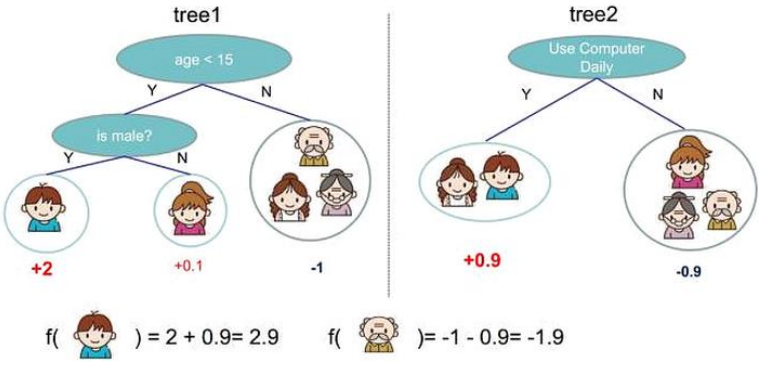
根据该项目获奖者的介绍，本项目我将使用 XGBoost 作为基准模型，XGBoost 是经过优化的分布式梯度提升库，旨在高效，灵活且可移植。它在 Gradient Boosting 框架下实现了机器学习算法。其特点是速度快，模型表现较好。

XGBoost 实现的是一种通用的 Tree Boosting 算法，此算法的一个代表为梯度提升决策树(Gradient Boosting Decision Tree, GBDT)，又名 MART(Multiple Additive Regression Tree)。

GBDT 的原理是，首先使用训练集和样本真值（即标准答案）训练一棵树，然后使用这棵树预测训练集，得到每个样本的预测值，由于预测值与真值存在偏差，所以二者相减可以得到“残差”。接下来训练第二棵树，此时不再使用真值，而是使用残差作为标准答案。两棵树训练完成后，可以再次得到每个样本的残差，然后进一步训练第三棵树，以此类推。树的总棵数可以人为指定，也可以监控某些指标（例如验证集上的误差）来停止训练。

在预测新样本时，每棵树都会有一个输出值，将这些输出值相加，即得到样本最终的预测值。

使用两棵树来预测一个人是否喜欢电脑游戏的示意图如下，小男孩和老人的预测值为两棵树预测值的加和。



XGBoost 所做的改进

2.1. 损失函数从平方损失推广到二阶可导的损失

GBDT 的核心在于后面的树拟合的是前面预测值的残差，这样可以一步步逼近真值。然而，之所以拟合残差可以逼近到真值，是因为使用了平方损失作为损失函数，公式如下

$$l(y_i, \hat{y}_i) = (y_i - \hat{y}_i)^2$$

如果换成是其他损失函数，使用残差将不再能够保证逼近真值。XGBoost 的方法是，将损失函数做泰勒展开到第二阶，使用前两阶作为改进的残差。可以证明，传统 GBDT 使用的残差是泰勒展开到一阶的结果，因此，GBDT 是 XGBoost 的一个特例。

2.2. 加入了正则化项

正则化方法是数学中用来解决不适定问题的一种方法，后来被引入机器学习领域。通俗的讲，

正则化是为了限制模型的复杂度的。模型越复杂，就越有可能“记住”训练数据，导致训练误差达到很低，而测试误差却很高，也就是发生了“过拟合”。在机器学习领域，正则化项大多以惩罚函数的形式存在于目标函数中，也就是在训练时，不仅只顾最小化误差，同时模型复杂度也不能太高。

在决策树中，模型复杂度体现在树的深度上。XGBoost 使用了一种替代指标，即叶子节点的个数。此外，与许多其他机器学习模型一样，XGBoost 也加入了 L2 正则项，来平滑各叶子节点的预测值。

2.3. 支持列抽样

列抽样是指，训练每棵树时，不是使用所有特征，而是从中抽取一部分来训练这棵树。这种方法原本是用在随机森林中的，经过试验，使用在 GBDT 中同样有助于效果的提升。

XGBoost 模型主要参数：

1、eta[默认 0.3]

和 GBM 中的 learning rate 参数类似。通过减少每一步的权重，可以提高模型的鲁棒性。

2、min_child_weight[默认 1]

决定最小叶子节点样本权重和。和 GBM 的 min_child_leaf 参数类似，但不完全一样。XGBoost 的这个参数是最小样本权重的和，而 GBM 参数是最小样本总数。这个参数用于避免过拟合。当它的值较大时，可以避免模型学习到局部的特殊样本。但是如果这个值过高，会导致欠拟合。这个参数需要使用 CV 来调整。

3、max_depth[默认 6]

和 GBM 中的参数相同，这个值为树的最大深度。这个值也是用来避免过拟合的。max_depth 越大，模型会学到更具体更局部的样本。需要使用 CV 函数来进行调优。

5、gamma[默认 0]

在节点分裂时，只有分裂后损失函数的值下降了，才会分裂这个节点。Gamma 指定了节点分裂所需的最小损失函数下降值。这个参数的值越大，算法越保守。这个参数的值和损失函数息息相关，所以是需要调整的。

6、max_delta_step[默认 0]

这参数限制每棵树权重改变的最大步长。如果这个参数的值为 0，那就意味着没有约束。如果它被赋予了某个正值，那么它会让这个算法更加保守。通常，这个参数不需要设置。但是当各类别的样本十分不平衡时，它对逻辑回归是很有帮助的。

这个参数一般用不到，但是你可以挖掘出来它更多的用处。

7、subsample[默认 1]

和 GBM 中的 subsample 参数一模一样。这个参数控制对于每棵树，随机采样的比例，减小这个参数的值，算法会更加保守，避免过拟合。但是，如果这个值设置得过小，它可能会导致欠拟合。

8、colsample_bytree[默认 1]

和 GBM 里面的 max_features 参数类似。用来控制每棵随机采样的列数的占比(每一列是一个特征)。

9、colsample_bylevel[默认 1]

用来控制树的每一级的每一次分裂，对列数的采样的占比。我个人一般不太用这个参数，

因为 subsample 参数和 colsample_bytree 参数可以起到相同的作用。但是如果感兴趣，可以挖掘这个参数更多的用处。

10、lambda[默认 1]

权重的 L2 正则化项。(和 Ridge regression 类似)。这个参数是用来控制 XGBoost 的正则化部分的。虽然大部分数据科学家很少用到这个参数，但是这个参数在减少过拟合上还是可以挖掘出更多用处的。

11、alpha[默认 1]

权重的 L1 正则化项。(和 Lasso regression 类似)。可以应用在很高维度的情况下，使得算法的速度更快。

12、scale_pos_weight[默认 1]

在各类别样本十分不平衡时，把这个参数设定为一个正值，可以使算法更快收敛。

基准模型

基准模型我选择随机森林作为参照，用来评判 XGBoost 模型的效果。

```
In [46]: from sklearn.ensemble import  RandomForestRegressor as rfr
         rfr_model=rfr(n_estimators = 100,
                       max_depth =10,
                       min_samples_split =20
                      )
         rfr_model.fit(X_train,y_train)
         rfr_pred=rfr_model.predict(X_val)
         print('随机森林的预测准确率为：',rmspe(y_val,rfr_pred))
```

```
随机森林的预测准确率为： 0.22005568995148558
```

本次项目的评估指标由 Kaggle 给出，为均方根百分比误差(RMSPE)，其计算公式如下图：

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

RMSPE 与 均方误差 RMSE 相比，区别在于在计算误差时要除以真实值 yi，我认为这样的好处当 yi 的取值范围很大时，比如最大值与最小值相差几十万倍，那么当高值预测出现误差时，产生的误差就会非常大，从而影响到了对其他值预测误差的评估。 通过将误差除以 yi，相当于在归一化这些误差，这样能够平等对待较大值与较小值。

随机森林测得的 RMSPE 大概在 0.22 左右。

III. 方法

数据预处理

Train 数据集中有几个变量采用的是枚举值，包括 StateHoliday，StoreType，Assortment。对这 3 个数据，我首先采用了 One-Hot 编码，但是事后发现被拆分出来的特征，显著性基本排在靠后位置，后来直接采用自然数代替。代码如下：

```
def stateholiday(data):
    Holiday={'0':0,'a':1,'b':2,'c':3}
    data['StateHoliday'] = data.StateHoliday.map(Holiday)
```

```
stateholiday(train_data)
```

```
def storetype(data):
    storetype={'0':0,'a':1,'b':2,'c':3,'d':4}
    data['StoreType'] = data.StoreType.map(storetype)
```

```
def assortment(data):
    assortment={'0':0,'a':1,'b':2,'c':3,'d':4}
    data['Assortment'] = data.Assortment.map(assortment)
```

```
storetype(train_data)
assortment(train_data)
```

然后，将日期中的年、月、日单独拆分出来

```
train_data['Month']=train_data.Date.map(lambda x :int(x.strftime('%m')))
train_data['Week']=train_data.Date.map(lambda x :int(x.strftime('%W')))
train_data['Year']=train_data.Date.map(lambda x :int(x.strftime('%Y')))
```

完成上述操作后，再回顾一下全体数据，发现部分特征的格式是浮点值：

```
In [173]: train_data.info()

<class 'pandas.core.frame.DataFrame'>
Int64Index: 1017209 entries, 0 to 1017208
Data columns (total 29 columns):
Store                        1017209 non-null int64
DayOfWeek                    1017209 non-null int64
Date                         1017209 non-null datetime64[ns]
Sales                        1017209 non-null int64
Customers                    1017209 non-null int64
Open                         1017209 non-null int64
Promo                        1017209 non-null int64
StateHoliday                 1017209 non-null int64
SchoolHoliday                1017209 non-null int64
StoreType                    1017209 non-null int64
Assortment                   1017209 non-null int64
CompetitionDistance          1017209 non-null float64
CompetitionOpenSinceMonth    1017209 non-null float64
CompetitionOpenSinceYear     1017209 non-null float64
Promo2                       1017209 non-null int64
Promo2SinceWeek              1017209 non-null float64
Promo2SinceYear              1017209 non-null float64
PromoInterval                1017209 non-null object
shopavgopen                  1017209 non-null float64
shopavgsalespercustomer      1017209 non-null float64
DayOfYear                    1017209 non-null int64
Month                        1017209 non-null int64
Week                         1017209 non-null int64
Year                         1017209 non-null int64
CompetitionOpen              1017209 non-null float64
monthStr                     1017209 non-null object
IsPromoMonth                 1017209 non-null int64
PromoOpen                    1017209 non-null float64
avgcustomer                  1017209 non-null float64
dtypes: datetime64[ns](1), float64(10), int64(16), object(2)
memory usage: 232.8+ MB
```

将其转换成整数：

```
F=['CompetitionDistance'
   ,'CompetitionOpenSinceMonth'
   ,'CompetitionOpenSinceYear'
   ,'Promo2SinceWeek'
   ,'Promo2SinceYear'
   ,'CompetitionOpen']
```

```
train_data[F]=train_data[F].apply(pd.to_numeric,downcast='integer')
```

接下来需要删除不需要的特征，并且将 train 的数据划分一部分出来用作验证：

```
#排除不营业的数据
train_data=train_data[(train_data['Open']==1)&(train_data['Sales']>0)]
```

```
#丢弃不需要的变量
drop_paramater=['Month','Week','Year','Date','Customers','Open','PromoInterval','monthStr']
train=train_data.drop(['Date','Customers','Open','PromoInterval','monthStr'],axis=1)
test=test_data.drop(['Date','Open','PromoInterval','monthStr'],axis=1)
```

将train数据集分为训练集和测试集

```
y_train=train['Sales']
X_train=train.drop('Sales',axis=1)
```

```
from sklearn.model_selection import train_test_split
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.1, random_state=0)
```

完成这些操作，接下来可以进行模型训练了。

执行过程

首先设置一个初始的参数进行训练，先开始第一轮的训练，训练结果如下：

```
XGBoost Model Valid Start....
Valid RMSPE:0.11785026135483434
XGBoost Model Valid End, Time: 5.344712 s....
```

感觉结果非常的不错，明显比随机森林要好上许多。然后上传 Kaggle 进行了测试，结果如下：

sample_submission.csv 3 days ago by Darren.wang add submission details	0.12948	0.12214

离预期目标差一点，但是感觉

完善

在调整参数的刚开始，我认为模型已经足够复杂，因此在参数的调整上，着重于防止其过拟合。结果往往是训练的效果较号，但是提交给 Kaggle 上就得到比较差的效果。

sample_submission.csv 2 days ago by Darren.wang add submission details	0.51188	0.50578

通过多次尝试，发现模型并不是过拟合，而是欠拟合了。因此，我增加了模型的复杂度，最终得到了比较好的效果，最终的参数如下：

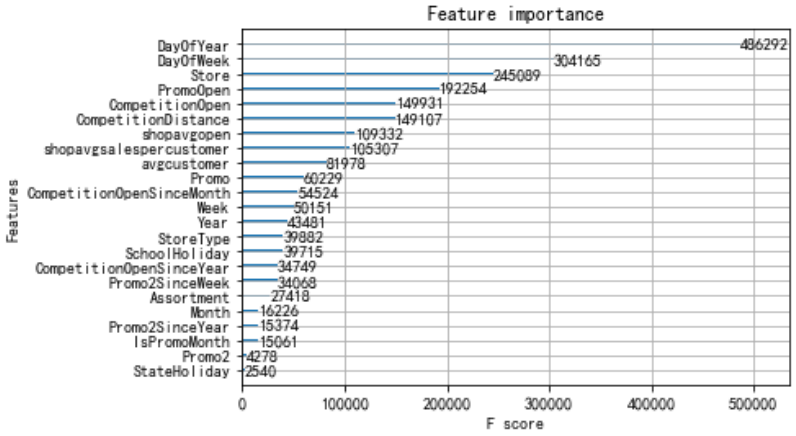
```
xgboost_tree = xgb.XGBRegressor(
#    n_jobs = -1,
    n_estimators = 6000,
    eta = 0.03,
    max_depth = 12,
#    min_child_weight = 9,
    subsample = 0.9,
    colsample_bytree = 0.9,
    silent = 0,
#    gamma=.7,
    random_state = 66,
    booster='gbtree',
#    reg_lambda=15,
    tree_method= 'gpu_hist',
    objective='reg:linear'
)
xgboost_tree.fit(X_train, np.log1p(y_train),
                 eval_set = [(X_train, np.log1p(y_train)), (X_val, np.log1p(y_val))],
                 eval_metric = rmspe_xg,
                 early_stopping_rounds = 100
                )
```

在训练集上，得到了如下效果：

```
In  [192]: rmspe(y_val, np.expm1(xgboost_tree.predict(X_val)))
Out[192]: 0.10926603952528588
```

各特征的有效性如下：

```
In  [106]: xgb.plot_importance(xgboost_tree)
Out[106]: <matplotlib.axes._subplots.AxesSubplot at 0x1eb1f616780>
```



将 submission 文件上传到 Kaggle 后，得到的结果如下：

Submission and Description	Private Score	Public Score	Use for Final Score
sample_submission.csv an hour ago by Darren.wang add submission details	0.12071	0.11725	☐

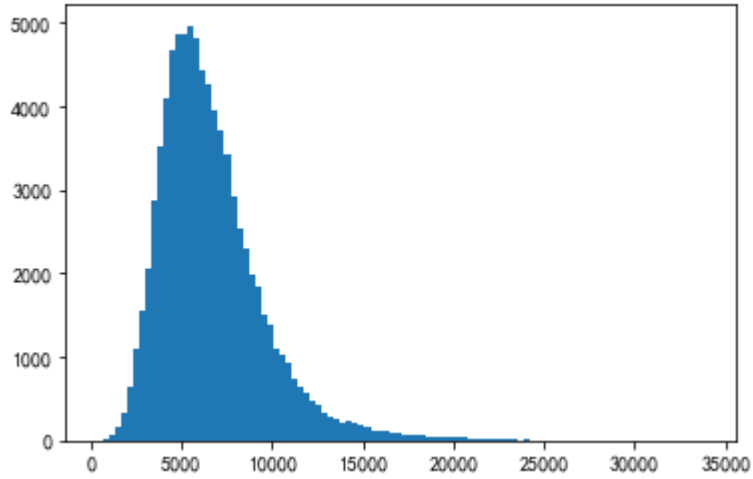
达到了 10%的目标。

IV. 结果

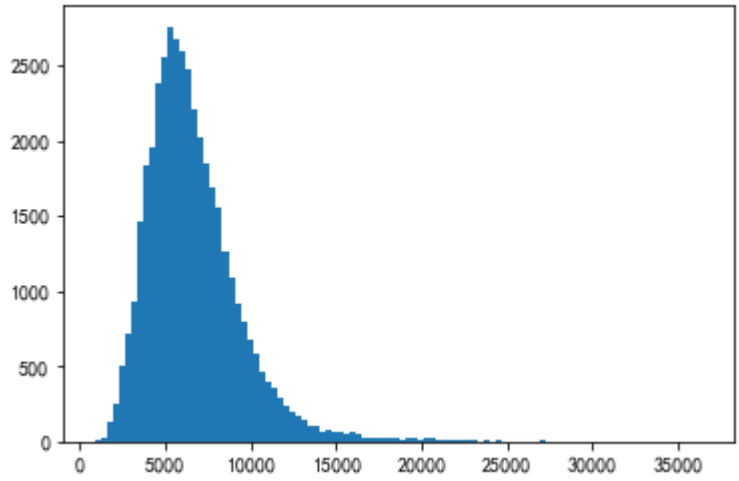
模型的评价与验证

为了校验分析的结论是否符合现实，需要对这部分数据进行交叉验证。首先，观察到模

型最有效的变量就是 DayOfYear，然后，test 数据的该特征全部包含在 213 于 260 之间。因此，我们查看了该区间销售额的分布情况，如下图：



然后查看我们预测数据的销售额分布情况，如下图：



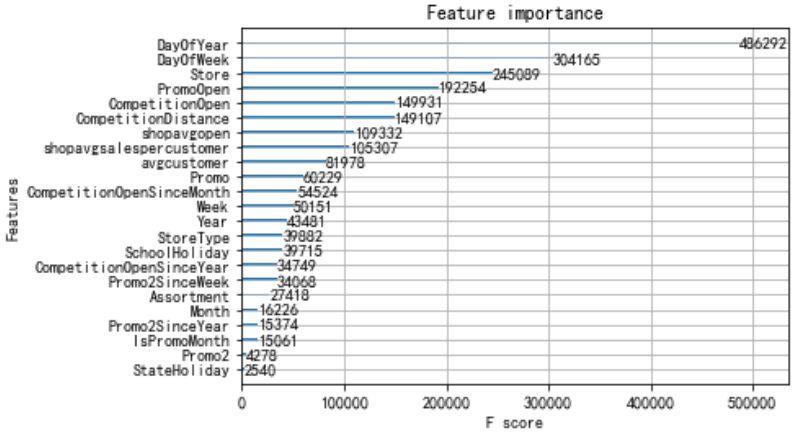
上面 2 图，横轴代表销售额，纵轴代表天数。可以看到，两者形状几乎相同，因此，我认为我做出的预测是合理的。

V. 项目结论

结果可视化

在这个部分，我还是要采用特征有效性的图做说明，即我认为，在这个项目中，特征工程真的是非常重要的，对结果影响较大的变量很多都要通过特征工程得到，而特征工程又是建立在前期的数据探索和数据可视化之上的。因此，我认为这些步骤看似繁琐，但对结果确具有决定性的作用，因此该步骤是不能省略和忽视的。

```
In  [106]: xgb.plot_importance(xgboost_tree)
Out[106]: <matplotlib.axes._subplots.AxesSubplot at 0x1eb1f616780>
```



对项目的思考

我认为最有意思的地方就是调参了，在这个过程中，我陷入了一个怪圈，就是我在训练集上的模型表现都非常的好，但是以到上传给 Kaggle 的时候，表现就很差，根据学习到的知识，我一直认为这是过拟合的表现，所以调参也是一直往防止过拟合的方向弄。所以，最后总结下来，就是调参有的时候可以多试试不同的组合，不用困死在一个圈子中。

需要作出的改进

其实在开始项目前浏览其他人的讨论的时候，大家普遍添加了天气数据用于分析，我认为这是合理的，如果时间充裕又能找到数据的话，我认为天气数据也是可以纳入进来的。那么拓展开来说，数据没必要局限于对方提供，所有有用的数据都可以考虑添加进来。