机器学习（进阶）纳米学位

何龙
2018 年 10 月 1 日

I. 问题的定义

项目概述

该问题属于典型的有监督回归问题，通过商店提供的历史数据，对未来的商店销售额进行估计，销售额是一个数值型数据，而提供的训练数据中也提供了销售额数据，因此属于有监督的回归问题。目前拥有的相关数据分三部分：train.csv 表示用于训练的数据，test.csv 表示用于最终测试的数据，store.csv 表示 1115 家商店的数据。问题背景很简单，来源于欧洲 Rossmann 公司提供过去的数据，需求也很简单，就是对其下的商店进行销售额预测，以辅助商店经理对于商店的促销活动、人员时间表等提供一个更加合理的安排。

问题陈述

问题定义：通过结合训练数据、商店数据，处理数据，训练模型，并通过模型来最大程度的预测测试数据。数据中包含有目标数据，因此是有监督的学习任务，另外目标值类型为数值型，因此是回归问题，针对这个有监督的回归问题，我选择的模型是 XGBoost。

任务大纲：1.读取并预处理数据，2.特征工程，3.模型训练、测试，4.对比基准阈值，5.模型优化，6.提交至 kaggle。

期望结果：模型预测测试数据的误差小于 20%。

评价指标

由于该问题需要提交至 kaggle，因此评价指标跟 kaggle 采用一致的最为合理，也就是 RMSPE（均方根百分误差），它的作用是评估所有预测数据与实际数值的平均误差的百分比形式，与 RMSE 相比引入了百分比的关系，避免了较小的误差被忽略。

公式：sqrt((sum(X_obs_i-X_model_i/X_obs_i)^2)/n)。

因为我们的目的是最小化预测值与实际值之间的差距，因此使用该评价指标是合理的，且该公式会避免出现负数，以免出现正负值中和的问题。

II. 分析

数据的探索

因为我们对数据的使用主要集中与训练阶段，因此我们主要关注、分析、挖掘的也是 train.csv 和 store.csv 文件，对于 train.csv，具有以下字段：

1. *Store*：表示商店编号。
2. *DayOfWeek*：表示星期几。
3. *Date*：表示日期，由年-月-日组成。
4. *Sales*：目标字段，表示销售额。
5. *Customers*：表示当天客户数量。
6. *Open*：表示是否开门。
7. *Promo*：表示是否有促销活动。
8. *StateHoliday*：表示国家假期。
9. *SchoolHoliday*：表示学校假期。

再来看看 store.csv 的字段：

1. *Store*：同上述。
2. *StoreType*：商店类型。
3. *Assortment*：分类级别。
4. *CompetitionDistance*：最近的竞争对手距离。
5. *CompetitionOpenSinceYear*：竞争对手开张年。
6. *CompetitionOpenSinceMonth*：竞争对手开张月。
7. *Promo2*：是否参与持续促销活动。
8. *Promo2SinceYear*：促销活动开始的年。
9. *Promo2SinceWeek*：促销活动开始的星期。
10. *PromoInterval*：促销活动的周期月份。

Train.csv 的基本信息：

```
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 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 object
SchoolHoliday    1017209 non-null int64
```

Store.csv 的基本信息：

```
RangeIndex: 1115 entries, 0 to 1114
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                1115 non-null object
```

首先，最直观的感觉是两份数据可以通过 Store 字段链接到一起。其次在 train.csv 中没有 null 值，而 store.csv 中的 CompetitionDistance、 CompetitionOpenSinceYear、 CompetitionOpenSinceMonth 存在 null 值，且容易看到他们的缺失应该是技术原因导致的缺失，也就是说我们需要对其进行填充处理的，由于目前看不出其与其他字段的直接关系，因此填充方式选择 median 值填充。而对于 Promo2SinceYear、 Promo2SinceWeek 的缺失则不同，我们发现在他们缺失的数据中，都存在 Promo2 为 0 的情况，也就是说他们并不是技术原因导致缺失，而是由于商店不参与活动导致没有活动相关数据，因此不需要特别处理。

再来看一些有意思的统计数据：通常我们会认为商店的销售额跟假期应该有关系，那么事实是怎么样的呢？在学校是否放假的条件下，分别对应的销售额为：

学校放假平均销售额：*6476.52220712*

学校不放假平均销售额：*5620.97903381*

可以看到在学校放假或者不放假的情况下，销售额没有很明显的差异，这一点说明至少从平均值上看，放不放假是没有影响或者说影响不大的，那么国家假期呢：

国家放假平均销售额：5779.7781799

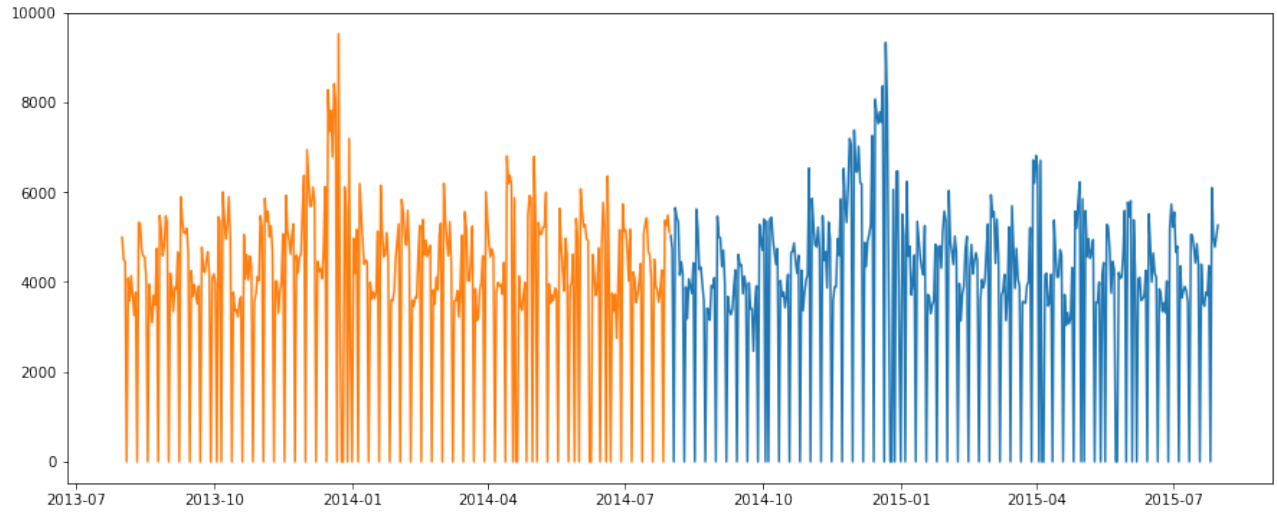
国家不放假平均销售额：5733.53062439

同样可以看到影响非常之小，这一点反应出来的跟我们的直觉可能不太一致。但是在数据中发现了一个非常有用的信息，那就是周末的销售额都是 0，也就是周末是不上班的，这一信息对于我们预测非常有帮助。而像

Date 信息、促销活动信息、竞争对手信息等都要进一步进行可视化验证其与我们预测值的相关性，综合分析才能判断是否需要挖掘更深层次的意义。

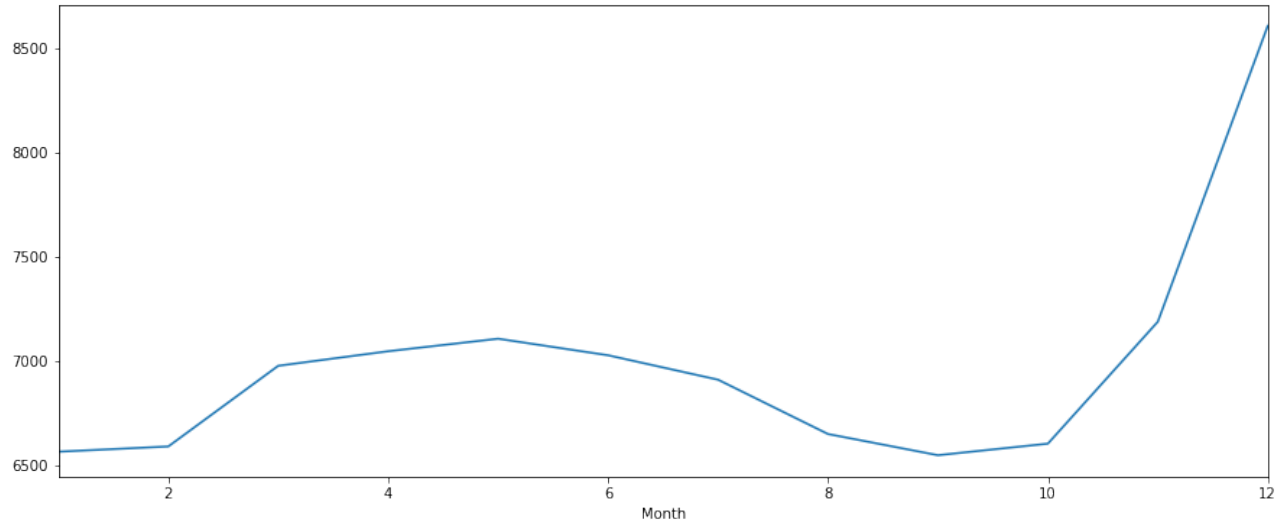
探索性可视化

1．时间信息探索：之所以挖掘时间信息，主要原因有以下几个，首先数据的时间是连续完整的，Date 字段的训练数据从 13 年 1 月 1 号到 15 年 7 月 31 号，大约两年半的连续时间，时间的完整保证了我们对时间的挖掘可以推广到所有数据上，首先我们看看相邻的两年时间下的销售额变化图：

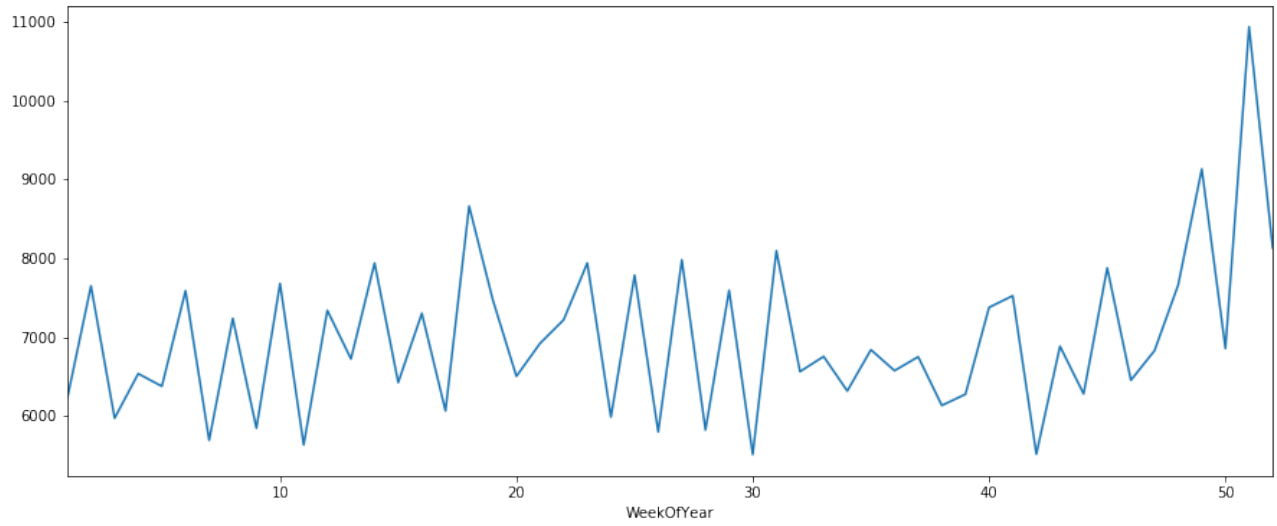


可以看到趋势惊人的一致，这说明了销售额是一个时间周期强相关的属性，因此对 Date 的挖掘一定要彻底，再来看看深层次一点的挖掘，即假期期间的销售额变化，注意图中上方的小短线表示假期的持续时间，对应下面的线表现期间的销售情况：

无法看到明显的趋势或相关性，这一点结合之前的学校假期、国家假期与平时的销售对比数据，说明假期对结果的影响很小，不需要特别挖掘处理，当然周日另算，下面看看挖掘后的月份、WeekOfYear 的销售额对比图，先看看月份的：

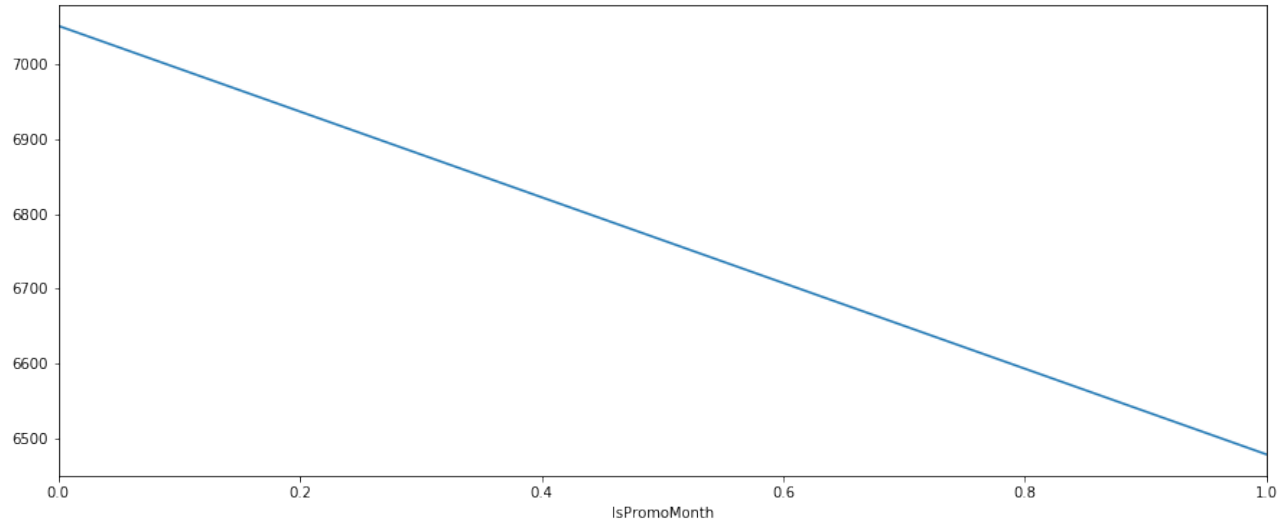


同样可以看出 7,11,12 月份明显高于其他月份，同时一年中，有 3 个销售额上升的情况出现，这一特点的原因不能肯定，但是应该与季节相关的可能性较大。再来看看 WeekOfYear 的：

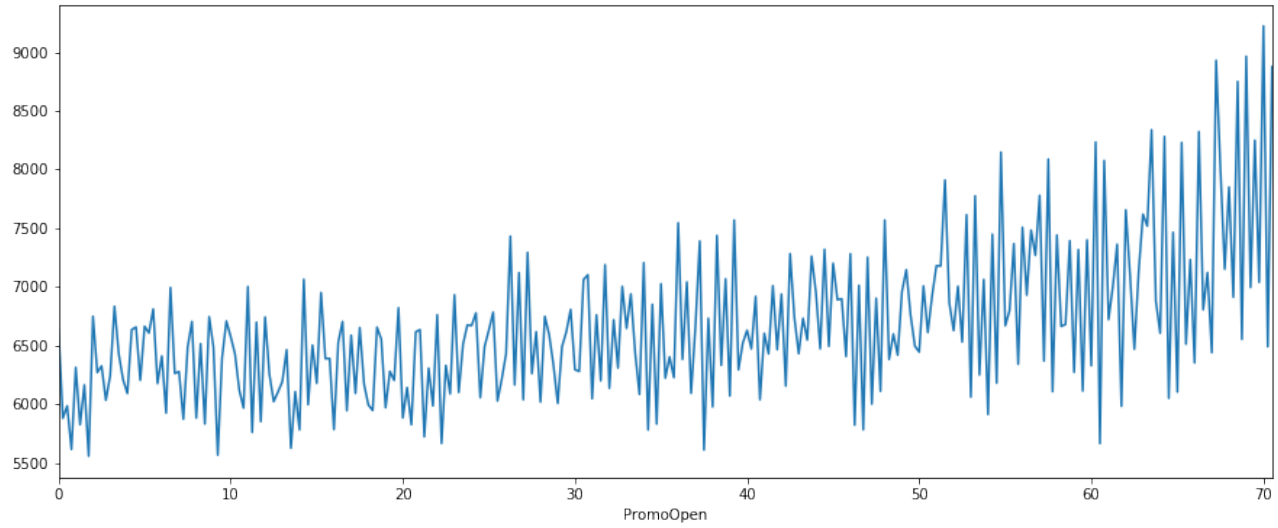


这个情况就不太好总结趋势，不过可以看做是对月份的一个细化表现，整体上跟月份上表现的趋势是一致的，只是将这一趋势细化到以星期为单位的数据上，最后我们可以肯定的是，时间字段对于结果影响非常大。

2．持续促销活动的探索：商店重要的促进销售的手段就是搞促销活动，因此我们没理由不对持续促销相关数据进行探索分析，先来看看是否处于促销月对于销售额的影响（PS：是否处于促销月是我们针对持续促销挖掘的新特征）：

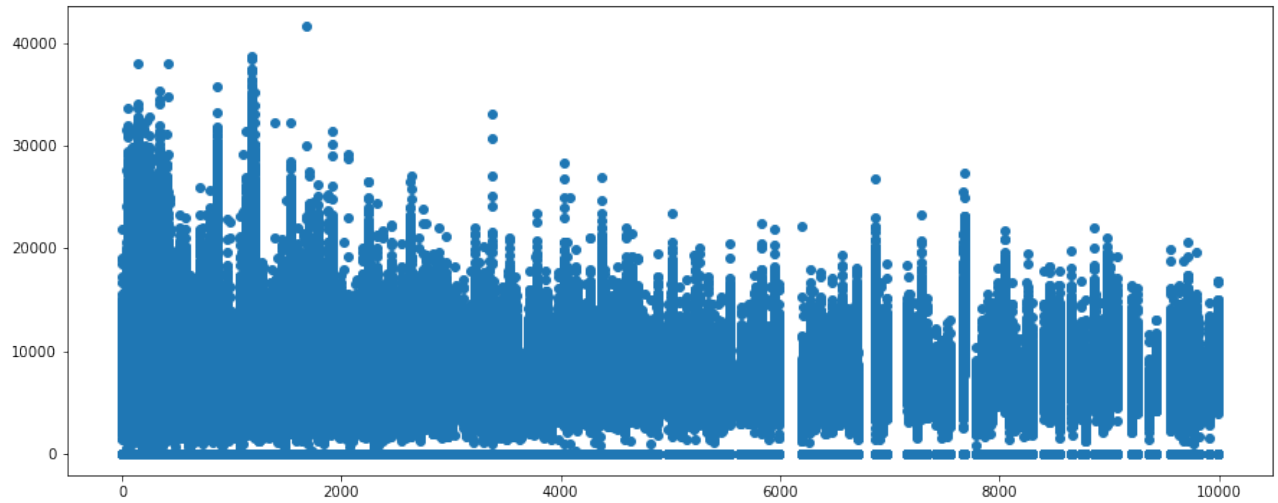


这就非常清楚的表达了持续促销对销售额的影响，下面再细化以下，看看促销时间跟销售额的影响：



这张图表示随着促销的持续，对销售额的影响是持续增高的，这个有可能是因为随着时间的推移，越来越多的客户知道这个商店在搞促销，因此也更热衷来这里购买商品。

3．竞争对手信息的探索：一家商店是否有竞争对手，以及两家店的距离，对方的营业时间都会对我们的销售额造成影响，这是很明显的道理，下面让我们可视化分析一下，开张营业时间的销售额对比图：

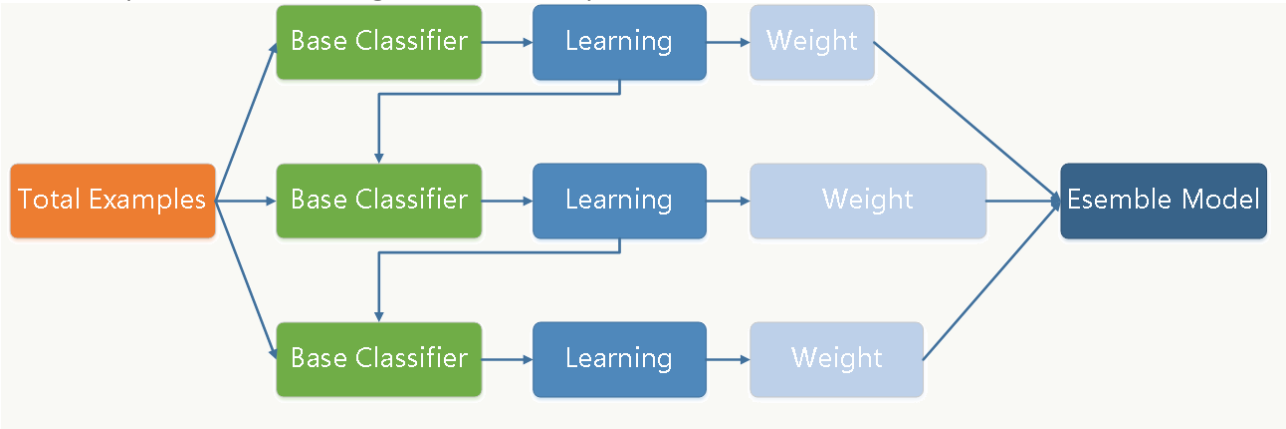


可以看到，对手的营业时间越短，我们的销售额越高，这一结果基本符合我们的预测，毕竟一家新店对我们销售额的影响肯定不如一家“老字号”。

算法和技术

采用 XGBoost [1] +校正系数 [2] +增量训练验证数据的组合来实现模型的训练以及优化。

XGBoost：继承自 GBDT 的一种 Boosting 算法，既然是基于 GDBT 的一种算法，那么我们就从 GBDT 说起。GBDT(Gradient Boosting Decision Tree)：



是一种迭代的决策树算法， 由多颗决策树组成，所有树生成的结论结合各自的权重相加得到最后的结果，简单说 GBDT 每颗决策树都利用前一颗树（因此是串行的）的残差进行学习，相比较使用一棵树完成整个学习过程，多棵树能够避免过拟合问题，而利用残差向量这一点保证了它是全局最优的，也就是 Gradient。GBDT 既适用于回归问题，同样可以用于分类问题（设置阈值即可）。而继承自 GBDT 的 XGBoost 可以说是一种全面强化的 GBDT，主要区别（也是 XGBoost 的优点）有以下几点：

1．基分类器的选择：传统 GBDT 以 CART 作为基分类器，XGBoost 还支持线性分类器（实际用的比较少，因为基于树的基分类器速度更快），这个时候 XGBoost 相当于带 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）。

2．二阶泰勒展开：传统 GBDT 在优化时只用到一阶导数信息，XGBoost 则对代价函数进行了二阶泰勒展开，同时用到了一阶和二阶导数。XGBoost 支持自定义损失函数，只要函数可一阶和二阶求导。

3．方差-方差权衡：XGBoost 在目标函数里加入了正则项，用于控制模型的复杂度。正则项里包含了树的叶子节点个数、每个叶子节点上输出分数的 L2 模的平方和。从 Bias-variance tradeoff 角度来讲，正则项降低了模型的 variance，使学习出来的模型更加简单，防止过拟合，这也是 XGBoost 优于传统 GBDT 的一个特性。

4．列抽样（column subsampling）：XGBoost 借鉴了随机森林的做法，支持列抽样，不仅能降低过拟合，还能减少计算，这也是 XGBoost 异于传统 GBDT 的一个特性。

5．缺失值处理：XGBoost 考虑了训练数据为稀疏值的情况，可以为缺失值或者指定的值指定分支的默认方向，这能大大提升算法的效率，paper 提到 50 倍。即对于特征的值有缺失的样本，XGBoost 可以自动学习出它的分裂方向。

6．XGBoost 工具支持并行：Boosting 不是一种串行的结构吗?怎么并行的？注意 XGBoost 的并行不是 tree 粒度的并行，XGBoost 也是一次迭代完才能进行下一次迭代的（第次迭代的损失函数里包含了前面次迭代的预测值）。XGBoost 的并行是在特征粒度上的。我们知道，决策树的学习最耗时的一个步骤就是对特征的值进行排序（因为要确定最佳分割点），XGBoost 在训练之前，预先对数据进行了排序，然后保存为 block(块)结构，后面的迭代中重复地使用这个结构，大大减小计算量。这个 block 结构也使得并行成为了可能，在进行节点的分裂时，需要计算每个特征的增益，最终选增益最大的那个特征去做分裂，那么各个特征的增益计算就可以开多线程进行。

7．可并行的近似直方图算法：树节点在进行分裂时，我们需要计算每个特征的每个分割点对应的增益，即用贪心法枚举所有可能的分割点。当数据无法一次载入内存或者在分布式情况下，贪心算法效率就会变得很低，所以 xgboost 还提出了一种可并行的近似直方图算法，用于高效地生成候选的分割点。大致的思想是根据百分位法列举几个可能成为分割点的候选者，然后从候选者中根据上面求分割点的公式计算找出最佳的分割点。

可以看到，相比较 **GBDT** 来说，**XGBoost** 在速度方面有很大的优化不论是支持并行，还是对缺失值的处理等，而另一方面就是增加了正则化、列抽样等来避免过拟合的手段，体现在实际使用中就是多了很多超参数可以调节。

下面是我最终选用 XGBoost 模型的一组超参数：

```
params_opt = {"objective": "reg:linear",
```

```
        "booster" : "gbtree",
        "eta": 0.03,
        "max_depth": 10,
        "subsample": 0.9,
        "colsample_bytree": 0.7,
        "silent": 1,
        "seed": 66
        }
    num_boost_round_opt = 6000
```

输出选择 reg:linear，符合我们的项目需求，booster 选用基于树模型的基分类器。eta、max_depth 为实验所得的经验值，num_boost_round 设置为 6000，后续训练中发现在 2899 次时达到最优（100 次不提升终止训练），在验证集的表示是 0.12637，已经是很好的结果了。early_stopping_rounds 设置为 100 保证了少量的连续迭代无法提升不会终止模型，但是在 100 次都没有提升的情况下会终止模型，避免模型过拟合，在验证集的性能降低的情况出现。

校正系数：这一做法的主要思想是，如果我们的预测数值相对于实际数值存在整体偏高或者偏低的情况时，我们可以通过校正系数来统一进行预测结果的校正来减小误差，试验中发现在校正系数为 0.996 时，RMSPE 值降低了大概 0.006，还是很不错的处理方式。

增量训练验证数据：保证了要预测的测试数据在时间上是紧挨着整个训练数据的（对于时序数据，这一点很重要），同时也是最大化的利用可用的数据进行训练（kaggle 上测试证明这种方式生效很困难，因为没有验证集能够辅助参数调整）。

基准模型

根据项目要求，设置基准阈值为 0.11773，即 kaggle 排名的前 10%。

III. 方法

数据预处理

数据预处理 0：类型转换，train 和 test 的 Date 字段使用 to_datetime 转换为 DateTime，方便后面可视化时的显示效果，store 的 PromoInterval 转换为 Str，虽然现实还是 Object，但是后续可以直接用 in 操作了，这两个转换都是为了后续一些操作上的遍历，并不是非转换不可。

数据预处理 1：空值填充，之前的数据探索中我们知道，test 的 open 字段、store 的 CompetitionOpenSinceMonth、CompetitionOpenSinceYear、CompetitionDistance、Promo2SinceYear、Promo2SinceWeek，下面我们依次描述对每个字段的填充，以及选择这种填充方式的原因：

1. Open 字段：由于如果是 close 的话，那么必然会预测为 0，因此此处填充均使用 1，表示营业，同样的数据中也可以看到 1 的数量远大于 0，从众数的角度也是 1 更合理。
2. CompetitionDistance：典型的数值型字段缺失，通常情况下在无法通过其他字段来推测时，我们直接采用 mean 值填充，是较为傻瓜合理的办法。
3. CompetitionOpenSinceYear、CompetitionOpenSinceMonth：同 2，但是有一点要注意，因为是年份和月份，因此最好取整数填充，方便后续如果想作为枚举型数据处理的需求。
4. Promo2SinceYear、Promo2SinceWeek：比较特别的一种缺失，它们两个的缺失并不是由于技术性、人为性错误导致数据收集上的问题，而是由于 Promo2 字段为 0 导致的，也就是当对应商店不参与持续促销活动时，这两个字段为空，那么这种情况我们是不能也不需要处理的，贸然处理会造成相反的效果，但是保持空值对算法貌似不太友好，不过后面我们的特征工程会处理它们。

数据预处理 2：数据拼接，train 和 store，test 和 store，目的是在一个整体的数据上结合 train 和 store 的所有特征进行分析、挖掘，实现 1+1 大于 2 的效果。

数据预处理 3：新特征的获取：

1. 时间相关：通过 Date 一个字段，挖掘出 Year、Quarter、Month、Day、WeekOfYear、IsWorkDay6 个字段。
2. 促销相关：根据 1 中挖掘的 Month 字段，通过 Promo2 判断商店是否参与了持续促销，如果参与了再判断当前数据的 Month 是否在 PromoInterval 的促销月中，挖掘出 IsInPromo 字段。而通过 IsInPromo 字段判断是否处于促销月，如果是，那么再通过 Day 字段获取该月已经持续的天数来挖掘出当前处于持续促销活动的第几天，即 PromoDays 字段。
3. 竞争对手相关：对于竞争对手相关字段，CompetitionOpenSinceYear、CompetitionOpenSinceMonth 两个字段并不是常规意义上的数值型字

段，关系由大小决定，而是时间点字段，这种字段直接使用的话很难保证实际效果，但是我们可以从这两个字段中挖掘出竞争对手针对于当前数据的总营业时间，单位为月，而这一字段明显是一个典型的字段关系由数值大小来表示的字段，即 CompetitionOpenMonths。

数据预处理 4：无用字段的抛弃，无用字段主要包括 Date、Customers、Open、PromoInterval、monthStr。

数据预处理 5：枚举性字段的整体映射，包括 StateHoliday、StoreType、Assortment，可以看到特征挖掘后，枚举字段也少了很多，那么对于这三个字段，由于其取值中有 a、b 等字符型数据，因此我们通过通过映射表{'0':0, 'a':1, 'b':2, 'c':3, 'd':4}将其映射至数字上。

数据预处理 6：提取目标字段，将 Sales 从 train 数据中提取出来，如果不提取，那么结果就是在训练集、验证集上表现完美的模型，在测试集上结果大打折扣，甚至因为字段的不匹配导致模型无法预测测试数据。

数据预处理 7：训练集、验证集的划分方式，由于数据是时序数据，因此采取按照时间先后顺序划分的方式，选取最后 6 周作为验证集数据，之所以选择 6 周是因为测试集数据就是 6 周的数据，验证集跟测试集数据越接近，起到的模型验证效果越好。

执行过程

模型执行过程如下：

1. 模型构建：指定最初的模型参数、迭代次数以及转换训练数据的格式为 xgboost 需要的 DMatrix。
2. 模型训练：利用 xgb 的 train 方法通过指定的参数使用指定的训练数据进行模型训练，返回一个 Booster 对象。
3. 使用 Booster 的 predict 进行验证集上的预测，并通过 RMSPE 方法计算分数。
4. 模型保存到磁盘，方便后续使用，毕竟训练一次都要 30min 左右。
5. 模型优化：
    1. 参数优化。
    2. 校正系数。
    3. 增量训练验证集数据。
6. 提交 kaggle。

在使用 XGBoost 库的过程中，最大的困难来源于对库的不了解以及网上资料的困乏，最大的信息源就是它的官方文档，庆幸的是官方文档的资料基本满足我们的需求。参数优化中，困难主要来自于对参数的调整，由于每次优化后训练都耗时很长，因此每次调整都要特外谨慎，在这个过程中也更加的了解了 XGBoost 的算法原理，以及各个参数对结果的影响。

完善

参数优化

模型参数优化情况如下图所示：

```
params_opt = {"objective": "reg:linear",
   "booster" : "gbtree",
   "eta": 0.03,
   "max_depth": 10,
   "subsample": 0.9,
   "colsample_bytree": 0.7,
   "silent": 1,
   "seed": 66
   }
num_boost_round_opt = 6000
```

该参数下的训练情况：

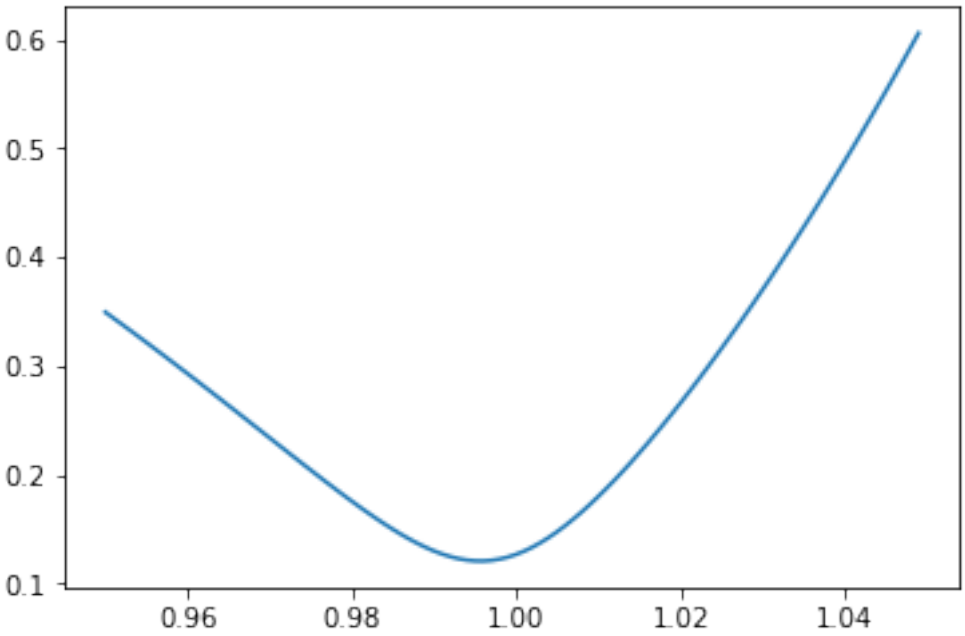
rmspe:train-rmspe:0.071787
valid-rmspe:0.126345
times:2872s
num_boost_round:2899

看到，train loss 低于 0.10，同时 valid loss 为 0.126，仅参数优化得到这个结果是很不错的了。

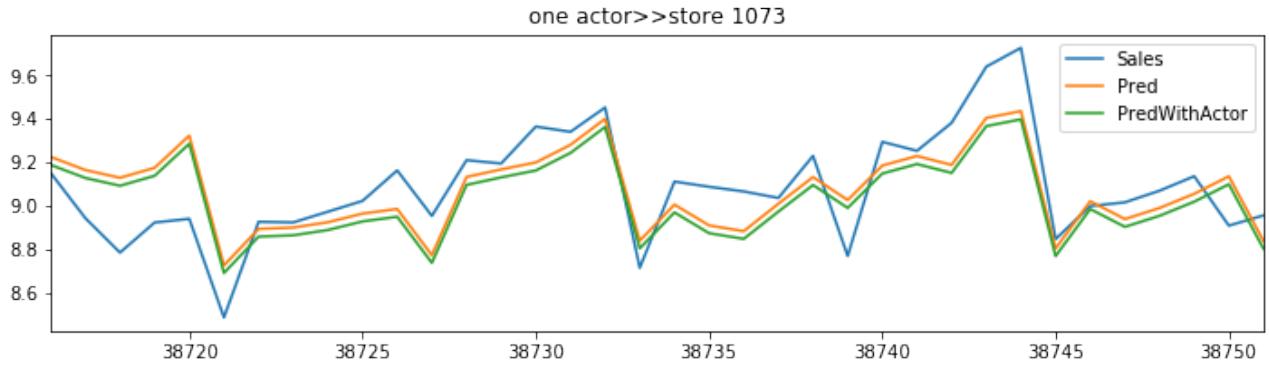
校正系数

下面我们应用校正系数，前面也提到，应用校正系数的前提是预测数据与实际数据有明显偏差，如图：

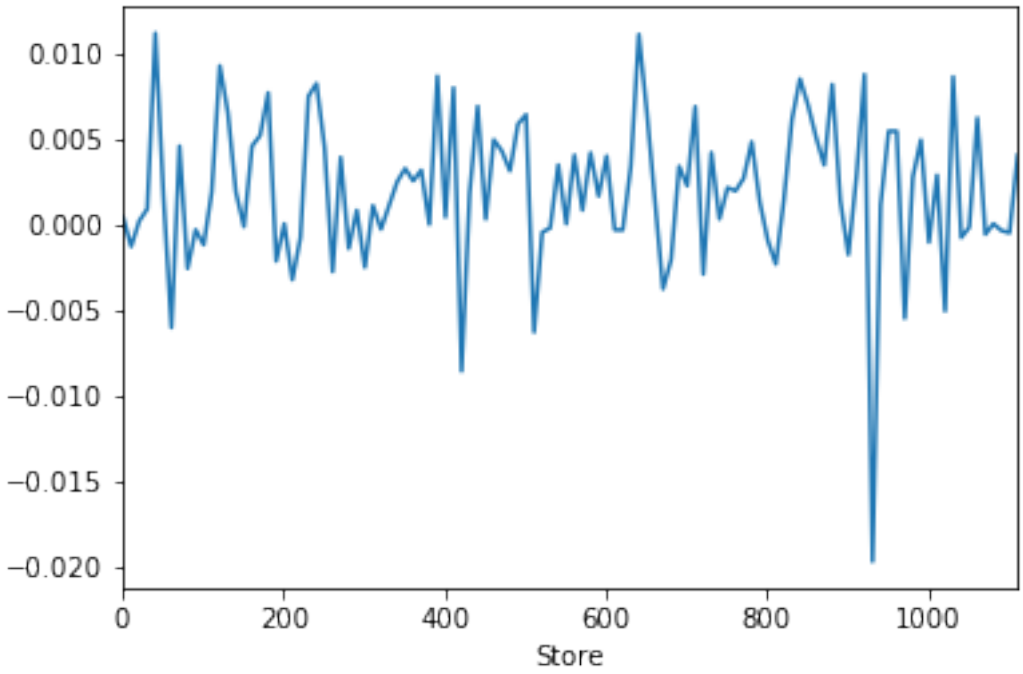
可以看到，直观感觉是整体偏高，数值上计算相对偏差为：`0.00218784928187`，正数也证明了我们的感觉是对的。通过赋予其一个校正系数来校正这一问题，如下：



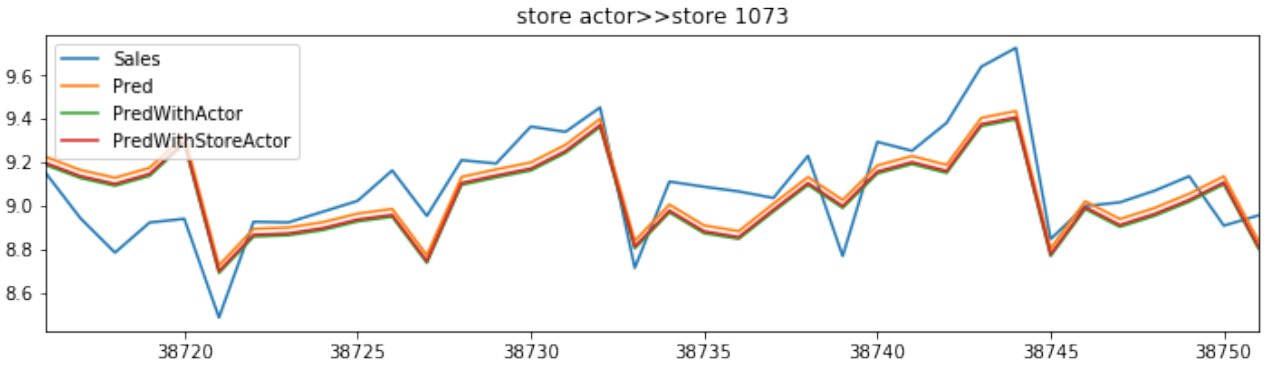
计算得到 0.996 为最佳校正系数，小于 1 也符合我们的数据整体偏高的看法，看看校正后效果：



可以看到相对于黄线（未校正），绿线（统一校正）校正后在验证集的表现：`(0.996, 0.12035940620005151)`，可以看到，校正后提升了大约 0.006，只是一个校正系数就起到这么大的作用，对于我们来说还是很不错的，毕竟几乎没代价。但是我们直到，商店之间的差异应该很大（经营原因、位置原因等等），因此这个校正系数是否需要针对每个 Store 来计算并设置呢，我们先看下多个商店的相对偏差，看下是否差异较大，只有差异大才有必要这么多：



可以看到，商店之间的差异还是很大的，这种情况统一使用一个校正系数是不准确的，因此我们赋予每个商店自己的校正系数，看看效果：



而分别校正后的 RMSPE 得分为 0.11099882652，提高了 0.009 左右，又是一个非常大的提升。

全数据训练

之前的所有训练都是通过划分数据集得到训练集和验证集来训练的，因为最终要放到 kaggle 去测试，因此最后一步优化我们将训练集+验证集统一作为训练数据去训练我们的模型，同时应用之前的所有优化手段。经过 kaggle 验证，使用全数据的模型在参数不变、校正系数不变的情况下，private score 为 0.1213，相比于之前的最佳得分 0.11188 升高了很多，因此不采用这一方式，主要原因在于没有验证集能够进行模型参数的选择，导致模型表现不佳。

IV. 结果

模型的评价与验证

XGBoost 模型通过数据训练进行有监督的回归学习，输入数据使用了 kaggle 提供的 train 和 store，输出为数值型的 Sale，满足项目需求，基准模型的 RMSPE：0.11773（kaggle 前 10%），模型达到了 0.11188，非常大的提升，因此所选的模型是合理可用的，与期待结果一致。

在将个别数据使用异常值替代后与正常输入数据对比，
1.原始输入

```
DayOfWeek                 5
Open                      1
Promo                     1
StateHoliday              0
SchoolHoliday             1
StoreType                 3
Assortment                1
CompetitionDistance    1270
Promo2                    0
Year                   2015
Quarter                   3
Month                     7
Day                      31
WeekOfYear               31
IsWorkDay              True
IsInPromo             False
PromoDays                 0
CompetitionOpenMonths    82
Name: 0, dtype: object
```

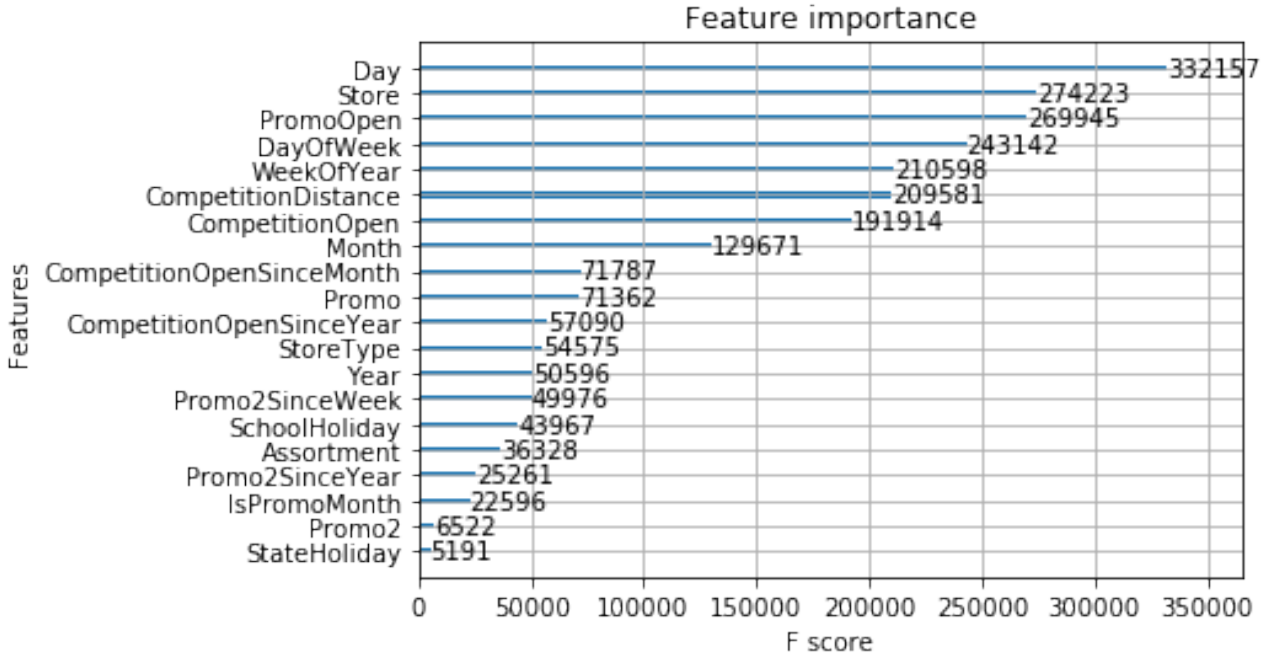
，输出：5510.4448，真实值：5262.9999，

2.修改 Year 为 0，输出：5763.11

3.修改 CompetitionDistance 为-100，输出：8143.42

4.修改 DayOfWeek 为 8，输出：5692.17

可以看到，在对输入数据进行了小范围的异常化后，对于输出的影响还是有的，影响的大小由对应字段对预测结果的重要程度而定，字段的重要性如下：



可以看到，对于预测结果而言，Day 是最为重要的字段，其次是 Store，Store 的重要性也佐证了我们针对每个 Store 单独校正的必要。

数据在训练集和验证集的划分是按照时间前后的，即取最后 n 周作为验证集数据，因为我们知道销售额是有周期性的，那么训练数据、验证数据、测试数据均有时间上的前后关系可以帮助我们利用这一周期性的特点，因此在验证集上的 RMSPE 得分是客观可信的，而在 kaggle 测试集上的误差得分要高于验证集的得分，其中最可能的原因是销售额数据本身是有时间前后关联性的，比如用去年的数据估计今年的销售额，误差会明显低于取前年的数据估计今年的销售额，因此我们用 13,14,15 前半年的数据同样预测这之间的某些时间的销售额，这个误差也要低于取预测 15 年后半年的销售额，这个结果是可以预见，并且合理的，对于模型来说，只要我们持续使用最新的数据去不断的训练它，那么它的预测结果就能一直保持一个很低的误差率，因此我们的模型是合理的可靠的。

合理性分析

基准模型在 Kaggle 测试集上的得分：，Private-0.11773，而我们的模型优化后最佳情况得分：Public-0.11188，Private-0.11212，可以看到使用模型预测的误差要远远小于基准阈值。

我们模型的最终目的是通过更加合理、准确的预测，帮助商店经理更好、更专注的进行人员、促销活动等的安排，提高效率以及销售额，那么事实证明我们的模型相比较基准阈值确实有很大提升，不管是更低的误差率还是更快的预测速度，因此我们的模型确实解决了问题，或者说一定程度的解决了问题，当然还是有很大的提升空间，即更准确的预测。

V. 项目结论

结果可视化

基准模型在 kaggle 的表现：0.11773。

优化参数后的 XGBoost 模型的表现：

加入统一校正系数后的表现：

分别进行校正后的表现：

全数据+上述所有优化的表现：

可以看到，在没进过验证集验证的情况下使用全数据训练的结果远不如使用验证集验证后的效果好（主要是参数无法调优）。

对项目的思考

项目的整体流程：

1. 数据读取：读入 *kaggle* 提供的训练、测试、商店数据。
2. 数据异常值分析、处理：对于 *store* 和 *test* 的异常数据进行了针对处理。
3. 数据链接：分别将 *trian* 和 *test* 与 *store* 链接到一起进行后续工作。
4. 枚举字段统一映射：将各种枚举字段的取值都映射到数值型的 *0,1,2,3...* 上。
5. 特征工程：主要是提取新特征，比如年、月、日、对手开张时长、是否促销月、是否周末等。
6. 基准阈值设定。
7. *XGBoost* 模型构建、训练、评分。
8. 参数调优。
9. 校正系数。
9. 全数据训练。
10. 最终模型在测试集上预测，并将结果整理成 *kaggle* 所需格式上传。

其中特征工程是最有趣的环节，一个个脑海中的想法被各种可视化验证有效或者无效，很多认为理所当然的想法，比如销售额跟假期相关被认证是关系不大的，比如竞争对手越远销售额越高，也被证实是错误的，而一些出乎意料的特征却跟预测结果关系很大，比如销售额是有周期性的，也就是日期强相关，竞争对手越近，销售额反而越高等等，这真的是一个很相像的挖掘过程，一铲子下去出来的是泥土还是黄金，没人知道。

最困难的部分应该是模型调参、优化的部分，毕竟在确定好适合的迭代次数（5000）之后，每次运行时间都比较长，而大部分运行的结果表现优化手段没有起到作用，因此这部分需要对算法非常了解，对运行结果非常敏感，才能做出针对性调整，否则就是做一堆无用功不说，结果没有任何提升。

还有一个比较困难的部分我认为是数据处理不彻底，比如很多数据上的瑕疵是在训练时，或者验证后才发现，而一旦数据有问题，那就意味着整个流程都要从头再来一遍，这也是一个相当大的工程，这一点也说明了前期工作的重要性。

总的来说最终结果是符合我的预期的，毕竟在有限的时间内，这个结果已经很不错了，通过这个模型在这个项目上的表现，我们可以判断同类的场景下的类似问题，都可以通过 XGBoost 为主，辅以各种优化手段来达到预测目的。

需要作出的改进

0．整个执行流程中的中间变量一定要单独保存下来，不然丢失在茫茫的内存中再找可就要再执行一遍。
1．最好应该将数据的处理、分析、特征挖掘以及最后的模型相关的操作分开来，这样既能避免每一大部分之间互相影响，又能更好的针对每一个步骤来选择工具和方法，比如对于数据的处理，目前同样是在 Jupyter 中使用 Python 完成，但是根据 Kaggle 第一名的分享得到的灵感，其实这部分用 SQL 来解决更加遍历，一个是 SQL 语句非常灵活、快速，另一个是数据库本身支持很多数据相关的操作，更加方便我们对数据进行分析处理。
2．更重视特征工程阶段，最开始做项目时，主要精力放在模型上，必须模型的选择、超参数的设置、各种优化手段等等，但是对数据的特征挖掘做的很少，最终 RMSPE 分数在 0.24 左右就很难再降低了，后面同样是根据 kaggle 第一名的分享，以及各种网上大神的分享中，开始慢慢重视特征挖掘过程，开始从 Date、促销活动、对手商店等各个基础字段中挖掘更深层次的、对预测结果影响更大的特征出来，最后加入这些特征做预测时，模型没有经过太大预测的情况下在验证集上即获得了 0.102 的好成绩，虽然在 kaggle 中一开始只得到了 0.16，但是已经比之前经过各种优化的模型还要好了，也证明了好的训练数据源对结果的影响远远大于各种 trick 的优化（同样过度的优化存在过拟合的问题）。
3．根据数据选模型，而不是根据模型改数据，根据我们数据的特点、期望答案、挖掘出的特征特点等等，我们最终选择了比较适合的 XGBoost 算法，而不是像一开始的直接定了 Adaboost，但实际上 Adaboost 在该项目上并不是那么合适，其中最重要的是我们的评估函数 RMSPE 可以直接在 XGBoost 上应用。
4．要更加重视前期调研工作，包括项目背景了解、前人们的经验分享等等，都会对我们做项目提供非常大的帮助，也能帮助我们少走很多弯路，很多错误是没有必要重复犯的，当然了踩过的坑可能印象更深刻，也是有好处的。
5．应用统一标准的归一化到各个数值性字段上，项目中没有应用归一化的主要顾虑在于各个数值型数据的最大最小值无法判断估计，担心贸然的归一化处理带来未知的问题。
6．使用 CNN 实现销售额预测，相比较而言，CNN 由于其强大的抽象能力，在特征表达上要更加优秀，不知道与我们的 XGBoost 对比会有怎样的结果，当然，由于各种原因，此次项目中没有进行这样的对比很遗憾，期望后续可以完成。
7．竞争对手的信息挖掘，我认为只是挖掘了对手的开张营业时间是不够的，或许可以根据它的 Distance 进行对手细分，比如每 5km 表示一类对手，可想而知，如果对手超过 10km，或许跟我们的营业额就没有直接关系了，类似 kaggle 泰坦尼克幸存率预测项目中的年龄处理，并不是直接当做枚举进行 OneHot，也不是当做数值型数据，而是根据年龄段，以及社会上常用的年龄分为幼儿、少年、青年、中年、老年等 4 个字段，更符合实际项目需求，事实也证明这样处理效果更好，同时不会造成维度灾难。
8．滑动窗口统计时间序列数据。

收获

相比较学习过程中的收获，在该项目上的收获要更加的接地气：

1. 刻意的保持验证集和最终测试集数据的大小一致（保证训练数据量的前提下），有助于模型更好的泛化到测试集上。

2. 要深入领域去了解数据、挖掘数据。

3. 妥善利用时序数据这一特点。

4. 特征工程决定了模型的上限，调参只是逼近这个上限。

5. 数据处理要小心，毕竟就训练本身而言，过程并不直观，很多时候无法从过程中的一些打印来判断是否数据处理有误（最开始数据划分前没有按 Date 排序，导致模型的 train loss 很低，valid loss 很高，因为 valid data 全是某几个商店的，而现在我们直到，Store 是很重要的特征）。

6. 某些字段被挖掘后依然有存在的价值，比如 Promo2SinceYear 等，不要贸然去掉。

7. 大神跟菜鸟的区别更多在于对数据的理解，而不是算法的使用。

8. 外部数据也可以尝试，比如那段时期的当地天气数据、流行疾病数据等等。

参考文献

【1】XGBoost 官方文档：https://xgboost.readthedocs.io/en/latest/

【2】XGBoost 参数优化：https://blog.csdn.net/aicanghai_smile/article

【3】Pandas 官方文档：http://pandas.pydata.org/pandas-docs/stable/api.html

【4】Kaggle 项目地址：https://www.kaggle.com/c/rossmann-store-sales