项目报告

王琰

Update Log

2019/4/15– 创建文档

目录

项目背景

Rossmann 在 7 个欧洲国家经营着 3,000 多家药店。 目前，Rossmann 商店经理的任务是预先提前六周预测他们的日常销售。 商店销售受许多因素的影响，包括促销，竞争，学校和州假日，季节性和地方性。 成千上万的个体经理根据他们独特的情况预测销售情况，结果的准确性可能会有很大差异。因此需要我们根据现有的数据构造一个销售额预测模型，来帮助经理们更好的安排工作

问题描述

我们需要根据 Rossmann 药妆店的信息（比如促销，竞争对手，节假日）以及在过去的销售情况，来预测 Rossmann 未来的销售额。这是一个典型的机器学习有监督的回归问题，相关算法模型已经非常成熟，同时拥有 Rossmann 提供的大量数据保证了这一问题是可以解决的。

针对位于德国的 1115 家店进行 6 周的销售额预测，销售额是一个数值型数据，所需数据 kaggle 已经提供，其中包含训练数据、商店数据、测试数据，根据评价指标，也就是预测值与实际值的差值越小越好

数据的输入

本项目数据集来源于 kaggle，数据本身是针对该项目收集的数据，所以可以使用他们来进行模型的训练。除此之外，可以先对现有的数据进行数据分析，对整体的数据有一个认知。

评估标准

kaggle 官方的测评指标， Root Mean Square Percentage Error (RMSPE)，其计算方法为：

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

该误差用于描述预测值与实际值之间的偏差，应用于本项目即为预测销售额与实际销售额之前的偏差，可以很好的评估模型的性能。

公式本意就是计算所有预测的数值与实际值之间误差的百分比形式的平均值来衡量模型性能，该值越小越好，如果是 0，那么就表示预测值与实际值一致

基准模型

基准模型可以直接采用平均值来进行评定。可以通过建模与基准进行对比，从而体现模型的优劣。

通过对训练集求平均得到：6954.82295

则此数值用于对所有店铺的预测。

对基准模型计算得到：RMSPE = 0.43923

可以看到直接用均值预测的情况下，误差率还是很高的

项目设计

项目流程

1. 获取数据

2. 数据初步处理，解决数据中的异常值、空值等问题

3. 数据初步分析。根据数据的结构，选择一些方向并提取对应的内容来分析问题，例如分析销售额与天数的变化、销售额与店铺类型的变化等等，先对数据有一个初步的认识

4. 数据的特征处理。对数据进行模型训练前的进一步处理，例如 one-hot 编码、去除无用特征等等

5. 选择模型，划分数据集，训练模型

6. 利用训练结果查看预测的效果

7. 模型的参数调优

8. 上传 kaggle 获得评测结果

9. 项目总结

项目模型

XGBoost：相比较 Adaboost，继承自 GBDT 的 XGBoost 采用梯度的方式来评估模型的不足，支持更多的的目标函数 ，同时相比较 GBDT，支持并行，速度更快，可调参数更多。但 xgboost 也有自己的缺点，例如： 在每一次迭代的时候，都需要遍历整个训练数据多次。如果把整个训练数据装进内存则会限制训练数据的大小；如果不装进内存，反复地读写训练数据又会消耗非常大的时间。其次他的预排序方法空间消耗大，这样的算法需要保存数据的特征值，还保存了特征排序的结果（例如排序后的索引，为了后续快速的计算分割点），这里需要消耗训练数据两倍的内存。

Lightgbm：LightGBM 是一个梯度 boosting 框架，使用基于学习算法的决策树。它可以说是分布式的，高效的。LightGBM 使用 histogram based 算法，将连续特征（属性）值装进离散的容器中，从而加速训练流程并减少内存使用。

本次项目中，打算使用使用 xgboost 模型。Xgboost 模型的原理是非常值得探讨的。其实对于每一个模型而言，都会存在一个目标函数，这个目标函数存在的意义就是指导模型前进的方向。其中，xgboost 的目标函数为：

$$L(\phi) = \sum_i l(y_i, \hat{y}_i) + \sum_k \Omega(f_k), \quad \text{其中} \quad \Omega(f_k) = \gamma T + \frac{1}{2}\lambda ||w||^2$$

显然，等式右边第一项，为损失函数，第二项为正则项。损失函数用于描述模型拟合数据的程度， 正则项用于控制模型的复杂度。这也是 GBDT 和 xgboost 不同的地方。GBDT 是没有正则项的。

这里再回到模型的基础本身。该模型的基础思想是选择树模型作为学习器，通过不断地添加树，不断地进行特征分裂来生长一棵树，每次添加一个树，其实是学习一个新函数，去拟合上次预测的残差。那么这个树的基础概念为：

$$\hat{y}_i^0 = \text{常数}$$

$$\hat{y}_i^1 = \hat{y}_i^0 + f_1(x_i)$$

$$\hat{y}_i^2 = \hat{y}_i^1 + f_2(x_i)$$

$$\hat{y}_i^K = \hat{y}_i^{K-1} + f_K(x_i)$$

那对于第 k 个树，它的目标函数为:

$$Obj^K = \sum_i L(y_i, \hat{y}_i^K) + \Omega(f_K) + constant$$

$$==> Obj^K = \sum_i L\left(y_i, \hat{y}_i^{K-1} + f_K(x_i)\right) + \Omega(f_K) + constant$$

从这里可以看出，只要找到一个合适的 fk 使得对应的目标函数越小，然后不断的迭代 K 次就可以完成 K 个学习器的训练。所以，现在的目标就变成了，如何找到一个 fk 能够最小化目标函数。最小化目标函数有两个方面，第一，最小化损失函数，第二，最小化正则。

Xgboost 对最小化损失函数的思路是，对损失函数利用泰勒二阶展开，

$$\sum_i L\left(y_i, \hat{y}_i^{K-1} + f_K(x_i)\right) = \sum_i \left[L(y_i, \hat{y}_i^{K-1}) + L'(y_i, \hat{y}_i^{K-1}) f_K(x_i) + \frac{1}{2} L''(y_i, \hat{y}_i^{K-1}) f_K^2(x_i)\right]$$

其中 gi 为一阶导数，hi 为二阶导数。用到二阶导数的原因是仅仅用一阶导数的问题就是, 我们无法保证我们找到的是全局最优。

对正则最小化的思路是叶子节点个数 T 要小，对叶子节点加上惩罚值。（叶子节点个数少，树就简单）这也是为什么一开始提到，正则项可以控制模型的复杂度，因为目标就是最小化正则项，而正则项最小是思路就是简化树。而一棵树可以用其叶子节点进行表达，最后化简可得目标函数：

$$\sum_{j=1}^{T} \left[\left(\sum_{(i \in I_j)} g_i\right) w_j + \frac{1}{2}\left(\sum_{(i \in I_j)} h_i + \lambda\right) w_j^2\right] + \gamma T$$

确定目标函数后，接下来就是寻找 fk，即建树的过程。在 xgboost 里面，建树分裂准则是直接与损失函数挂钩的准则。因为 xgboost 会计算增益，

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

在分裂的过程中，左右分裂来选择一个损失函数减少最多的分支。

这样，整个 xgboost 就运行起来了，确定了目标函数，目标函数和建树过程挂钩，那么整个树就以目标函数为核心进行建立。

最后再总结一下 xgboost 的特点：

区别：1.xgboost 和 GBDT 的一个区别在于目标函数上。在 xgboost 中，损失函数+正则项。GBDT 中一般只有损失函数。2.xgboost 中利用二阶导数的信息，而 GBDT 只利用了一阶导数, 即在 GBDT 回归中利用了残差的概念。3.xgboost 在建树的时候利用的准则来源于目标函数推导，即可以理解为牛顿法。而 GBDT 建树利用的是启发式准则。4.xgboost 中可以自动处理空缺值，自动学习空缺值的分裂方向，GBDT(sklearn 版本)不允许包含空缺值。

相似点：1.xgboost 和 GBDT 的学习过程都是一样的，都是基于 Boosting 的思想，先学习前 n-1 个学习器，然后基于前 n-1 个学习器学习第 n 个学习器。而且其都是将损失函数和分裂点评估函数分开了。2.建树过程都利用了损失函数的导数信息,。3.都使用了学习率来进行 Shrinkage，从前面我们能看到不管是 GBDT 还是 xgboost，我们都会利用学习率对拟合结果做缩减以减少过拟合的风险。

数据集划分方式

通过参考 kaggle 第一名的计算方法，将训练集的后六周作为验证集。

模型调优方式

模型调优有多种方式，例如模型参数调整、模型融合、数据分布校正等等。本项目打算建立两个模型，进行对比，说明一个模型的局限性以及模型融合的优势。然后再针对一个模型，采用数据分布校正的方式进行调优。

数据初步分析

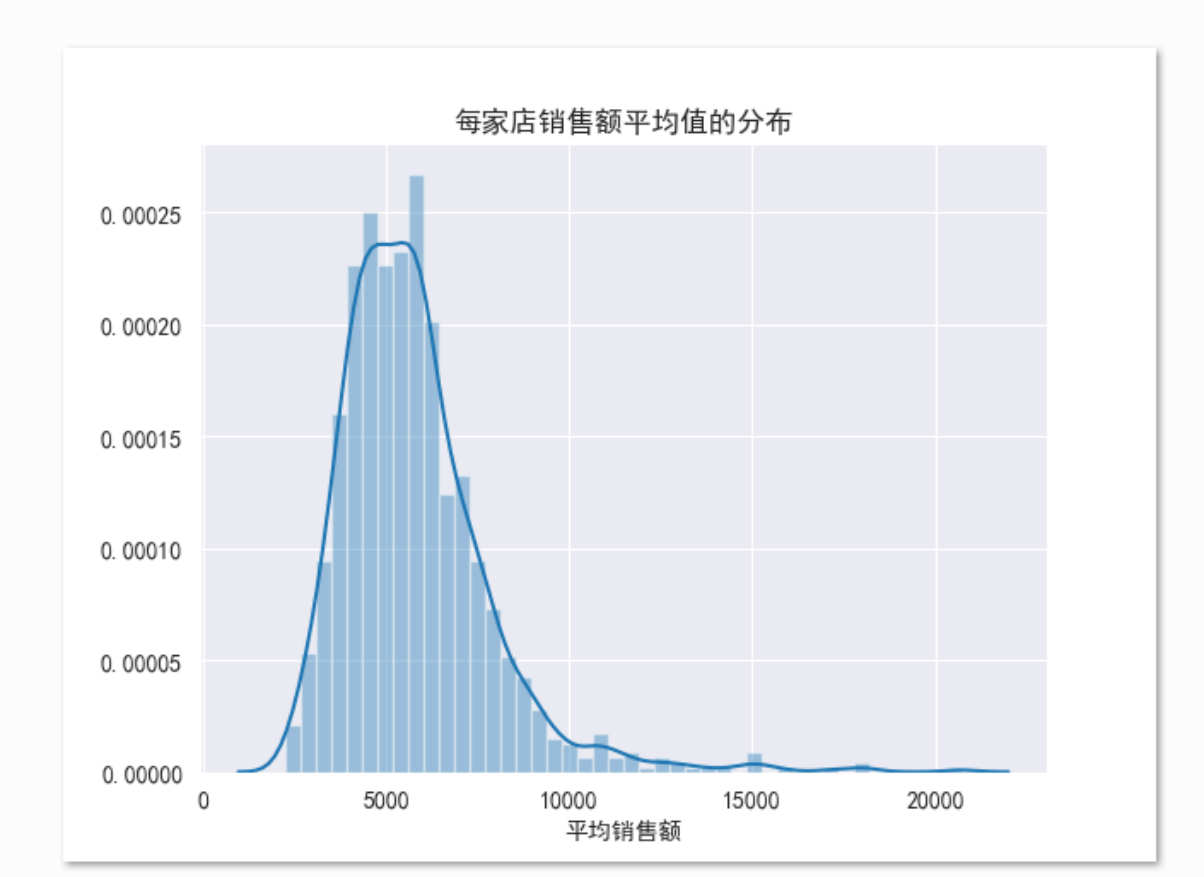
在开始正式项目前，可以先进行项目流程中的 1~3 步，对数据有一个初步的认识，也方便后续模型的选择和调参。

首先对数据进行填充，open 字段用 1 填充，竞争对手字段用平均值填充，填充完毕后，查看对应的填充效果，如图：

```
Data columns (total 10 columns):
Store                         1115 non-null int64
StoreType                     1115 non-null object
Assortment                    1115 non-null object
CompetitionDistance           1115 non-null float64
CompetitionOpenSinceMonth     1115 non-null float64
CompetitionOpenSinceYear      1115 non-null float64
Promo2                        1115 non-null int64
Promo2SinceWeek               1115 non-null float64
Promo2SinceYear               1115 non-null float64
PromoInterval                 1115 non-null object
dtypes: float64(5), int64(2), object(3)
```

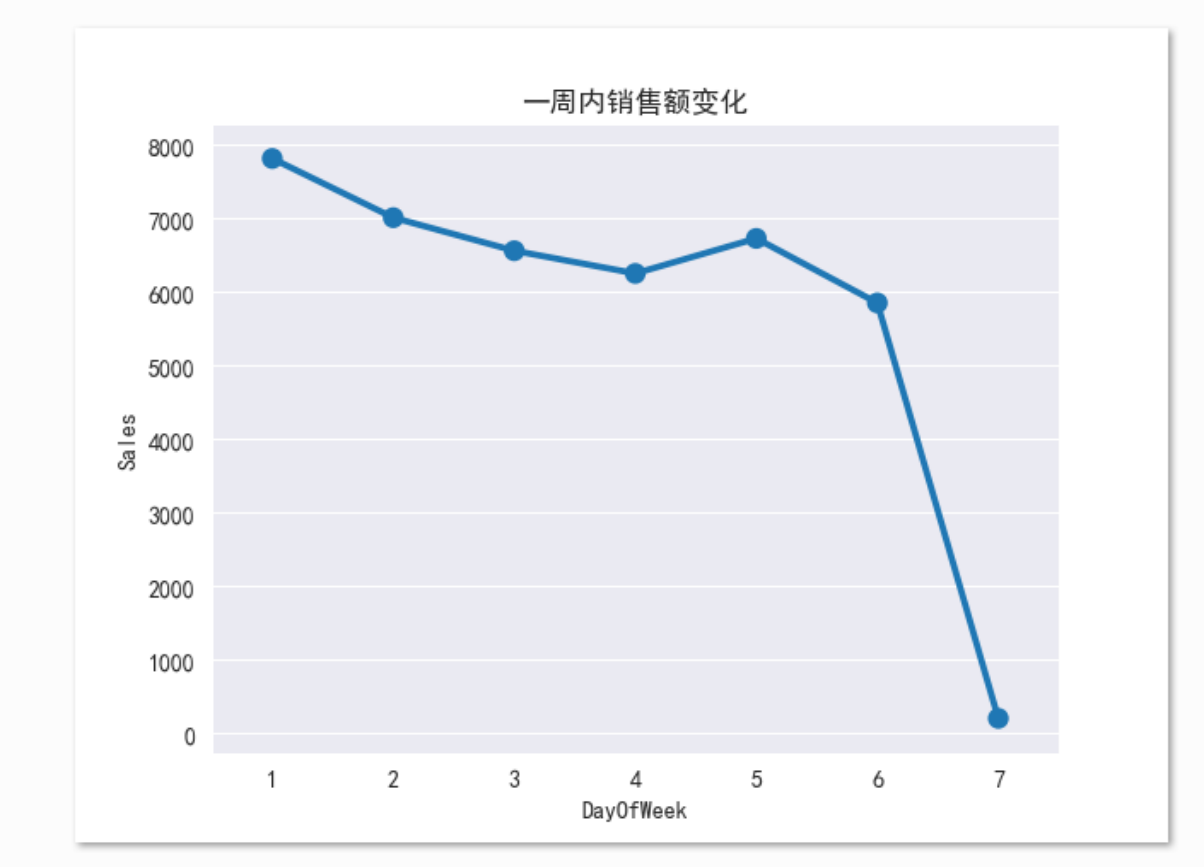
可以看到没有空值了，接下来进行数据分析

1. 销售额分布



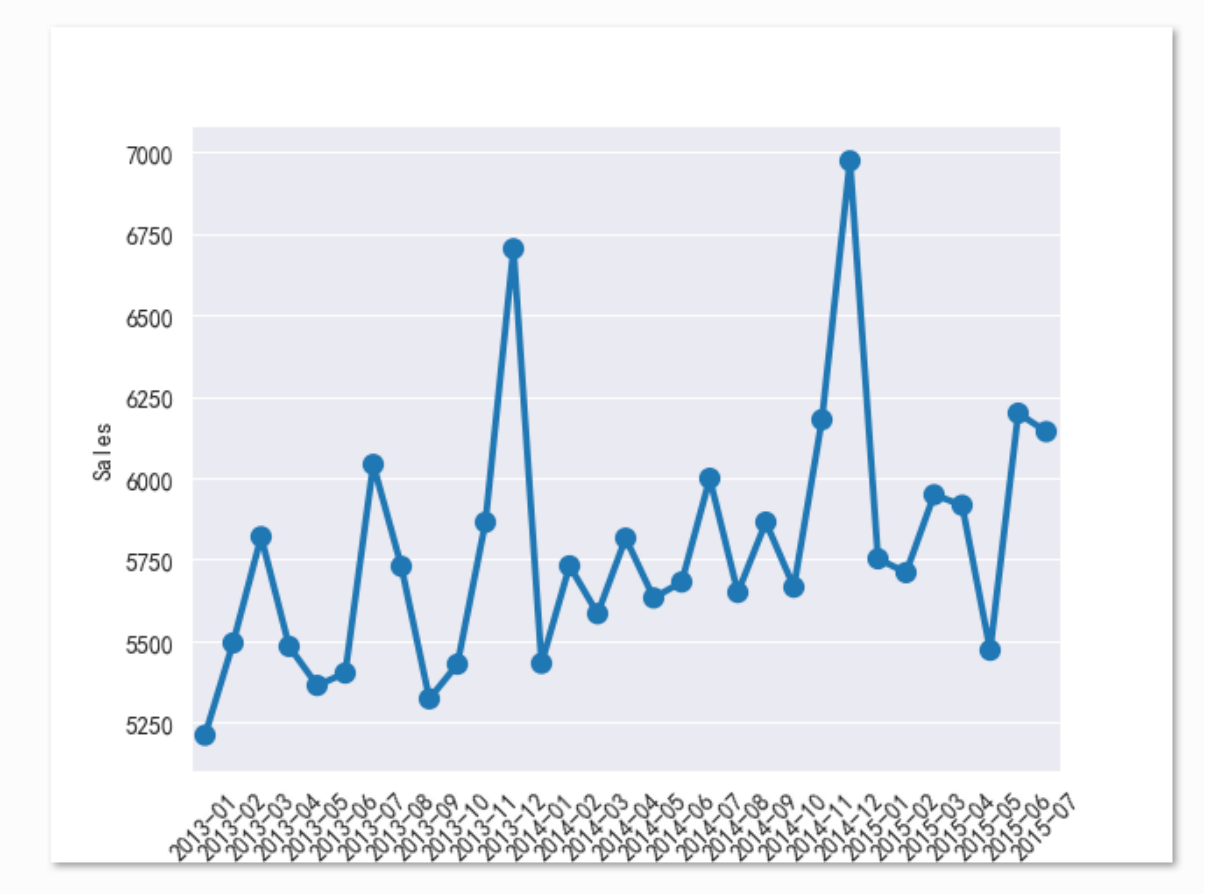
从图中可以看出，店面的平均销售额呈现一种偏态分布，这也符合基本的常识

2. 开始打印销售额随时间变化



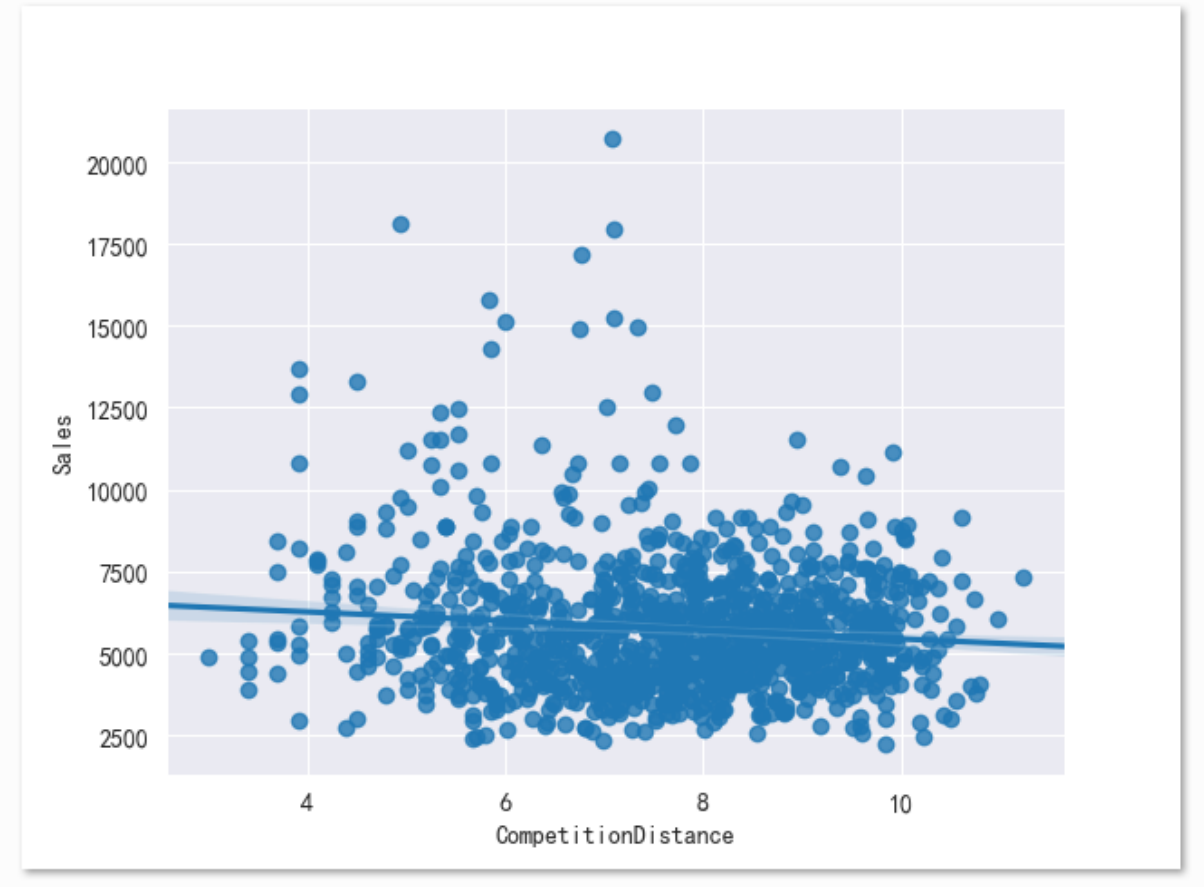
从周一开始持续下降，但是趋势比较缓慢。只有周日的销量立刻下降为接近 0 ，估计很多店铺周日不开门，因此周日积累的消费需求在周一得到大幅度释放。

3. 销售额季节效应



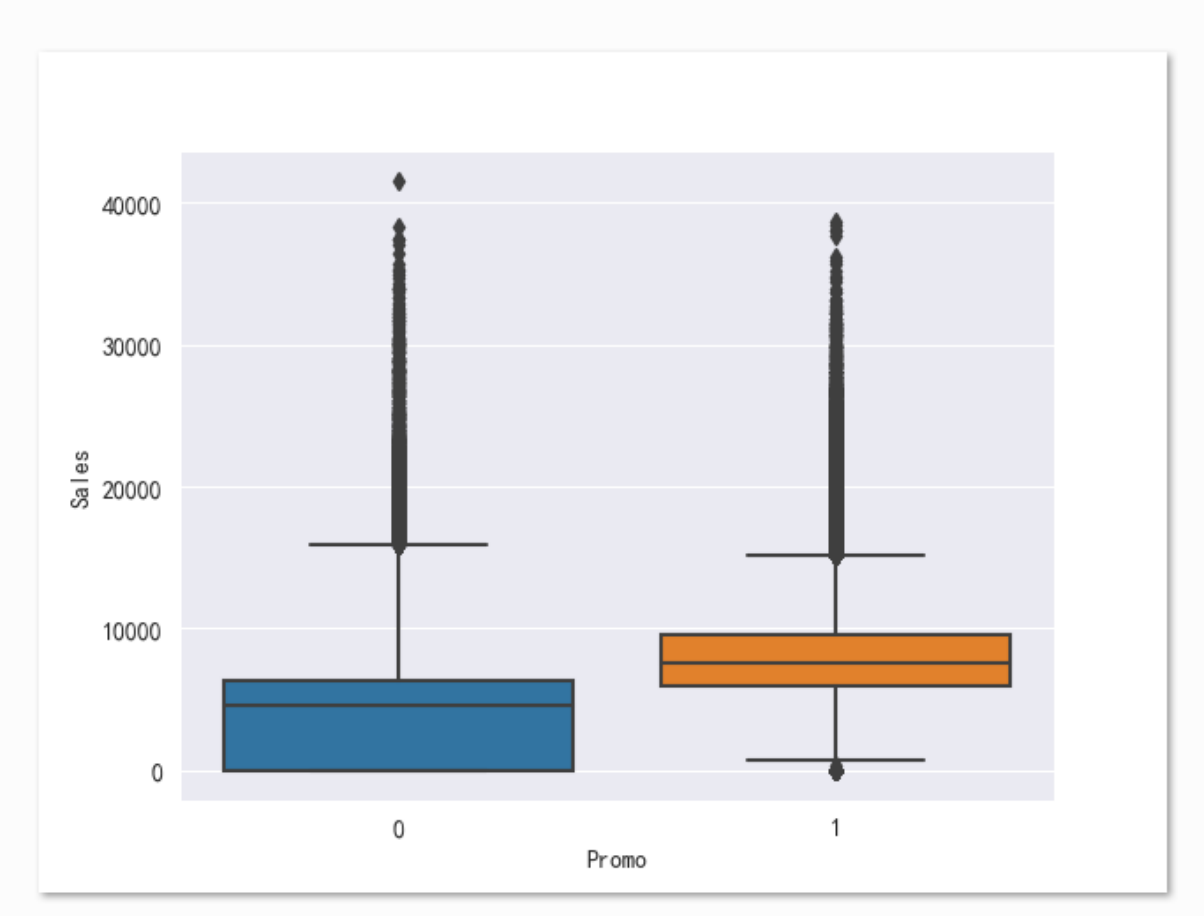
从图中可以看出，每年的 10~12 月是销量最好的时候，可能是圣诞节来临或者天气变冷造成的影响，14 年相比 13 年，整体的销售额水平都有所提升，维持在较高的水平

4. 竞争对手距离与销售额关系



可以看到，竞争对手的距离与销售量并没有非常明显的正反比关系，但是部分店铺在竞争对手距离更近时，能够有更大的销售额，这说明竞争对于销量有一定的促进作用，但不是特别明显

5. 促销与销售额的关系



可以看到有促销与无促销，两者差距还是比较大的。有促销的情况下，基本上销售额肯定会有一定的提升，也符合常识。

第一次建模

在正式建模之前，再重新梳理一下数据的相关结构和内容，便于建模过程中的数据填充和特征处理。

1. 查看数据结构

	Store	DayOfWeek	Date	Sales	Customers	Open	Promo	StateHoliday	SchoolHoliday
0	1	5	2015-07-31	5263	555	1	1	0	1
1	2	5	2015-07-31	6064	625	1	1	0	1
2	3	5	2015-07-31	8314	821	1	1	0	1
3	4	5	2015-07-31	13995	1498	1	1	0	1
4	5	5	2015-07-31	4822	559	1	1	0	1

	Id	Store	DayOfWeek	Date	Open	Promo	StateHoliday	SchoolHoliday
0	1	1	4	2015-09-17	1.0	1	0	0
1	2	3	4	2015-09-17	1.0	1	0	0
2	3	7	4	2015-09-17	1.0	1	0	0
3	4	8	4	2015-09-17	1.0	1	0	0
4	5	9	4	2015-09-17	1.0	1	0	0

	Store	StoreType	Assortment	CompetitionDistance	CompetitionOpenSinceMonth	CompetitionOpenSinceYear	Promo2	Promo2SinceWeek	Promo2SinceYear
0	1	c	a	1270.0	9.0	2008.0	0	NaN	NaN
1	2	a	a	570.0	11.0	2007.0	1	13.0	2010.0
2	3	a	a	14130.0	12.0	2006.0	1	14.0	2011.0
3	4	c	c	620.0	9.0	2009.0	0	NaN	NaN
4	5	a	a	29910.0	4.0	2015.0	0	NaN	NaN

可以看到，3 个数据表都有相同的 store 字段，因此可以用 store 表作为核心来链接两个表。因为很多数据是枚举或者根据时间排序的，所以需要进行组合后/处理后才能使用。

2. 查看数据异常信息

```
<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 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
dtypes: datetime64[ns](1), int64(7), object(1)
memory usage: 69.8+ MB
<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 datetime64[ns]
Open             41077 non-null float64
Promo            41088 non-null int64
StateHoliday     41088 non-null object
SchoolHoliday    41088 non-null int64
dtypes: datetime64[ns](1), float64(1), int64(5), object(1)
memory usage: 2.5+ MB
<class 'pandas.core.frame.DataFrame'>
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                571 non-null object
dtypes: float64(5), int64(2), object(3)
memory usage: 87.2+ KB
```

traindata 里面的数据都很完整没有缺失，不需要做数据的填充处理。测试数据再 open 这个字段有缺失，到时候填充的时候，可以都采用开门。store 中，特征数据缺失严重，主要集中在 6 个特征：

CompetitionOpenSinceMonth——竞争对手开张月份

CompetitionOpenSinceYear——竞争对手开张年份

CompetitionDistance——竞争对手距离

Promo2SinceWeek——开始持续促销的年

Promo2SinceYear——开始持续促销的星期

PromoInterval————促销间隔

说明 300 多个商店没有竞争对手的开张时间，3 个店面是缺少了竞争对手的距离。对应的持续促销，需要看看是时间丢失，还是根本没有采取促销活动

	Store	StoreType	Assortment	CompetitionDistance	CompetitionOpenSinceMonth	CompetitionOpenSinceYear	Promo2	Promo2SinceWeek	Promo2SinceYear
0	1	c	a	1270.0	9.0	2008.0	0	NaN	NaN
3	4	c	c	620.0	9.0	2009.0	0	NaN	NaN
4	5	a	a	29910.0	4.0	2015.0	0	NaN	NaN
5	6	a	a	310.0	12.0	2013.0	0	NaN	NaN
6	7	a	c	24000.0	4.0	2013.0	0	NaN	NaN
7	8	a	a	7520.0	10.0	2014.0	0	NaN	NaN
8	9	a	c	2030.0	8.0	2000.0	0	NaN	NaN
9	10	a	a	3160.0	9.0	2009.0	0	NaN	NaN
15	16	a	c	3270.0	NaN	NaN	0	NaN	NaN
22	23	d	a	4060.0	8.0	2005.0	0	NaN	NaN

可以看到前 10 条店铺信息，他们的 promo2 全是 0，说明完全没有参与持续促销，因此没有促销时间是正常的情况。观察到一个额外的信息，竞争对手距离不为 null，但开张时间为 null，说明有些店铺对竞争对手的数据丢失了，但竞争对手是客观存在的。

3. 填充数据

因为 test 中只有 open 数据缺失，因为没有 open 数据则对应的销售额即为 0，要关联到 store 中则量级会成倍增加，因此填充为 1。竞争对手先采用平均值来填充。

```
store_data_ana['CompetitionDistance'] = store_data_ana['CompetitionDistance'].fillna(store_data_ana['CompetitionDistance'].mean())
store_data_ana['CompetitionOpenSinceMonth'] = store_data_ana['CompetitionOpenSinceMonth'].fillna(method='pad')
store_data_ana['CompetitionOpenSinceYear'] = store_data_ana['CompetitionOpenSinceYear'].fillna(method='pad')
store_data_ana = store_data_ana.fillna(0)
```

填充完毕后利用 merge 函数进行数据合并即可

4. 数据集划分

前面提到，参考 kaggle 第一名的做法，将数据集后六周作为验证集。

5. 特征处理

在数据初步分析中我们看到，销售额随着时间的变化非常剧烈，因此考虑到时间特征可能是该销售额的一个重要影响因素，因此第一次建模可以将时间维度进行特征处理，先来看看效果。与此同时，因为某些字段的值属于枚举类型，需要进行重新编码，因此特征处理方式如下：

```
def features_edit(data):
    # 将存在其他字符表示分类的特征转化为数字
    mappings = {'0': 0, 'a': 1, 'b': 2, 'c': 3, 'd': 4,}
    data.StoreType.replace(mappings, inplace=True)
    data.Assortment.replace(mappings, inplace=True)
    data.StateHoliday.replace(mappings, inplace=True)
    data['Year'] = data.Date.dt.year
    data['Month'] = data.Date.dt.month
    data['Day'] = data.Date.dt.day
    data['DayOfWeek'] = data.Date.dt.dayofweek
    data['WeekOfYear'] = data.Date.dt.weekofyear
```

6. 建模训练

设置模型参数，开始训练模型。最终，模型的结果记录为：

```
Stopping. Best iteration:
[62]    train-rmse:2251.01      valid-rmse:2287.01      train-RMSPE:0.318218    valid-RMSPE:0.282952
```

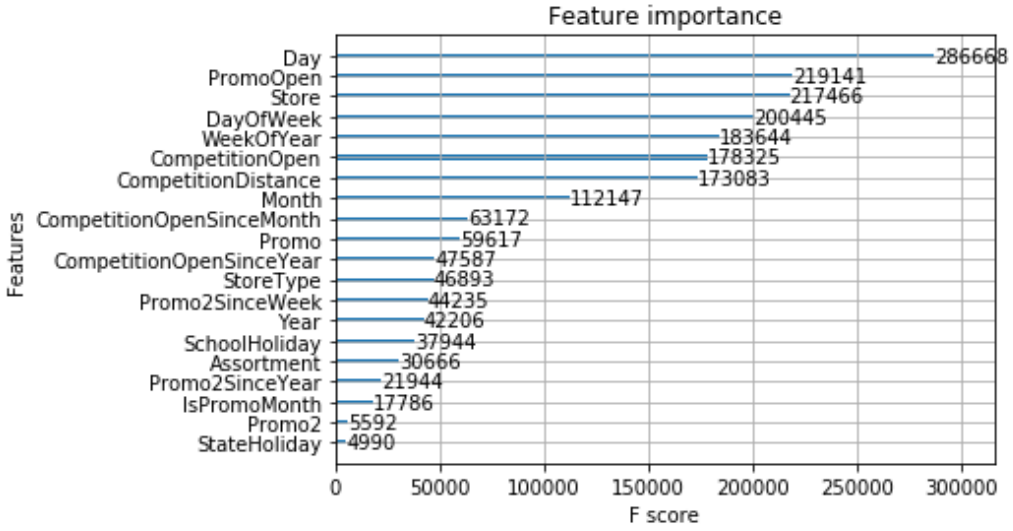
设置了最大的 num_boost_round_edit=6000，early_stopping_rounds=100，最后的 rmspe=0.16，发现该模型的上限 并不高，预测的准确率比较低。在验证集上进行验证，得到的结果为：

RMSPE: 0.295833

提交給 kaggle 后，获得的得分为

Submission and Description	Private Score	Public Score	Use for Final Score
Rossmann_submission_3.csv a few seconds ago by evilstata add submission details	0.19465	0.18673	☐

查看模型的特征重要程度，发现时间、促销开启是核心，其次才是竞争对手等。



第二次建模

在第一次建模的基础上进行优化改进。

1. 填充数据

发现竞争对手距离与销售额的关系并没有简单的可描述的关系，同时部分竞争对手的距离还会导致出现异常偏高的值，因此本次将竞争对手相关的数据填充为 0

2. 特征处理

考虑第一次建模时特征的重要程度，因此本次将竞争对手、促销等特征，也做特征化处理，以此来构建模型特征。

```
#编辑特征，进行独热编码，同时将竞争对手、促销信息也作为特征之一
def features_edit(data):


    mappings = {'0':0, 'a':1, 'b':2, 'c':3, 'd':4}
    data.StoreType.replace(mappings, inplace=True)
    data.Assortment.replace(mappings, inplace=True)
    data.StateHoliday.replace(mappings, inplace=True)


    data['Year'] = data.Date.dt.year
    data['Month'] = data.Date.dt.month
    data['Day'] = data.Date.dt.day
    data['DayOfWeek'] = data.Date.dt.dayofweek
    data['WeekOfYear'] = data.Date.dt.weekofyear



    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['CompetitionOpen'] = data.CompetitionOpen.apply(lambda x: x if x > 0 else 0)
    data['PromoOpen'] = data.PromoOpen.apply(lambda x: x if x > 0 else 0)



    month2str = {1:'Jan', 2:'Feb', 3:'Mar', 4:'Apr', 5:'May', 6:'Jun',7:'Jul', 8:'Aug', 9:'Sept', 10:'Oct', 11:'Nov', 12:'Dec'}
    data['monthStr'] = data.Month.map(month2str)
    data.loc[data.PromoInterval == 0, 'PromoInterval'] = ''
    data['IsPromoMonth'] = 0
    for interval in data.PromoInterval.unique():
        if interval != '':
            for month in interval.split(','):
                data.loc[(data.monthStr == month) & (data.PromoInterval == interval), 'IsPromoMonth'] = 1

    return data
```

3. 建模训练

设置模型参数，开始训练模型。最终，模型的结果记录为：

```
Stopping. Best iteration:
[3085]  train-rmse:0.066367     eval-rmse:0.117232      train-rmspe:0.070345    eval-rmspe:0.127409
```

可以看到，相比于第一次，有很大的提升。在验证集上进行验证，得到的结果为：

RMSPE： 0.127452

提交給 kaggle 后，获得的得分为

Submission and Description	Private Score	Public Score	Use for Final Score
Rossmann_submission_1.csv a minute ago by evilstata first	0.12796	0.12548	☐

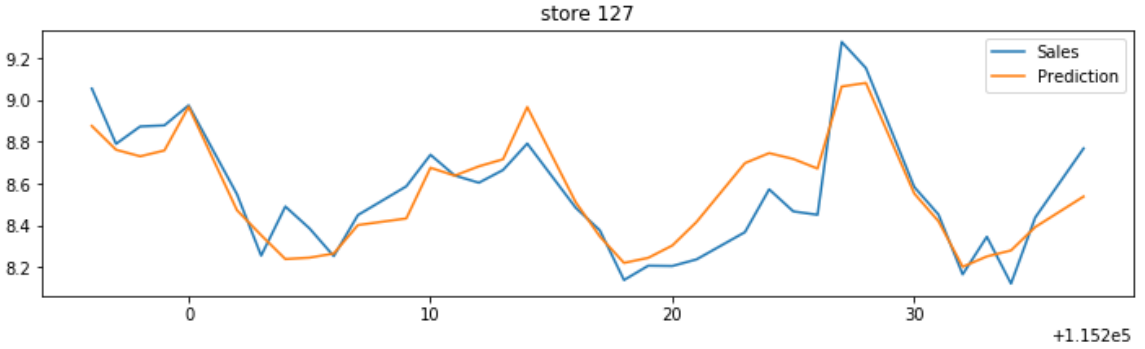
可以看出，不同的模型，其各自的上限和效果都不同，因此后面的模型优化，不仅可以考虑模型自身调参，也可以考虑模型融合，即选择各个模型的优势来进行建模。本次则针对进行修改。

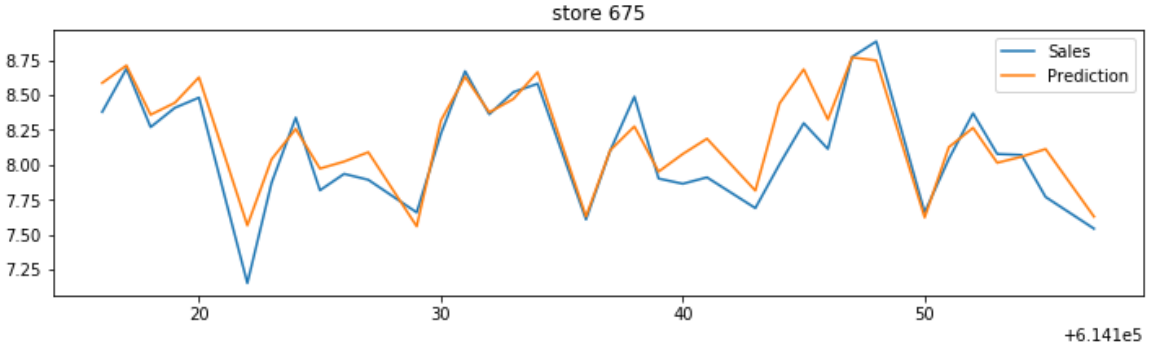
模型优化

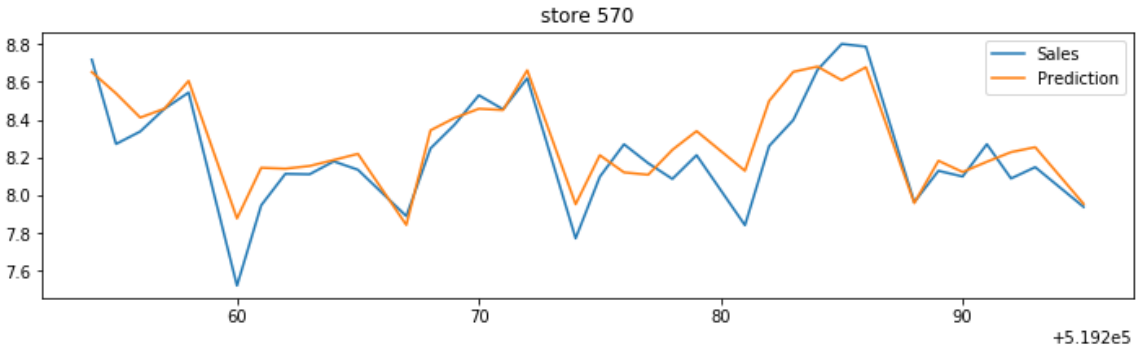
首先，统计模型在验证集上的一些相关信息，再确定优化手段。将预测值、实际值放在一个表格中，并计算出对应的偏差率。

SinceMonth	CompetitionOpen	SinceYear	...	Day	WeekOfYear	CompetitionOpen	PromoOpen	IsPromoMonth	Sales	Prediction	Ratio	Error	Weight
9.0		2008.0	...	31	31	82.0	24187.75	0	8.568646	8.581182	1.001463	0.001463	0.998539
9.0		2008.0	...	30	31	82.0	24187.75	0	8.521384	8.531733	1.001214	0.001214	0.998787
9.0		2008.0	...	29	31	82.0	24187.75	0	8.472823	8.470778	0.999759	0.000241	1.000241
9.0		2008.0	...	28	31	82.0	24187.75	0	8.519590	8.488213	0.996317	0.003683	1.003697
9.0		2008.0	...	27	31	82.0	24187.75	0	8.716536	8.571804	0.983396	0.016604	1.016885

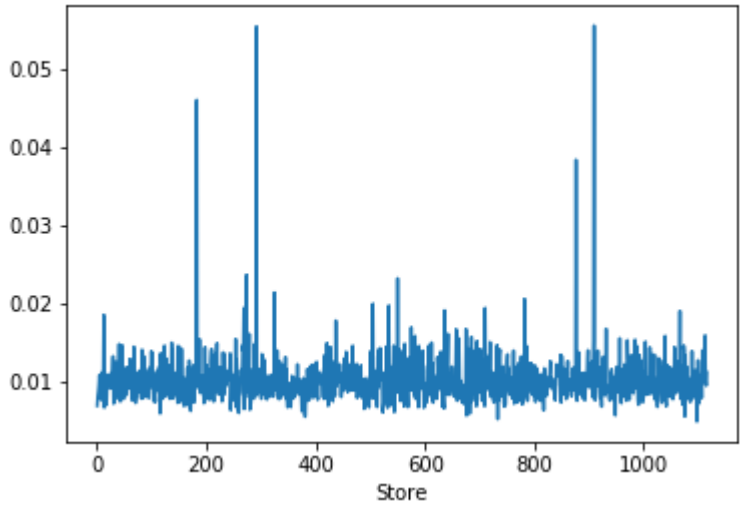
任意选择 3 个实际店铺，查看对应的预测关系曲线：







可以看到，预测值和实际值之间还是有较大差别的 再看看每个店铺的相对偏差，如图所示：



发现波动非常明显，如图进行数据校正时，统一采用一个值，虽然能起到效果，但是肯定不如针对每个店铺做调整。因为我们是可以计算出每个店铺对应的偏差的。将每个店铺的预测值都乘以偏差校准率以后，计算得到 `RMSPE 0.114313`。

可以看到，下降了 0.013 在最后的测试集上来进行预测，并提交 kaggle，得分：

Submission and Description	Private Score	Public Score	Use for Final Score
Rossmann_submission_2.csv just now by evilstata 2nd	0.11329	0.11114	☐

总结

1. 通过两次建模，可以看出，不同的模型，由于特征的不同，其最终的效果会有很大的差别
2. 同一个模型，除了对模型的超参进行不断的调优以外，还可以通过数据分布校正的方式，对预测值进行校正，而且该效果可能非常显著。
3. 同一个模型的效果一定存在着其上限，因此，通过模型融合，取长补短，可以更大化的提高模型的准确性。但相对应的，会增加大量的时间，因为对每个模型都需要进行训练。本项目由于时间关系，先做校正的效果，等之后有时间可进一步尝试模型融合。

参考资料：

LightGBM 介绍：

https://mp.weixin.qq.com/s?__biz=MzA3MzI4MjgzMw==&mid=2650719786&idx=3&sn=ab1c5a77237dc

4b2ee5ae12c7a68ff87&chksm=871b0254b06c8b42d5a4fdf3327f7284c9ffbe72fe7911301d368b157024b32923d88401c2a8&scene=0&open_source=weibo_search

Lightgbm 介绍：https://github.com/Microsoft/LightGBM/

本项目 Kaggle 第一名：https://www.kaggle.com/c/rossmann-store-sales/discussion/18024

Xgboost 介绍：https://zhuanlan.zhihu.com/p/25308051

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

Xgboost 详解：https://www.jianshu.com/p/ac1c12f3fba1

Xgboost **原理分析：**https://blog.csdn.net/qq_22238533/article/details/79477547